TONY BAUTTS,
TERRY DAWSON, GREGOR N. PURDY

LINUX

HÁLÓZATI ADMINISZTRÁTOROK KÉZIKÖNYVE

KOSSUTH KIADÓ

Tartalom

Előszó

Az internet a földkerekség legtöbb országában jól ismert és használt kommunikációs rendszer és az üzleti világban is egyre inkább a mindennapok részévé válik. Nap mint nap milliók kapcsolódnak a világhálóra, és a számítógépes hálózatok elterjedtsége a tv-készülékekével és a mikrohullámú sütőkével vetekszik. Nem kíván nagyobb erőfeszítést egy vezeték nélküli jelosztó beszerzése és telepítése sem. Az internet szokatlanul nagy területét teszi ki a médiának: számos olyan webhelyet találunk, amelyik kifejezetten „vadászik" a hagyományos médiahírforrások szenzációt ígérő friss híreire, és a virtuális valóság típusú környezetek, például az on-line játékok és más hasonló számítógépes eszközök ugyancsak mélyen beépültek az „internetes kultúrába".

Természetesen a hálózatkezelés már régóta ismert fogalom a számítógépes rendszerekben. Az egymással összekapcsolt számítógépekből álló lokális hálózatok ugyananynyira elterjedtek a kisebb kommunikációs infrastruktúrákban, mint a méretes távközlési vállalatok nagy távolságú átviteli kapcsolatait használó rendszerekben. A világméretű hálózatokat magában foglaló, gyorsan növekvő konglomerátum szinte mindenki számára indokolttá teszi az internetes csatlakozást, aki számítógép közelébe kerülhet. A gyors levelezést és webes hozzáférést nyújtó nagy sávszélességű internetes szolgáltatások mindinkább megfizethetők.

A számítógépes hálózatokról szólva gyakran a Unixra gondolunk. Természetesen a Unix nem az egyetlen operációs rendszer, amely a hálózatkezelést lehetővé teszi, és éllovas helye sem garantált az idők végezetéig, de kétségtelen, hogy már régóta jelen van a hálózatkezelésben, és jó ideig meglesz a maga szerepe. Az egyéni felhasználók számára a Unix azért érdekes, mert számos erőfeszítésről tudunk, amelyek azt célozták, hogy ingyenes, Unix-szerű operációs rendszerek jelenhessenek meg személyi számítógépre, ilyenek például a NetBSD, a FreeBSD és a Linux.

A Linux voltaképpen szabadon terjeszthető Unix-klón személyi számítógépekre. Jelenleg sokféle számítógépen fut, többek között az Intel processzorcsaládon, de a PowerPC architektúrákon is, például az Apple Macintosh gépeken; mindemellett futtatható a Sun SPARC és az UltraSPARC gépeken; a Compaq Alpha gépeken; a MIPS gépeken, sőt még néhány videojáték-konzolon is: ilyen a Sony PlayStation 2, a Nintendo GameCube vagy a Microsoft Xbox. A Linuxot átírták néhány kevésbé ismert platformra: ilyen a Fujitsu AP1000 és az IBM System S/390. A fejlesztők laboratóriumaiban már készülnek az átírások más érdekes architektúrákhoz is; sikert ígér a Linux például a beágyazott vezérlők területén.

A Linuxot önkéntesekből álló hatalmas csapat az interneten fejlesztette ki. A munkát 1990-ben Linus Torvalds finn egyetemista kezdte el, aki eredetileg tanulmányi feladatként kezelte egy operációs rendszer megalkotását. Azóta a Linux lavinához hasonlóan hatalmasra növekedett, és teljes tulajdonságkészletű Unix-klónná vált, amely a legkülönbözőbb alkalmazások futtatására képes: szimulációs és modellező programok, szövegszerkesztők, hangfelismerő rendszerek, internetes böngészők jelentek meg hozzá, és rengeteg más program is, például kitűnő játékok. Számos hardver támogatása mellett a Linux a TCP/IP hálózatkezelés teljes implementációját tartalmazza, közöttük a PPP-t, a tűzfalakat és számos más olyan lehetőséget és protokollt, melyeket egyéb operációs rendszereken nem találunk. A Linux nagy tudású, gyors és szabad szoftver, melynek népszerűsége az internetes világban egyre nő.

A Linux operációs rendszert a GNU General Public License szabályozza, ugyanaz a szerzői jogi védelem óvja, amely a Free Software Foundation által fejlesztett programokat. Az engedély bárki számára lehetővé teszi a programok terjesztését és módosítását (ingyenesen vagy pénzért), feltéve hogy minden módosítás és disztribúció továbbra is szabadon terjeszthető. A „szabad szoftver" a felhasználás szabadságára utal, nem a jogdíjmentességre.

A könyv célja és célközönsége

A könyvet önállóan használható referenciamunkának szántuk a Linux-környezet hálózati adminisztrációjához. A kezdő és a tapasztalt felhasználók egyaránt megtalálhatják benne a Linux hálózati konfiguráció kezelésével kapcsolatos szinte összes fontos adminisztratív tevékenység leírását. A tárgyalható témakörök száma szinte végtelen, így természetesen lehetetlen volna valamennyi témáról az összes tudnivalót elmondani. Arra törekedtünk, hogy a legfontosabb és leggyakoribb kérdésekkel foglalkozzunk. A Linux hálózatkezelésben járatlan felhasználók, még azok is, akiknek a Unixhoz hasonló operációs rendszerekkel nem volt korábban dolguk, elegendő segítséget találtak e könyv korábbi kiadásaiban, hogy összeállítsák és elindítsák a saját Linux-hálózatukat, és felkészüljenek újabb ismeretek befogadására.

Számos könyv és egyéb információforrás áll rendelkezésre, hogy a kötet egyes témaköreiről bővebb információkat szerezzünk. Mi is összeállítunk egy bibliográfiát azok számára, akik úgy érzik, szükségük volna további felfedezésekre a Linuxról.

A tudás forrásai

A Linux világában járatlanok számára ajánlunk most néhány hasznos ismeretforrást. Jól jöhet, ha internetes hozzáférésük van, de ez nem elengedhetetlen feltétel.

A Linux Documentation Project kézikönyvek

A Linux Documentation Project (LDP) egy csoport önkéntesekből, akik kézikönyveket, HOWTO dokumentumokat és súgóoldalakat (man page) készítenek különféle témakörökről, a telepítéstől a rendszermag-programozásáig.

Könyvek

Linux Installation and Getting Started
Matt Welsh és rajta kívül több társszerző könyve a Linux beszerzéséről, telepítéséről és használatáról. Tartalmaz egy bevezető szintű Unix-tananyagot is, valamint foglalkozik a rendszeradminisztráció, az X Window System és a hálózatkezelés kérdésével.

Linux System Administrators Guide
Írói Lars Wirzenius és Joanna Oja. Kötetük a Linux rendszeradminisztráció általános kérdéseivel foglalkozik, például a felhasználók létrehozásával és beállításával, a rendszer biztonsági másolatainak készítésével, a főbb szoftvercsomagok beállításával és a szoftverek telepítésével, valamint frissítésével.

Linux System Administration Made Easy
Steve Frampton kötete a Linux-felhasználók számára érdekes mindennapi adminisztrációval és karbantartással foglalkozik.

Linux Programmers Guide
B. Scott Burkett, Sven Goldt, John D. Harper, Sven van der Meer és Matt Welsh kötete a Linux-rendszer alkalmazásfejlesztői számára fontos témaköröket ölel fel.

The Linux Kernel
Írója David A. Rusling. A könyv bevezetés a Linux rendszermagjának, elsősorban a rendszermag felépítésének és működési elvének megismerésébe. Tanulmányozzuk a kernel nevezetességeit!

The Linux Kernel Module Programming Guide
Ori Pomerantz kézikönyvéből megtudhatjuk, hogyan írhatunk rendszermagmodulokat Linuxra. A könyv eredetileg az LDP részeként látott napvilágot. A jelenlegi kiadás szövegére a Creative Commons Attribution-Share Alike License szabályozás érvényes, vagyis szabadon módosítható és terjeszthető.

További kézikönyvek megjelenése várható. Az LDP-ről többet is megtudhatunk a *http://www.linuxdoc.org/* címen, vagy a számos tükröoldal valamelyikén.

A HOWTO dokumentumok

A Linux HOWTO dokumentumok átfogó kiadványsorozatot alkotnak, amely részletesen tárgyalja a rendszer különböző aspektusait, például az X Window System szoftver telepítését és beállítását, vagy az assembly nyelvű programok írását Linuxra; letölthetők az LDP számos tükröoldaláról (lásd néhány mondattal később). Az elérhető dokumentumok listáját az állomány *HOWTO-INDEX* részében találjuk.

Érdemes letölteni az *Installation HOWTO* dokumentumot, amely a Linux telepítését ismerteti; a *Hardware Compatibility HOWTO* dokumentumot, amely a Linuxon bizonyítottan működőképes hardverek listáját tartalmazza; és a *Distribution HOWTO* dokumentumot, amely a Linux-rendszert lemezen vagy CD-ROM-on értékesítő forgalmazók listáját foglalja magában.

A Linux Frequently Asked Questions gyűjtemény

A *Linux Frequently Asked Questions with Answers* (FAQ) a rendszert érintő kérdések és válaszok válogatott gyűjteménye. A kezdők számára feltétlenül ajánlott olvasmány.

Dokumentumok a világhálón

Számos Linux alapú webhelyet találhatunk. A Linux Documentation Project honlapja a *http://www.tldp.org/* címen érhető el.

Ha más felvilágosításra volna szükségünk, valószínűleg elegendő egy gyors keresés a Google keresőben. Úgy látszik, szinte mindent kipróbált már valaki a Linux-közösségből, és jó esélyünk van rá, hogy a tapasztalatait írásban is összefoglalta.

A kereskedelemben kapható kiadványok

Több kiadó és szoftverforgalmazó nyomtatásban jelenteti meg az LDP munkáit. Ilyenek a Specialized Systems Consultants Inc. (SSC) *(http://www.ssc.com)* és a Linux Systems Labs *(http://www.lsl.com)*. Mindkét vállalat értékesít HOWTO dokumentumokból összeállított gyűjteményeket és más Linux-dokumentációkat nyomtatott és kötött formában.

Az O'Reilly Media egy sor Linux-könyvet jelentetett meg. Ezek az LDP munkái, noha a legtöbbjüket önállóan írták:

Running Linux
: Telepítési és felhasználói kézikönyv a rendszerhez, amelyből megtudhatjuk, hogyan hozhatjuk ki a legtöbbet linuxos személyi számítógépünkből.

Linux Server Security
: Kiváló útmutató a Linux-kiszolgálók biztonságos beállításához. A webkiszolgálókat és más, védőbástya-funkciókat betöltő számítógépeket fejlesztő és kezelő rendszergazdák rengeteg hasznos tanácsot találhatnak a könyvben.

Linux in a Nutshell
: A sikeres „in a Nutshell" sorozat egyik kötete, mely elsősorban széles körű referenciát kínál a Linuxhoz.

Linux iptables Pocket Reference
: A Linux tűzfalrendszerének rövid, de teljes áttekintése.

A Linux Journal és a Linux Magazine

A *Linux Journal* és a *Linux Magazine* a Linux-közösség havonta megjelenő lapjai, amelyeket Linux-aktivisták írnak és adnak közre. A cikkek a legegyszerűbb kérdésektől a rendszermag-programozás műhelytitkaiig számos kérdéssel foglalkoznak. Függetlenül attól, hogy van-e Usenet hozzáférésünk, a lapok segíthetnek, hogy állandó kapcsolatot tartsunk fenn a Linux-közösséggel.

A *Linux Journal* régebbi kiadvány, melyet az SSC ad ki – erről bővebben az előző részben olvashattunk. A magazint a *http://www.linuxjournal.com/* oldalon is megtaláljuk.

A *Linux Magazine* újabb független publikáció. A magazin honlapja a *http://www.linuxmagazine.com/*.

Linux Usenet hírcsoportok

Amennyiben hozzáférünk a Usenet hírekhez, a következő Linux vonatkozású hírcsoportokat olvashatjuk:

comp.os.linux.announce
: Moderált hírcsoport, amely új szoftvereket és disztribúciókat ismertet, felderített hibákat mutat be és beszámol a Linux-közösség aktuális eseményeiről. A Linux-felhasználóknak mindenképpen érdemes olvasgatni.

comp.os.linux.help
: Általános kérdések és válaszok a Linux telepítéséről és használatáról.

comp.os.linux.admin
: A Linux rendszerek rendszeradminisztrációs kérdéseivel foglalkozik.

comp.os.linux.networking
: A Linux rendszerek hálózatkezelésének témakörével foglalkozik.

comp.os.linux.development
: A Linux rendszermagnak és magának a rendszernek a fejlesztéséről szól.

comp.os.linux.misc
: Átfogó hírcsoport, ahol az előbbiekben említett témákon túlmutató kérdésekkel foglalkoznak.

Létezik néhány nem angol nyelvű Linux-hírcsoport is, például a francia *fr.comp.os.linux* vagy a német *de.comp.os.linux*.

Linux levelezőlisták

Nagy számban léteznek specializált Linux levelezőlisták, ahol mindig találhatunk valakit, aki szívesen válaszol a kérdésünkre.

A listák közül a legismertebb a Linux Kernel Mailing List. A mindig forgalmas, sokak által látogatott listára naponta roppant mennyiségű információ érkezik. További részleteket a *http://www.tux.org/lkml* címen olvashatunk.

Linux felhasználói csoportok

A Linux felhasználói csoportok világszerte közvetlen támogatást kínálnak a felhasználóknak, telepítési napok, beszélgetések és szemináriumok, demonstrációs éjszakák és

egyéb társasági események a leglátogatottabb fórumaik. Mindezek kiváló alkalmakat teremtenek, hogy a környékünkön lakó Linux-felhasználókkal megismerkedjünk. A Linux felhasználói csoportokról több listát is közzétesznek. Az egyik legátfogóbb lista a Linux Users Groups Worldwide *(http://lugww.counter.li.org/index.cms)*.

A Linux beszerzése

A Linux nem egyetlen formában létezik: a szoftvernek több különböző terjesztését is beszerezhetjük; ilyen például a Debian, a Fedora, a Red Hat, a SUSE, a Gentoo vagy a Slackware. A terjesztések mindent tartalmaznak, amire egy teljes Linux-rendszer futtatásához szükség lehet: a rendszermagot, az alapvető segédprogramokat, a könyvtárakat, a támogatási állományokat és az alkalmazásokat.

A Linux-terjesztéseket sokféle on-line forrásból beszerezhetjük, például az internetről. Minden nagyobb disztribúció önálló FTP oldallal és webhellyel rendelkezik. Lássunk most egypárat!

Debian
http://www.debian.org/

Gentoo
http://www.gentoo.org/

Red Hat
http://www.redhat.com/

Fedora
http://fedora.redhat.com/

Slackware
http://www.slackware.com/

SUSE
http://www.suse.com/

A népszerű, általános WWW archívumok közül több archívum a különféle Linux-terjesztéseket is tükrözi; a legismertebb talán a *http://www.linuxiso.org*.

Minden nagyobb terjesztést közvetlenül letölthetünk az internetről, de egyre több szoftverforgalmazó kínálja a Linuxot CD-ROM lemezen. Ha nem találunk ilyet a helyi számítógép-kereskedésben, érdemes figyelmeztetni őket, hogy tartsanak raktáron! A legnépszerűbb disztribúciók megvásárolhatók CD-ROM-on. Egyes forgalmazók olyan csomagokat forgalmaznak, amelyek több CD-ROM lemezt tartalmaznak, rajtuk különböző Linux-disztribúciókkal. Ez remek lehetőséget nyújt, hogy több terjesztést is kipróbáljunk, mielőtt kiválasztjuk jövendő kedvencünket.

A fájlrendszer-szabványok

A múltban a Linux-terjesztések és a rajtuk futó szoftvercsomagok egyik problémáját az jelentette, hogy nem létezett egységes fájlrendszer-kialakítás. Ezért a különböző csomagok nem voltak egymással kompatibilisek, a felhasználók és a rendszergazdák pedig a különféle állományok és programok keresgélésével tölthették az idejüket.

A helyzet javítására 1993 augusztusában megalakult a Linux File System Standard Group (FSSTND) csoport. A hat hónapig tartó egyeztetés után a csoport felvázolt egy koherens fájlrendszer-szerkezetet, és meghatározta az alapvető programok és konfigurációs állományok helyét.

A nagy Linux-terjesztések és -csomagok elvileg ezt a szabványt követik. Sajnos azonban, noha a legtöbb terjesztésen észrevehetők az FSSTND-előírások megvalósítására tett kísérletek, valójában csak nagyon kevés terjesztés alkalmazta a szabványt teljes egészében. A könyvben feltételezzük, hogy a tárgyalt állományok a szabványban előírt helyen találhatók, és csak akkor említjük az alternatív lelőhelyeket, ha azoknak régi hagyománya van és ütközik az előírásokkal.

A Linux FSSTND tovább fejlődött ugyan, de 1997-ben a Linux File Hierarchy Standard (FHS) vette át a helyét. Az FHS a többszörös architektúrák kihívásainak is megfelel, az FSSTND viszont nem. Az FHS beszerezhető a *http://www.freestandards.org* címről.

A Linux Standard Base projekt

A számos különböző Linux-terjesztés egészséges választási lehetőséget biztosít a Linux-felhasználóknak, de fejfájást okoz a szoftverfejlesztőknek - különösen a kereskedelmi forgalomba kerülő szoftverek esetében.

Minden terjesztés tartalmaz bizonyos alapvető könyvtárakat, konfigurációs eszközöket, rendszeralkalmazásokat és konfigurációs állományokat. Sajnálatos módon a verziószámuk, a nevük és az elhelyezkedésük eltérő lehet, így nagyon nehéz megmondani, hogy adott terjesztésben melyek találhatók meg közülük. A fejlesztőknek komoly kihívás, hogy az összes alapvető Linux-terjesztésben egyformán megbízhatóan üzemelő bináris alkalmazások legyenek.

A helyzet javítására hozták létre a Linux Standard Base projektet, amelynek célja egy szabványos alapdisztribúció definiálása a szabványt elfogadó terjesztések számára. Így ha egy fejlesztő a szabványos alaprendszerre készít alkalmazást, az működőképes és hordozható lesz a szabványnak megfelelő összes Linux-disztribúcióban.

A Linux Standard Base projekt aktuális híreiről annak honlapján, a *http://www.linuxbase.org/* címen tájékozódhatunk.

Ha fontosnak tartjuk a hordozhatóságot, akkor győződjünk meg arról - különösen a kereskedelmi forgalmazóktól származó szoftverek esetén -, hogy Linux-terjesztésünk részt vesz a szabványosítási törekvésben.

A könyvről

Amikor Olaf Kirch 1992-ben a Linux Documentation Projecthez csatlakozott, két rövid fejezetet írt az UUCP protokollról és a smail alkalmazásról, melyeket a System Administrator's Guide (a rendszeradminisztrátorok kézikönyve) kiadványba szánt. A TCP/IP hálózatkezelés fejlesztése akkoriban indult el, és amikor a „rövid fejezetek" duzzadni kezdtek, a szerző elgondolkozott azon, mennyire jó volna, ha létezne egy Networking Guide, vagyis egy hálózati adminisztrátori kézikönyv. Nagyszerű ötlet! – biztatták a többiek. – Kezdj hozzá! Így írta meg a Networking Guide első változatát, amely 1993 szeptemberében látott napvilágot.

Olaf tovább dolgozott a hálózatkezelési kézikönyvön, és végül elkészült a kiadvány jóval részletesebb változata. Az eredeti sendmail levelezőről szóló fejezetet Vince Skahan írta, amit ebben a kiadásban teljesen új anyaggal helyettesítettek, mivel a sendmail konfiguráció új interfészt kapott.

2000 márciusában Terry Dawson hozta naprakész állapotba Olaf eredeti munkáját, hozzáfűzve néhány új fejezetet és korszerűsítve a könyvet az új millennium elvárásainak megfelelően.

A kötet, amelyet az Olvasó most a kezében tart, a korábbi kiadás alapos átdolgozása és frissítése; a munkát az O'Reilly Media kezdeményezésére Tony Bautts végezte el. Tony már annyira régóta lelkes Linux-felhasználó és információbiztonsági tanácsadó, hogy ezt nem is szívesen vallja be. A számítógépek biztonságával foglalkozó több könyv társszerzője, és a témáról előszeretettel tart előadásokat is. Lelkes támogatója a Linux felhasználásának kereskedelmi környezetben, és szokása, hogy megpróbál mindenkit meggyőzni a Gentoo Linux kivételes előnyeiről. A jelen kiadáshoz több fejezetet is írt a Linux olyan hálózatkezelési jellemzőiről, melyek a második kiadás óta keletkeztek, illetve számos változtatást végzett, hogy a kötetet korszerűsítse.

Az iptables-fejezeteket (7., 8. és 9. fejezet) Gregor Purdy frissítette a jelen kiadáshoz.

A könyv felépítése többé-kevésbé azokat a feladatokat követi, melyeket a rendszerünkön el kell végeznünk, amikor felkészítjük a hálózatkezelésre. Mindenekelőtt a hálózatok alapvető kérdéseit vizsgálja meg, különös tekintettel a TCP/IP alapú hálózatokra. Ezután a TCP/IP eszközszintű beállításától módszeresen eljut a tűzfal, a nyomkövetés (accounting) és az álcázás (masquarade) konfigurálásáig, majd az olyan elterjedt alkalmazások, mint az SSH, az Apache és a Samba beállításáig. Az elektronikus levelezéssel foglalkozó rész bevezetője a levelek átvitelének és útválasztásának meghitt részleteiben kalauzol, valamint bemutatja a milliárdnyi címzési sémát, amelyekkel munkánk során találkozhatunk. Szintén itt olvashatunk a legelterjedtebb levélkézbesítő program, a sendmail beállításáról és karbantartásáról, valamint az IMAP-ról, amellyel leveleket kézbesíthetünk önálló levelező felhasználókhoz.

A korszerű hálózati adminisztráció infrastruktúrájának bemutatását az LDAP-ről és a vezeték nélküli hálózatokról szóló rész zárja le.

Természetesen egyetlen könyv sem adhat választ az összes felmerülő kérdésre. Ezért ha a kötet utasításait követve valami mégsem működik, legyünk türelmesek. Egyrészt előfordulhat, hogy mi hibáztunk (lásd a *Kérdések és megjegyzések* című részt), másrészt a hálózati szoftver változása is okozhatja a hibákat. Ezért mindenekelőtt nézzünk szét a megadott információforrások között! Valószínűleg nem csak mi tapasztaljuk a hibát, így jó esélyünk van rá, hogy találunk egy hibajavítást vagy legalább egy kiskaput – a kere-

sőmotorok ilyenkor jó szolgálatot tehetnek. Ha lehetséges, szerezzük be a legújabb rendszermagot és hálózatkezelő szoftvert a *http://www.kernel.org* címről. Sok esetben a hibákért a fejlesztés különböző stádiumaiban lévő különböző szoftverek a felelősek, melyek nem működnek együtt megfelelően. Végül is a Linux folyamatosan „fejlesztés alatt" áll.

A hivatalos nyomtatott változat

1993 őszén Andy Oram, aki szinte a kezdetektől jelen volt a Linux Documentation Project levelező listáján, megkérdezte Olaftól, mit szólna hozzá, ha az O'Reilly & Associates megjelentetné a könyvet. Olafot nagyon izgatta a lehetőség, de soha nem gondolta volna, hogy a kötet ennyire sikeres lehet. Ő és Andy végül megállapodott, hogy az O'Reilly kiadja a Networking Guide bővített, hivatalos nyomtatott változatát, de Olaf megtarthatja az eredeti szerzői jogokat, hogy a könyv forrása továbbra is szabadon terjeszthető maradjon. Ez azt jelenti, hogy választhatunk: a legközelebbi LDP tükröoldalról beszerezzük a dokumentum valamely ingyenes formátumát és kinyomtatjuk, vagy megvásároljuk az O'Reilly hivatalos nyomtatott kiadványát.

De miért fizetnénk olyasmiért, amit ingyen is megkaphatunk? Elment Tim O'Reilly józan esze, hogy kiad valamit, amit bárki kinyomtathat és akár értékesíthet is?* Van-e különbség az egyes változatok között?

A válasz ilyesmi: „attól függ", „nem, egyáltalán nincs", vagy „igen is és nem is". Az O'Reilly Media valóban kockázatot vállalt a Networking Guide megjelentetésével, de úgy látszik, kifizetődött nekik (hiszen azóta még kétszer kérték tőlünk a könyvet). Úgy hisszük, kiadásunk nagyszerű példa, hogyan jöhet létre mindkét fél számára kifizetődő együttműködés a szabad szoftver világa és a cégek között. Nézetünk szerint az O'Reilly hatalmas szolgálatot tett a Linux-közösségnek (túl azon, hogy már a közeli könyvesboltban is beszerezhetjük a kötetet): segítettek bebizonyítani, hogy a Linuxot komolyan kell venni, mert életképes és hasznos alternatívája a kereskedelmi forgalomban kapható más operációs rendszereknek. Csak sajnálhatjuk azt a műszaki könyvesboltot, ahol nincs legalább egy polc telerakva az O'Reilly linuxos köteteivel.

És még ahhoz, miért jelentetik meg az O'Reilly-nál a könyvet? Mert úgy gondolják, nekik való könyv. Ilyet szeretnének kapni, ha megbíznának egy szerzőt, hogy írjon könyvet a Linuxról. A tempó, a kidolgozottság és a stílus jól illeszkedik az O'Reilly egyéb kiadványaihoz.

Az LDP licenc lényege, hogy senkit ne lehessen kirekeszteni. Mások is kiadhatják a könyvet, és senki nem hibáztatja azt, aki más kiadást választ. De ha még nem láttuk volna az O'Reilly-változatot, érdemes ellátogatnunk egy könyvesboltba vagy kölcsönkérnünk egy barátunk példányát. Hisszük, hogy tetszeni fog, és mindenki szeretne majd egyet magának is.

Tehát mi a különbség a nyomtatott és az on-line változat között? Andy Oram nagyon sok munkát fektetett abba, hogy összefüggéstelen irományunkból nyomtatásra érdemes anyagot állítson össze. (Az LDP több más könyvét is átvizsgálta, szakmai ismereteivel támogatva a Linux-közösséget.)

* Fontos tudnunk, hogy az on-line változatot szabadon nyomtathatjuk, az O'Reilly kötetét viszont nem fénymásolhatjuk, és a (feltételezett) másolatokat nem értékesíthetjük.

Amióta Andy hozzákezdett a Networking Guide átnézéséhez és az elküldött példányok megszerkesztéséhez, a kötet hatalmas fejlődésen ment keresztül eredeti formájához képest, és minden alkalommal, amikor elküldjük neki az anyagot, és megkapjuk tőle a visszajelzést, úgy látjuk, az anyag még tovább javul. Nem szabad kihasználatlanul hagynunk azt a lehetőséget, hogy egy magas szakmai képzettségű szerkesztő segítségét vehetjük igénybe. Andy munkája több szempontból is legalább olyan fontos volt, mint a szerzőké. Ugyanez elmondható a kötet előállításán dolgozó munkatársakról, akiknek az itt látható megjelenés köszönhető. A szerkesztői változtatások bekerültek az online változatba is, így a tartalomban nincs különbség.

Az O'Reilly változat azonban *mégis* más. A kötése kiváló minőségű, és bár fáradságot nem kímélve magunk is kinyomtathatjuk az ingyenes változatot, annak minősége valószínűleg nem lesz ugyanolyan. Másodsorban, kezdetleges rajzaink helyére az O'Reilly hivatásos művészeinek szépen kivitelezett ábrái kerültek. Az indexkészítők javított tárgymutatót hoztak létre, ami jócskán leegyszerűsíti az éppen kívánt információ megkeresését a könyvben. Amennyiben szándékunkban áll a könyvet az elejétől a végéig elolvasni, érdemes a hivatalos nyomtatott változatot választanunk.

Áttekintés

A *Bevezetés a hálózatkezelésbe* című 1. fejezet bemutatja a Linux történetét és foglalkozik a hálózatkezelés alapjaival – a UUCP, a TCP/IP és más különféle protokollok, a hardver és a biztonság szemszögéből. A következő néhány fejezet a Linux beállítását TCP/IP hálózatkezelésre és az egyes fontosabb alkalmazások futtatását ismerteti.

A TCP/IP hálózatkezelés kérdései című 2. fejezet részletesebben taglalja az IP protokollt, mielőtt komolyabban belemerülnénk az állományszerkesztés kérdéseibe és hasonlókba. Aki tisztában van az IP útválasztás és a címfeloldás működési elvével, átugorhatja a fejezetet.

A soros hardver beállítása című 3. fejezet a soros portok beállítását ismerteti.

A TCP/IP hálózatkezelés beállítása című 4. fejezet segítséget nyújt a TCP/IP hálózatkezelés beállításában a számítógépünkön. Találunk itt ötleteket az önálló és a hálózatra kapcsolt gazdagépek telepítéséhez, valamint megismerünk néhány célszerű eszközt, amelyeket a beállítások tesztelésére és a hibakeresésre használhatunk.

A névszolgáltatás és beállítása című 5. fejezet a gazdagépnév feloldásának kérdését vizsgálja és bemutatja a névkiszolgáló beállításának lépéseit.

A Point-to-Point Protokoll című 6. fejezet témája a PPP, valamint a *pppd*, a PPP daemon.

A TCP/IP tűzfal című 7. fejezetben tovább bővítjük ismereteinket a hálózati biztonság témáiról, és megismerkedünk a Linux iptables nevű TCP/IP tűzfalával. Az IP tűzfalakkal rendkívül pontosan szabályozhatjuk, ki férhet hozzá hálózatunkhoz és számítógépeinkhez.

Az *IP nyomkövetés* című 8. fejezet bemutatja az IP nyomkövetés (accounting) konfigurálását a Linux rendszeren. Az eljárással nyomon követhetjük a forgalom nagyságát, irányát és forrását.

Az IP álcázás és hálózati címfordítás című 9. fejezet a Linux hálózatkezelő szoftverének egyik tevékenységével, az úgynevezett IP álcázással (vagy Network Address Translation, NAT) foglalkozik, amely lehetővé teszi, hogy teljes IP hálózatok egyetlen IP címen

keresztül kapcsolódjanak az internetre, miközben a belső rendszerek a külvilág elől rejtve maradnak.

A *Fontos hálózati jellemzők* című 10. fejezet röviden bemutatja néhány fontos hálózati infrastruktúra és alkalmazás, például az SSH beállítását. Foglalkozik emellett azzal is, hogyan kezeli az inetd szuperdaemon a szolgáltatásokat, és hogyan korlátozhatunk bizonyos, biztonsági szempontból lényeges szolgáltatásokat a megbízható számítógépek csoportjára.

Az elektronikus levelezés adminisztrációs kérdései című 11. fejezet bevezetés az elektronikus levelezés alapvető fogalmaiba; megtudhatjuk például, hogyan épülnek fel a levélcímek, és hogyan juttatja el a levelezéskezelő rendszer a leveleket a címzetthez.

A sendmail című 12. fejezet a Linux rendszereken futó *sendmail* levelezési közvetítő beállítását tárgyalja.

Az IPv6 hálózatok beállítása című 13. fejezet új tájakra vezet bennünket: megismerjük az IPv6 beállítását, és megtanuljuk, hogyan kapcsolódhatunk az IPv6 gerinchálózatra.

Az Apache webkiszolgáló beállítása című 14. fejezet az Apache webkiszolgáló kiépítésének és néhány alapvető szolgáltatás biztosításának lépéseit veszi sorra.

Az *IMAP* című 15. fejezet az IMAP levelezőkiszolgáló beállítását tárgyalja, és megvitatja annak előnyeit a hagyományos POP levelezési megoldással szemben.

A *Samba* című 16. fejezet segít megértenünk, hogyan állíthatjuk be a Linux kiszolgálónkat úgy, hogy tökéletesen beilleszkedjen a Windows hálózatok világába - ami azt illeti, olyan tökéletesen, hogy windowsos felhasználóink nem is veszik észre a különbséget.*

Az *OpenLDAP* című 17. fejezet témája az OpenLDAP protokoll. A fejezet a szolgáltatás beállításával és lehetséges felhasználási területeivel foglalkozik.

A *Vezeték nélküli hálózatok* című 18. fejezetben végül a vezeték nélküli hálózatok beállításának és egy Linux szerveren futó vezeték nélküli hozzáférési pont kiépítésének a lépéseit vesszük sorra.

A könyvben használt jelölések

A könyvben szereplő példák feltételezik, hogy sh kompatibilis héjat (shell) használunk. A bash héj sh kompatibilis és a Linux-disztribúciók szabványos része. Ha történetesen csh héjat futtatunk, el kell végeznünk a megfelelő módosításokat.

A könyvben a következő jelölésekkel találkozunk:

Dőlt

Dőlt betűvel jelezzük az állományok és könyvtárak nevét, a program- és parancsneveket, az e-mail címeket és az útvonalakat, az URL címeket és a hangsúlyozni kívánt új kifejezéseket.

* A kézenfekvő poént az Olvasóra bízzuk. (Ha valaki nem ismerné, a közkeletű szellemes megjegyzés úgy szól, hogy a Linux mindennel képes kommunikálni, csak füstjelekkel nem, de már dolgoznak rajta. A mondás eredetileg Dr. Greg Wettsteintől származik - a fordító.)

Félkövér
A számítógépneveket és a hálózati helyek nevét jelöli, valamint alkalmanként figyelemfelhívásra szolgál.

`Azonos betűszélesség`
A példákban a kódállományok tartalmát vagy egy parancs kimenetét jelöli, valamint a kódban megjelenő környezeti változókat és kulcsszavakat mutatja.

`Azonos betűszélesség, dőlt betű`
Változó opciókat, kulcsszavakat vagy olyan megváltoztatható szöveget jelez, amelynek helyére a felhasználónak kell a tulajdonképpeni értéket beírnia.

`Azonos betűszélesség, félkövér betű`
A példákban a parancsokat és más, szó szerint begépelendő szöveget jelöli.

Az ikon ötletet, javaslatot vagy általános megjegyzést jelöl.

Az ikon figyelmeztet vagy óvatosságra int: olyan hibát ejthetünk, amely megrongálhatja a rendszert, vagy csak nehezen korrigálható.

A Safari Enabled ikon

Amikor kedvenc műszaki könyvünkön felfedezzük a Safari® Enabled jelzést, tudhatjuk, hogy a könyv on-line elérhető az O'Reilly Network Safari Bookshelf könyvtárban.

A Safari az e-könyveknél jobb megoldást kínál. Valójában egy virtuális könyvtárról van szó, ahol több ezer magas szintű műszaki könyvben kereshetünk, példákat másolhatunk ki, fejezeteket tölthetünk le és gyorsan megtalálhatjuk kérdéseinkre a legpontosabb, legfrissebb információn alapuló válaszokat. Ingyenesen kipróbálhatjuk a *http://safari.oreilly.com* címen.

Kérdések és megjegyzések

A könyvben közölt adatokat a legjobb képességünk szerint ellenőriztük és felülvizsgáltuk, mégis előfordulhat, hogy bizonyos jellemzők megváltoztak (vagy akár mi is tévedhettünk!). Kérjük, tudassák velünk, ha hibát vagy pontatlanságot tapasztalnak, és osszák meg velünk javaslataikat a jövőbeni kiadásokkal kapcsolatban. A következő címre írhatnak:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (az Egyesült Államokban és Kanadában)
(707) 829-0515 (nemzetközi/helyi)
(707) 829-0104 (fax)

Elektronikus üzenetet is küldhetnek. Ha fel szeretnének kerülni a levelezőlistára vagy katalógust kérnek, a következő e-mail címre küldhetnek üzenetet:

info@oreilly.com

A műszaki kérdéseket és megjegyzéseket a következő címre várjuk:

bookquestions@oreilly.com

Külön webhely áll rendelkezésre a könyvhöz, ahol felsoroljuk a hibajavításokat, a példákat és a jövőbeni kiadások terveit. Az oldal címe:

http://www.oreilly.com/catalog/linag3

További információt a könyvről és egyéb kiadványokról az O'Reilly webhelyén találhatnak:

http://www.oreilly.com

A magyar kiadó webcíme és e-mail-címe:

http://www.kossuth.hu
it@kossuth.hu

Köszönetnyilvánítás

A Linux hálózati adminisztrátorok kézikönyvének jelenlegi kiadása sokat köszönhet Olaf, Vince és Terry kiváló munkájának. Amíg az ember maga nem ír hasonló könyvet, el sem tudja képzelni, mennyi munkát kell a kutatásba és az írásba fektetni. A könyv korszerűsítése nagy kihívást jelentett, de élvezetes volt, különösen mert nagyszerű alapra építhettünk.

A kötet sokat köszönhet azoknak is, akik idejüket nem kímélve elolvasták és kijavították hibáinkat. Phil Hughes, John Macdonald és Kenneth Geisshirt roppant hasznos (és egészében véve igen konzisztens) visszajelzést adott a könyv harmadik kiadásának tartalmáról. Andres Sepúlveda, Wolfgang Michaelis és Michael K. Johnson értékes segítséget nyújtottak a második kiadásnál. És végül, a könyv nem születhetett volna meg Holger Grothe támogatása nélkül, aki Olafnak internetes hozzáférést biztosított az eredeti változat kidolgozásakor.

Terry feleségének, Maggie-nek szeretne köszönetet mondani, aki első gyermekük, Jack születésének idején is türelmesen támogatta a részvételét a munkában. Szintén szeretne köszönetet mondani a Linux-közösség számos tagjának, akik gyengéden vagy erőszakosan, de rávették, hogy részt vegyen a tevékenységünkben és ahhoz aktívan hozzájáruljon. „Segítek neked, ha cserébe te is segítesz másoknak!"

Tony a két Linux-gurunak, Dan Ginsbergnek és Nicolas Lidzborskinak fejezi ki köszönetét támogatásukért, és mert műszaki ismereteiket latba vetve átvizsgálták az új fejezeteket. Köszönetet mond továbbá Katherine-nek a fejezetekhez adott ötleteiért, bár ő valójában csak az e-mailjét szerette volna elolvasni. Köszönöm Mick Bauernek, hogy részt vett a munkában és mindvégig támogatott. Végül sok köszönet a megszámlálhatatlan Linux-felhasználónak, akik segítőkészen dokumentálták a kockázatokat, amelyekkel egy-egy feladat megoldásakor találkoztak, hogy azokat ne is említsük, akik a levelezőlistára küldött kérdésekre nap mint nap válaszoltak. Ilyen közösségi támogatás nélkül a Linux sehol sem volna.

1

Bevezetés a hálózatkezelésbe

Történeti áttekintés

A hálózatkezelés gondolata valószínűleg egyidős a távközléssel. Gondoljunk csak kőkori barlanglakó őseinkre, akik mondanivalójukat dobok segítségével osztották meg egymással. Tegyük fel például, hogy A elődünk szeretné kődobálásra hívni B elődünket, azonban B túl messze lakik ahhoz, hogy hallja A dobverését. Ekkor A ősünk a következő lehetőségek közül választhat: 1. személyesen keresi fel B-t, 2. készít egy nagyobb dobot vagy 3. megkéri a félúton lakó C-t, hogy továbbítsa az üzenetet. Ez utóbbit nevezzük *hálózatkezelésnek.*

Természetesen elődeink egyszerű kedvtelései és eszközei már csak a múlt emlékei. Jelenleg ha barátainkat meg szeretnénk hívni a szombat esti meccsre, akkor olyan számítógépeken keresztül tesszük, amelyek kábelek, fényvezető üvegszálak, a mikrohullámú tartományba eső rádióhullámok és egyéb kommunikációs csatornák végtelen nyalábjain keresztül tartanak egymással kapcsolatot.* A következőkben megnézzük, milyen eszközökkel és eljárásokkal valósítható meg mindez – bár a vezetékekről nem lesz szó, és a meccsektől is kénytelenek vagyunk eltekinteni.

Meghatározás szerint a hálózat olyan gazdagépek (host) csoportja, amelyek képesek egymással kommunikálni – ehhez gyakran dedikált, vagyis különleges célra, jelen esetben az üzenetek továbbítására fenntartott gazdagépek szolgáltatásait veszik igénybe. A gazdagépek általában számítógépek, de nem feltétlenül; egy X terminál vagy egy intelligens nyomtató is lehet gazdagép. A gazdagépek alkotta csoportot *hálózati helynek* (site) is nevezzük.

A kommunikáció lehetetlen volna valamiféle nyelv vagy kód nélkül. A számítógépes hálózatokban e nyelveket együttesen *protokolloknak* hívjuk. Ne papírra vetett és pontokba szedett szabályzatra gondoljunk azonban – valójában olyan kifinomult viselkedési formáról van szó, amilyet az államfők találkozásakor figyelhetünk meg. Hasonló módon a számítógépes hálózatokban alkalmazott protokollok is csupán rendkívül szigorú szabályok, melyek meghatározzák, hogy két vagy több gazdagép hogyan cserélhet információt egymással.

* Ennek ellenére ilyenkor néha még felszínre bukkan Európában az ősi szellem (lásd fent).

TCP/IP hálózatok

A modern hálózatkezelő alkalmazások kifinomult eljárásokat igényelnek az adatok továbbításához az egyik számítógépről a másikra. Ha például olyan linuxos számítógépet tartunk fenn, amelyre számos felhasználó csatlakozik, és mindegyikük egyidejűleg szeretne a hálózat különböző távoli gépeire kapcsolódni, szükségünk van egy eljárásra, mely lehetővé teszi a hálózati kapcsolatunk megosztását anélkül, hogy a felhasználók zavarnák egymást. Erre a célra a korszerű hálózatkezelő protokollok jelentős része *csomagkapcsolást* alkalmaz. A csomag tulajdonképpen egy kis adattöredék, amely a hálózaton keresztül utazva az egyik gépről a másikra vándorol. Kapcsolásról akkor beszélünk, amikor a datagram a hálózat egyes kapcsolatain (link) halad keresztül. A csomagkapcsolt hálózat számos felhasználó között oszt meg egy-egy kapcsolatot úgy, hogy a csomagokat a kapcsolaton keresztül váltakozva küldi az egyik felhasználótól a másikig.

Az először a Unix rendszerek, majd később sokféle nem Unix rendszer által is elfogadott és alkalmazott eljárás neve TCP/IP. A TCP/IP hálózatokkal ismerkedve találkozhatunk a *datagram* kifejezéssel, melynek megvan a saját jelentése, noha gyakran a csomag szinonimájaként alkalmazzák. A következő részben a TCP/IP protokollok vezérelveivel, alapvető elgondolásaival ismerkedünk meg.

Bevezetés a TCP/IP hálózatokba

A TCP/IP protokoll eredete 1969-re, az Egyesült Államok Védelmi Minisztériumának kutatásszervezési ügynöksége (Advanced Research Projects Agency, ARPA) által pénzelt kutatáshoz nyúlik vissza. Az ARPANET kísérleti hálózat volt, melyet sikerére való tekintettel 1975-ben rendszeresítettek.

Az új TCP/IP protokollt 1983-ban szabványosították, és a hálózat összes gépe ezt használta. Amikor az ARPANET-ből végül az internet kifejlődött (miközben maga az ARPANET 1990-ben megszűnt), a TCP/IP az interneten is túlnőtt. Ma már számos cég épít ki vállalati TCP/IP hálózatokat, az internet pedig mint technológia a felhasználók hatalmas tömege számára vált hozzáférhetővé. Nincs olyan napilap vagy magazin, melyben ne találnák hivatkozást a ma már mindenki számára elérhető internetre.

Azért, hogy a TCP/IP protokoll tárgyalását a következőkben némileg kézzelfoghatóbbá tegyük, vegyük a freedoniai Groucho Marx Egyetem (GME) példáját. A legtöbb tanszék saját helyi, lokális hálózatot (LAN) tart fenn, bizonyos tanszékek egy hálózaton osztoznak, mások többet is futtatnak. A hálózatok mindegyike kapcsolatban áll egymással, valamint egy széles sávú kapcsolaton keresztül az internettel is.

Tételezzük fel, hogy az **erdos** névre hallgató Linux számítógépünk a Matematika tanszék Unix gazdagépekből álló lokális hálózatára kapcsolódik. A Fizika tanszék egy adott gazdagépének eléréséhez, melynek neve legyen **quark**, a következő parancsot gépeljük be:

```
$ ssh quark.school.edu
Enter password:
Last login: Wed Dec 3 18:21:25 2003 from 10.10.0.1
quark$
```

Miután a parancskérő jelnél megadtuk a jelszavunkat, a **quark** héjába (shell)* jutunk, melybe úgy írhatunk, mintha közvetlenül a rendszer konzolja előtt ülnénk. A héjból való kilépést követően ismét saját gépünk parancssorához jutunk vissza. Az előbbi példában a TCP/IP protokollra épülő gyors és interaktív alkalmazások egyikét, a biztonságos héjat (secure shell) használtuk.

Miután bejelentkeztünk a **quark** gépre, szükségünk lehet a grafikus felhasználói felületet biztosító alkalmazások, például egy szövegszerkesztő, egy rajzprogram vagy akár egy internetes böngésző futtatására. Az X Windows System olyan grafikus felhasználói környezet, amelyet tökéletesen felkészítettek a hálózati üzemre, és emellett számos különböző számítógépes rendszerhez felhasználható. Azért, hogy alkalmazásunkat rávegyük az ablakok megjelenítésére a számítógépünk képernyőjén, meg kell győződnünk arról, hogy az SSH kiszolgálónk és ügyfelünk alkalmas az X továbbításra. Ehhez ellenőrizzük, hogy a rendszerünk *sshd_config* állománya tartalmaz-e egy ehhez hasonló sort:

```
X11Forwarding yes
```

Ha most elindítjuk a programot, az továbbítja az X Window System alkalmazásokat, így azok ablakai saját X kiszolgálónkon, és nem a **quark** gépen jelennek meg. Természetesen ehhez az szükséges, hogy az **erdos** gépen X11 fusson. Számunkra jelen esetben az a fontos, hogy a TCP/IP lehetővé teszi a **quark** és **erdos** gépeknek az X11 csomagok küldését és fogadását, azt az illúziót keltve, hogy egyetlen rendszert használunk. A hálózat számunkra szinte teljesen átjárható.

Mindez persze csak példa arra, mi mindenre vagyunk képesek a TCP/IP hálózatokkal. A lehetőségek csaknem korlátlanok, és a könyv hátralévő részében többel is megismerkedünk.

Most nézzük meg közelebbről, hogyan működik a TCP/IP. A tanultak segítségével megérthetjük, miért és hogyan állítsuk be számítógépünket. Az ismerkedést a hardver vizsgálatával kezdjük.

Az Ethernet hálózatok

A lokális hálózatok (LAN) legelterjedtebb megvalósítása az *Ethernet hálózat.* Legegyszerűbb formájában ez a kiszolgálókat különféle csatlakozókon, dugaszokon vagy adóvevő készüléken keresztül összekötő egyetlen kábelt jelent. Az egyszerű Ethernet hálózatok telepítése viszonylag olcsó, és ha ehhez hozzávesszük a 10, 100, 1000 vagy ma már akár 10 000 Megabit/másodperc hasznos átviteli sebességet, érthetővé válik a népszerűségük.

Az Ethernet kábelek több típusa létezik, így lehetnek *vékony, vastag* vagy *csavart érpárúak.* A régebbi Ethernet típusok, például a ma már csak elvétve látható vékony és vastag Ethernetek különböző átmérőjű, és a számítógépekhez eltérő módon csatlakozó koaxiális kábelt használnak. A vékony Ethernet hálózaton T alakú „BNC" csatlakozót találunk, amelyet a kábel végére illesztve a számítógép hátán lévő csatlakozóba csavarhatunk. A vastag Ethernet kábelbe egy kis lyukat fúrunk, és „vámpír" leágazóval illesztjük

* A héj a Unix operációs rendszerek parancssoros kezelőfelülete, hasonló a Microsoft Windows környezet DOS parancssorához, de annál jóval hatékonyabb.

rá az adóvevőt, melyhez egy vagy több gépet csatlakoztathatunk. A vékony és vastag Ethernet kábelek hossza legfeljebb 200, illetve 500 méter lehet, és 10-base2, illetve 10-base5 néven is ismertek. A „base" itt a „baseband modulation" kifejezésre utal, ami egyszerűen azt jelenti, hogy az adatok bármiféle modem beiktatása nélkül kerülnek rá a kábelre. Az első szám a Megabit/másodpercben mért sebességet, az utolsó szám pedig a kábel száz méterben kifejezett maximális hosszát jelzi. A csavart érpárú Ethernet kábel két rézvezetéket tartalmaz, és rendszerint egy további hardver, az úgynevezett *jelelosztó* (hub) jelenlétét igényli. A csavart érpár 10-baseT néven is ismert, ahol a „T" jelentése csavart (twisted). A 100 Mbps változat neve 100-baseT, az 1000 Mbps sebességű változat pedig érthető módon az 1000-baseT vagy gigabit névre hallgat.

A vékony Ethernet rendszerre csak úgy tudunk telepíteni egy gazdagépet, ha a hálózati szolgáltatást legalább néhány percre felfüggesztjük, mert a csatlakozó elhelyezéséhez el kell vágnunk a kábelt. Noha a vastag Ethernetre csak kevéssel bonyolultabb eljárással telepíthetünk új gazdagépet, a hálózatot általában nem szükséges leállítanunk. A csavart érpárú Ethernet még ennél is egyszerűbb, mert rajta úgynevezett jelelosztót (hub) vagy hálózati kapcsolót (switch) találunk, mindegyik kapcsolódási pontként szolgál. A jelelosztó vagy kapcsoló segítségével úgy távolíthatunk el, illetve adhatunk gazdagépeket a hálózathoz, hogy azzal a felhasználókat egyáltalán nem zavarjuk.

A vastag és vékony Ethernet rendszerekkel ma már ritkán találkozunk, mert a legtöbbet felváltotta a csavart érpárú kiépítés. Az olcsó hálózati kártyáknak és kábeleknek köszönhetően ez utóbbi gyakorlatilag szabvánnyá vált – nem is szólva arról, hogy a korszerű laptop készülékekbe már csak elvétve építenek BNC csatlakozót.

A vezeték nélküli lokális hálózatok szintén nagyon népszerűek. Ezek a hálózatok a 802.11a/b/g szabványokat követik, és az Ethernetet rádióátvitellel biztosítják. A vezetékes „testvéréhez" hasonló funkcionalitást kínáló vezeték nélküli Ethernetnél számos biztonsági kérdés merült fel, mégpedig a titkosítással kapcsolatban. A protokoll fejlődésével és a különböző titkosítási eljárásokkal azonban a legsúlyosabb biztonsági hiányosságok könnyen kiküszöbölhetővé váltak. A 18. fejezetben részletesen foglalkozunk a vezeték nélküli hálózatkezeléssel linuxos környezetben.

Az Ethernet a buszos rendszerhez hasonlóan működik, ahol egy gazdagép akár 1500 bájt méretű csomagokat (vagy *kereteket)* küldhet az azonos Ethernet hálózaton lévő más gazdagépekre. A gazdagépek címzése az Ethernet hálózati interfészkártya (network interface card, NIC) firmware-ébe előre programozott hatbájtos címekkel történik. A címeket rendszerint kettősponttal elválasztott két számjegyű hexadecimális számok sorozataként írjuk fel, például **aa:bb:cc:dd:ee:ff**.

Egy adott állomás által elküldött keretet minden csatlakoztatott állomás lát, de csak a megcélzott gazdagép fogadja és dolgozza fel. Amikor egyidejűleg két állomás próbál adatot küldeni, azok *összeütköznek.* Az Ethernet hálózaton bekövetkező ütközéseket az interfészkártyák elektronikája rendkívül gyorsan felismeri és megoldja: mindkét állomás megszakítja a küldést, majd véletlenszerű ideig tartó szünet beiktatása után ismét megkísérli az átvitelt. Gyakran hallhatjuk, hogy az ütközések komoly gondot jelentenek az Ethernet hálózaton, és hogy miattuk az Ethernetek kihasználtsága a teljes sávszélességnek nagyjából csak a 30 százaléka. Az Ethernet hálózaton az ütközés teljesen *normális* jelenség. Nagy forgalmú Ethernet hálózaton számíthatunk rá, hogy az ütközések aránya a 30 százalékot is megközelíti. Az Ethernet hálózatok korlátait

reálisan megítélve kijelenthetjük, hogy hozzávetőleg 60 százalék alatt nincs okunk az aggodalomra.*

Más típusú hardverek

Nagyobb hálózatokon vagy egyedi vállalati környezetben az Ethernet rendszerint kiegészül más típusú felszerelésekkel. Számos egyéb adatkommunikációs protokoll is létezik és van használatban. A Linux a következőkben felsorolt protokollok mindegyikét támogatja, de a hely korlátozott volta miatt csupán rövid leírást közlünk róluk. Sok protokollhoz találhatunk részletes ismertetést tartalmazó HOWTO leírásokat, ezért ha olyan protokollt szeretnénk megismerni, mellyel a könyvben nem foglalkozunk, érdemes olvasgatni a HOWTO-kat.

Régi és eltűnőben lévő technológia az IBM Token Ring hálózata. A Token Ring egyes lokális hálózatokban az Ethernet alternatíváját jelentheti; sebessége az Etherneténél alacsonyabb (4 Mbps vagy 16 Mbps). Linux rendszeren a Token Ring hálózatkezelés beállítása szinte teljesen megegyezik az Ethernettel, ezért nem foglalkozunk vele bővebben.

A távközlési vállalatok által fenntartott számos országos méretű hálózat támogatja a csomagkapcsolt protokollokat. Korábban az egyik legnépszerűbb protokoll az *X.25* jelzésű volt. A benne leírt hálózatkezelési eljárások meghatározzák az adatterminálok, például a gazdagépek és az adatok továbbítására szolgáló berendezések (X.25 kapcsoló) közötti kommunikáció módját. Az X.25 egyidejű (szinkron) adatkapcsolatot, ezért speciális szinkron soros portos hardvert igényel. Normál soros porton is használhatjuk a Packet Assembler Disassembler (PAD) segítségével. A PAD egy önálló hardver, amely aszinkron soros portokat és egy szinkron soros portot biztosít. A hardver úgy kezeli az X.25 protokollt, hogy az egyszerű terminálok is képesek X.25 kapcsolatok kezdeményezésére és fogadására. Az X.25 gyakran más, például TCP/IP hálózati protokollok csomagszállítását szolgálja. Mivel az IP datagramok nem képezhetők le egyszerűen az X.25 protokollra (vagy fordítva), ezért ezeket X.25 csomagokba foglalva küldjük át a hálózaton. Az X.25 protokollok implementációja Linux rendszeren is hozzáférhető, ezzel azonban nem foglalkozunk részletesen.

A távközlési vállalatok széles körben alkalmazzák a *Frame Relay* protokollt. A Frame Relay sok műszaki tulajdonsága megegyezik az X.25 protokollal, viselkedésében azonban az IP protokollra hasonlít. Mint az X.25, a Frame Relay is különleges szinkron soros hardvert igényel. A hasonlóságoknak köszönhetően számos kártya mindkét protokollt támogatja. Olyan változata is létezik, amely nem igényel különleges belső hardvert, hanem egy külső eszközt; ez a Frame Relay Access Device (FRAD). Az eszköz feladata, hogy az Ethernet csomagokat Frame Relay csomagokba foglalja, így átküldhetők a hálózaton. A Frame Relay ideális a TCP/IP átvitelére a különböző hálózati helyek között. A Linux biztosít olyan meghajtókat, melyek támogatják a belső Frame Relay eszközök egyes típusait.

Ha nagyobb sebességű hálózatot igénylünk, amely a hagyományos adataink mellett sok más különböző adattípus, például digitalizált hang és videó átvitelére is képes, az

* A *http://www.faqs.org/faqs/LANs/ethernet-faq/* oldalon olvasható Ethernet FAQ is foglalkozik a kérdéssel, Charles Spurgeon Ethernet webhelyén pedig részletes történeti áttekintést és műszaki információs kincsesbányát találhatunk a *http://www.ethermanage.com/ethernet/ethernet.htm/* címen.

Asynchronous Transfer Mode (ATM) jelenthet megoldást. Az ATM olyan új hálózati technológia, melyet kimondottan arra terveztek, hogy biztosítsa a kezelhető, nagy sebességű, rövid várakozási idejű adattovábbítást és a szolgáltatás minőségének (Quality of Service, QoS) szabályozását. Számos távközlési vállalat alkalmaz ATM hálózati infrastruktúrát, mert így lehetőség nyílik a különböző hálózati szolgáltatások egyesítésére egyetlen platformon, ami a működtetési és fenntartási költségekben megtakarítással kecsegtet. Az ATM-et gyakran a TCP/IP hordozására használják. A Linux rendszerek ATM támogatásáról a *Networking HOWTO* nyújt bővebb felvilágosítást.

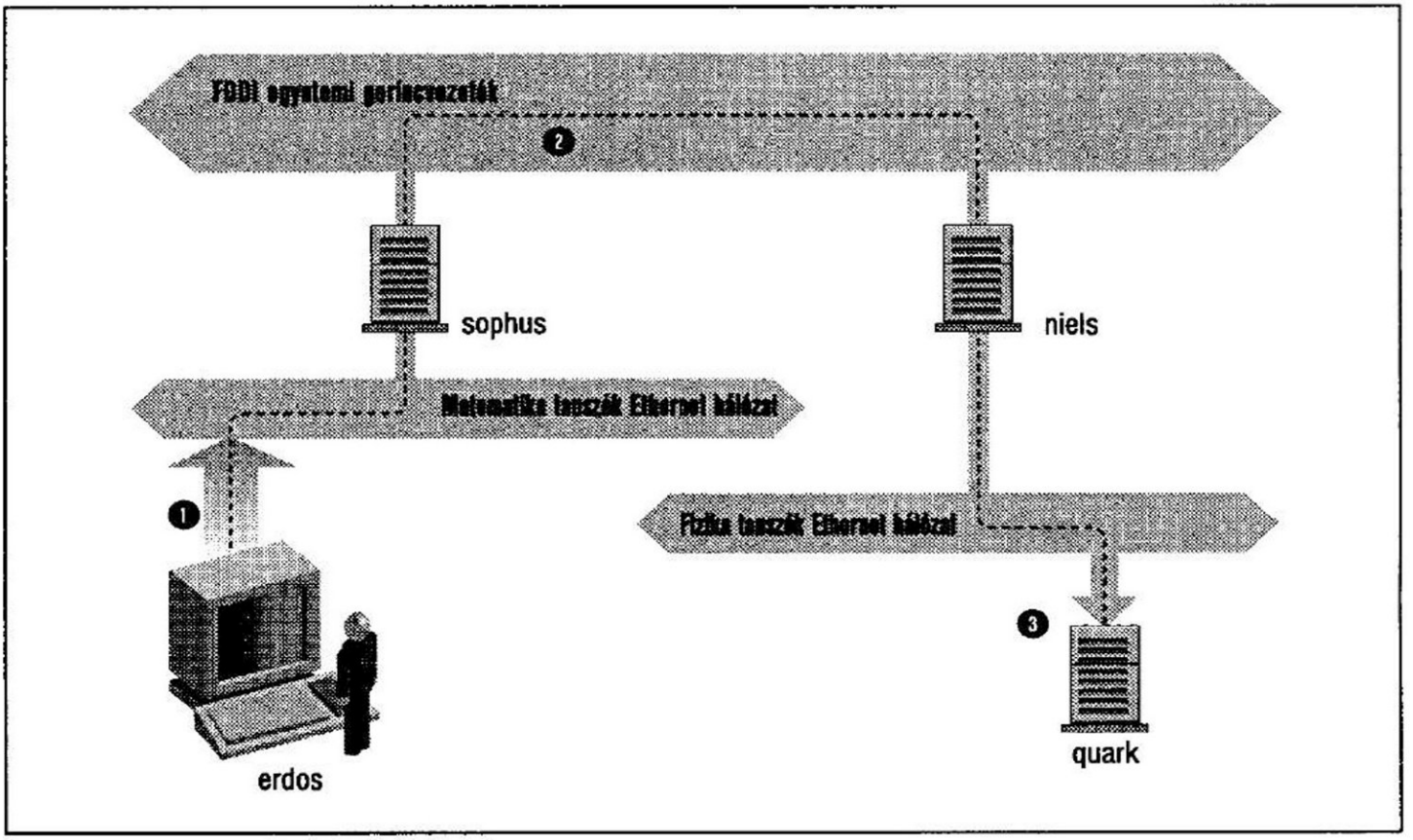
A rádióamatőrök gyakran használják rádiójukat számítógépeik hálózatba kapcsolására; ezt *csomagrádiónak* nevezzük. Az amatőr rádiósok egyik protokollja az X.25 távoli leszármazottja, az AX.25. Az amatőr rádiósok a TCP/IP, illetve más protokollok továbbítására is az AX.25 protokollt alkalmazzák. Az X.25-höz hasonlóan az AX.25 is szinkron üzemre alkalmas soros hardvert igényel, illetve egy *Terminal Node Controller* nevű külső eszközt, mely az aszinkron soros kapcsolaton keresztül továbbított csomagokat szinkron módon átvitt csomagokká alakítja. A csomagrádiót támogató interfészkártyákból számos különböző típus létezik; a felszerelésekben használt legnépszerűbb kommunikációs vezérlő típusa után ezekre általánosságban mint „Z8530 SCC alapú" kártyákra utalunk. Az AX.25 segítségével gyakran továbbított egyéb protokollok közül említsük meg a NetRom és a Rose protokollokat, amelyek hálózatréteg-protokollok. Miután az AX.25-re épülve futnak, azonos a hardverigényük is. A Linux az AX.25, a NetRom és a Rose protokollok teljes tulajdonságkészletű implementációját tartalmazza. Az *AX25 HOWTO* gazdag információt kínál e protokollok linuxos megvalósításáról.

Az egyéb típusú internetes hozzáférések betárcsázó eljárással, lassú, de olcsó soros vonalon oldják meg a központi rendszer elérését (telefon, ISDN és így tovább). A csomagok átviteléhez ezek további protokollt igényelnek: ilyen a SLIP és a PPP, melyekről később szólunk.

Az Internet Protokoll

De hát ki szeretné hálózatát egy Ethernet hálózatra vagy csupán egy két pontot összekötő adatkapcsolatra korlátozni? Ideális esetben jó lenne, ha bármilyen kiszolgálóval képesek volnánk kommunikálni, függetlenül attól, hogy a kiszolgáló fizikailag milyen hálózatra kapcsolódik. Példának okáért, az olyan nagy rendszereken, amilyen a Groucho Marx Egyetem, rendszerint több önálló hálózatra is kapcsolódhatunk. A GME rendszerében a Matematika tanszék két Ethernet hálózatot futtat, egyet a tanárok és tanársegédek gyorsabb, egyet a diákok lassúbb gépeivel.

A kapcsolatot egy speciális célú dedikált kiszolgáló, az *átjáró* (gateway) kezeli: ez másolja át a bejövő és kimenő csomagokat a két Ethernet hálózat és az FDDI száloptikai kábelei között. Ha például a Matematika tanszéken lévő linuxos számítógépünkről szeretnénk elérni a Fizika tanszék lokális hálózatán lévő **quark** gazdagépet, a hálózatkezelő szoftver a csomagokat nem közvetlenül a **quark**ra küldi, hiszen különböző Ethernet hálózaton vannak. Helyette a szoftver az átjáróra bízza a csomagok továbbítását. Az átjáró (a neve legyen **sophus**) a csomagokat ekkor a gerinchálózaton keresztül a Fizika tanszéken üzemelő „társához", a **niels** nevű átjáróhoz továbbítja, mely végül eljuttatja azt a címzett géphez. Az 1.1. ábrán az **erdos** és **quark** közötti adatforgalmat követhetjük nyomon.



1.1. ábra. A datagram útjának három fázisa az erdostól a quarkig

A folyamatot, amelynek során az adatokat a távoli kiszolgálókhoz irányítjuk, *útválasztásnak* (routing) nevezzük, a csomagokat pedig ebben az összefüggésben gyakran datagramnak hívjuk. Azért, hogy a folyamat egyszerűbb legyen, a datagramok átvitelét egyetlen, hardverfüggetlen protokoll végzi: az IP, vagyis az *Internet Protokoll.* A 2. fejezetben részletesen foglalkozunk az Internet Protokollal és az útválasztás kérdéseivel.

Az Internet Protokoll fő előnye, hogy segítségével a fizikailag különböző hálózatokat egyetlen homogén hálózatként kezelhetjük. Az eljárás neve hálózatkapcsolás, a létrejött „metahálózat" pedig egy *internet.* Figyeljük meg a névelő használatát: *egy* internet és *az* Internet. Ez utóbbi kifejezés egy bizonyos globális internet hivatalos elnevezése (ha nem általában az internetről, hanem hivatalosan erről a konkrét globális hálózatról beszélünk, akkor nevét nagy kezdőbetűvel írjuk).

Természetesen az Internet Protokoll a hardvertől független címzési sémát igényel. A megoldás, hogy minden gazdagéphez egy egyedi, 32 bites számot, tulajdonképpen egy *IP címet* rendelünk. Az IP címeket rendszerint négy, ponttal elválasztott decimális számként írjuk fel, melyek mindegyike egy 8 bites résznek felel meg. Ha a **quark** IP címe például **0x954C0C04**, azt a **149.76.12.4** formában írjuk fel. A formátum neve *pontozott decimális jelölés,* de néha *pontozott négyes jelölésnek* is hívják. Ez mindinkább az IPv4-hez (az Internet Protokoll 4-es verziójához) kapcsolódik, hiszen az új szabvány, az IPv6 több más korszerű tulajdonsága mellett jóval rugalmasabb címzést tesz lehetővé. Az IPv6 elterjedésére előreláthatólag a könyv megjelenésétől számítva még legalább egy évig várnunk kell.

Láthatjuk, hogy három különböző címtípussal rendelkezünk: az első a gazdagép neve, például a **quark**; a második az IP cím; és a harmadik a hardvercím, például a 6 bájtos

Ethernet cím. Mindezeknek a címeknek valamiképpen egyezniük kell ahhoz, hogy az *ssh quark* parancs kiadásakor a hálózatkezelő szoftvernek átadhassuk a **quark** IP címét; hasonló módon, amikor az IP eljuttatja az adatokat a Fizika tanszék Ethernet hálózatára, meg kell találnia, melyik Ethernet cím felel meg az IP címnek.

Ezekkel a helyzetekkel a 2. fejezetben foglalkozunk. Most elegendő annyit megjegyeznünk, hogy a gazdagépek nevének leképzését IP címekre a *gépnév meghatározásának* (hostname resolution) nevezzük, az IP címek leképzését hardvercímekre pedig *címfeloldásnak.*

IP a soros vonalon

A soros vonalak „de facto" szabványának neve *Serial Line IP* (SLIP). A SLIP egy módosított változata, a *Compressed SLIP* (CSLIP) tömöríti az IP fejléceket, és így hatékonyabban használja ki a legtöbb soros kapcsolatra jellemző viszonylag alacsony sávszélességet. Egy további soros protokoll neve a *Point-to-Point Protokoll* (PPP). A PPP újabb a SLIP-nél, és számos olyan tulajdonságot sorakoztat föl, melyek vonzó választássá teszik. Fő előnye a SLIP-pel szemben, hogy képességei nem merülnek ki az IP datagramok szállításában, mert kialakításánál fogva szinte bármilyen protokollt továbbíthat. A PPP részletes ismertetésére a 6. fejezetben kerül sor.

Az adatforgalom-vezérlő protokoll (TCP)

Nem csak arról van azonban szó, hogy datagramokat továbbítsunk az egyik gazdagépről a másikra. A **quarkra** bejelentkezve elvárjuk, hogy megbízható kapcsolat jöjjön létre az **erdos** *ssh* folyamata és a **quark** héj folyamata között. Ehhez a küldő a kimenő és bejövő információt csomagokra bontja, a fogadó pedig újra karakterfolyammá állítja össze. Noha mindez egyszerűnek tűnik, számos összetett feladatot foglal magában.

Fontos tudnunk, hogy az Internet Protokoll szándékosan nem megbízható. Tegyük fel például, hogy az Ethernet hálózatunkon tíz felhasználó nekilát, hogy letöltse a Mozilla webböngésző forráskódját a GME FTP kiszolgálójáról. Előfordulhat, hogy a keletkező forgalmat az átjáró már nem képes kezelni, mert túl lassú és nincs elegendő memóriája. Ha ilyenkor történetesen egy csomagot küldünk a **quarkra**, a **sophus** esetleg nem képes azt továbbítani, mert nem áll rendelkezésére elegendő pufferhely. Az IP úgy oldja meg a helyzetet, hogy „eldobja" a csomagot, amely ezért visszavonhatatlanul elvész. Következésképpen a kommunikáló kiszolgáló igen fontos feladata, hogy az adatok integritását és épségét ellenőrizze, és hiba esetén ismételten elküldje a csomagot.

A folyamatért egy további protokoll, az *adatforgalom-vezérlő protokoll* felel (Transmission Control Protocol, TCP), amely az Internet Protokollra ráépülve hoz létre megbízható szolgáltatást. A TCP fő feladata, hogy az IP segítségével a gazdagépünk és a távoli számítógép folyamatai közötti egyszerű kapcsolat illúzióját keltse, így nem kell azzal foglalkoznunk, hogyan és milyen útvonalon utaznak valójában az adatok. A TCP kapcsolat alapvetően egy kétirányú kommunikációs csatorna, melyet mindkét folyamat írhat és olvashat – képzeljük el úgy, mint egy telefonbeszélgetést!

A TCP a kapcsolat két végpontját a kapcsolatban részt vevő gazdagépek IP címe, valamint a gazdagépek *portjának* száma alapján azonosítja. A portokat a hálózati kapcsolatok csatlakozási pontjának is tekinthetjük. A telefon példájánál maradva képzeljük el

úgy, hogy a két kiszolgáló két város, az IP címek pedig a körzetszámok (a számok egy-egy városnak felelnek meg). Ekkor a portszámot felfoghatjuk egy helyi telefonszámként, amelyen a részt vevő felek elérik egymást. Egy gazdagép több eltérő szolgáltatást is nyújthat, és ezek között a szolgáltatásokhoz tartozó portszám alapján teszünk különbséget.

Az *ssh* példában az ügyfélalkalmazás *(ssh)* megnyit egy portot az **erdoson** és csatlakozik a **quark** 22-es portjára, mert tudja, hogy az sshd kiszolgáló ezt a portot figyeli. Ezzel létrejön a TCP kapcsolat, melyen keresztül az *sshd* elvégzi az azonosítást és létrehozza a héjat. A héj szabványos be- és kimenete a TCP kapcsolatra rendeződik át, így a saját számítógépünkön az *ssh* folyamatnak kiadott parancsok a TCP kapcsolaton keresztül mint szabványos bemenet a héjba jutnak.

A felhasználói datagram protokoll (UDP)

Természetesen a TCP/IP hálózatkezelésben a TCP nem az egyetlen felhasználói protokoll. Az *ssh* és a hasonló alkalmazások esetén megfelel ugyan, de olyan többletterhet jelent, melyet más alkalmazások, például az NFS már nem bír el. Az NFS a TCP „rokonát", a *felhasználói datagram protokollt* használja (User Datagram Protocol, UDP). A TCP-hez hasonlóan az UDP is lehetővé teszi, hogy az alkalmazás a távoli számítógép egy adott portjának szolgáltatásához hozzáférjen, ehhez azonban nem létesít kapcsolatot, hanem az önálló csomagokat a kiválasztott szolgáltatáshoz küldi – nevét is erről kapta.

Tegyük fel, hogy csupán néhány adatot szeretnénk egy adatbázis-kiszolgálóról lekérni. A TCP kapcsolat kiépítéséhez legalább három datagram szükséges, majd további három az adatok elküldéséhez és visszaigazolásához mindkét irányban, és ismét három a kapcsolat lezárásához – és mindez csupán néhány adatért! Az UDP segítségével már két datagrammal is hasonló eredményt érhetünk el. Az UDP-t kapcsolat nélküli protokollnak is nevezik, mert nincs szükségünk a kapcsolat létrehozására és lezárására. Adatainkat egyszerűen elhelyezzük egy datagramban és elküldjük a kiszolgálóra; a kiszolgáló előállítja a választ, az adatokat a címünkre szóló datagramba helyezi és visszaküldi számunkra. Noha az egyszerű tranzakcióknál az eljárás gyors és a TCP-nél hatékonyabb, az UDP nem képes kezelni az elveszett datagramokat. Erre az alkalmazásnak, például a névkiszolgálónak kell ügyelnie.

Időzzünk még a portoknál!

A portokat a hálózati kapcsolatok csatlakozási pontjának is tekinthetjük. Ha egy alkalmazás egy bizonyos szolgáltatást akar kínálni, rácsatlakozik egy portra és várja az ügyfeleket (azt is mondjuk, hogy a porton *figyel).* A szolgáltatást igénylő ügyfélalkalmazás lefoglal egy portot a helyi gazdagépen és rákapcsolódik a távoli gazdagép kiszolgálójának portjára. Ugyanaz a port több különböző gépen is nyitva lehet, de egy adott portot egyidejűleg minden gépen csak egy folyamat nyithat meg.

A portok fontos tulajdonsága, hogy az ügyfél és a kiszolgáló közötti kapcsolat létrejöttekor a kiszolgáló további példányai is a portra csatlakozhatnak, hogy további ügyfeleket fogadjanak. Így lehetséges például, hogy egyidejűleg több távoli bejelentkezés is történhet ugyanarra a gazdagépre, ugyanazon a porton. A TCP képes e kapcsolatokat megkülönböztetni, hiszen eltérő portokról, illetve gazdagépekről származnak. Ha például az **erdos** gépről kétszer bejelentkezünk a **quark** gépre, az első *ssh* kliens használhat-

ja a 6464-es helyi portot, a második pedig megnyithatja a 4235-ös helyi portot – ennek ellenére mindkettő a **quark** 513-as portjára kapcsolódik. A két kapcsolat megkülönböztetését az **erdos** gépen használt portszámok teszik lehetővé.

A példában a portokat találkozóhelyként használtuk fel: az ügyfél meghatározott portra kapcsolódott, hogy meghatározott szolgáltatáshoz hozzáférhessen. Ahhoz, hogy az ügyfél ismerje a megfelelő portszámot, a két rendszer adminisztrátorainak meg kell állapodniuk a portszámok kiosztásában. A széles körben keresett szolgáltatások, például az *ssh* esetén a számokat központilag kezelik. Ezt a feladatot az Internet Engineering Task Force (IETF) látja el, amely rendszeresen közreadja az RFC *hozzárendelési számokat* (RFC-1700). A dokumentum többek között a jól ismert szolgáltatások portszámait tartalmazza. A Linux rendszerek az */etc/services* állomány segítségével alakítják a szolgáltatásokat számokká.

Érdemes megjegyeznünk, hogy bár mind a TCP, mind az UDP kapcsolatok a portokra támaszkodnak, a számok nem kerülhetnek összeütközésbe, vagyis a 22-es TCP port nem azonos a 22-es UDP porttal.

A Socket könyvtár

A Unix operációs rendszereken a fent leírt feladatok és protokollok kezelésére szolgáló szoftver általában a rendszermag része, és ez a Linuxon sincs másképpen. A unixos világ leginkább elterjedt programozási felülete a Berkeley Socket Library. Neve egy népszerű analógiából ered, mely a portokat foglalatnak (socket) tekinti, a portokhoz kapcsolódást pedig csatlakoztatásnak (plug in). Az interfész biztosít egy bind hívást a távoli kiszolgáló azonosítására, egy átviteli protokollt és egy szolgáltatást, melyre a programok kapcsolódhatnak vagy figyelhetnek (a connect, a listen és az accept parancsokkal). A socket könyvtár annyival általánosabb, hogy nem csak TCP/IP alapú socket osztályokat tartalmaz (ezek az *AF_INET* foglalatok), hanem olyan osztályt is, amely a számítógép szempontjából helyi kapcsolatnak minősülő kapcsolatok kezelésére szolgál (az *AF_UNIX* osztályt). Egyes megvalósítások további osztályokat is kezelnek, például a Xerox Networking System (XNS) protokollt vagy az X.25 protokollt.

A Linux rendszereken a socket könyvtár a szabványos *libc* C könyvtár része. Támogatja az *AF_INET* és az *AF_INET6* foglalatokat a TCP/IP protokollhoz és az *AF_UNIX* osztályt a Unix domain foglalatokhoz. Szintén támogatja az *AF_IPX* foglalatokat a Novell hálózati protokollokhoz, az *AF_X25* foglalatot az X.25 hálózati protokollhoz, az *AF_ATMPVC* és az *AF_ATMSVC* foglalatokat az ATM hálózati protokollokhoz, valamint az *AF_AX25*, az *AF_NETROM* és az *AF_ROSE* foglalatokat az amatőrrádió-protokollokhoz. Jelenleg további protokollcsaládok fejlesztése is folyik, melyek idővel elérhetők lesznek.

Hálózatkezelés a Linuxon

Minthogy a Linux a világ különböző részein munkálkodó programozók egyesített erőfeszítésének eredménye, nem jöhetett volna létre a globális hálózat nélkül. Nem meglepő tehát, hogy már a fejlesztés korai szakaszában elkezdtek dolgozni a hálózati lehetőségek kialakításán is. Egy UUCP implementáció szinte már a legelső pillanattól futott a

Linuxon, a TCP/IP alapú hálózatkezelés fejlesztése pedig 1992 őszén indult el, amikor Ross Biro többedmagával megalkotta azt, amit ma Net-1 néven ismerünk.

Miután Ross 1993 májusában felhagyott az aktív fejlesztéssel, Fred van Kempen egy új megvalósításon kezdett dolgozni, átírva a kód jelentős részét. Ez Net-2 néven vált ismertté. Az első nyilvános kibocsátás, a Net-2d 1993 nyarán jelent meg (a 0.99.10 kernel részeként), és karbantartásával, bővítésével azóta is többen foglalkoznak – és itt elsősorban Alan Cox nevét kell megemlítenünk. Alan az eredeti programját Net-2Debugged névre keresztelte, azonban számos hiba kijavítása és még több fejlesztés után – a Linux 1.0 megjelenésekor – nevét Net-3-ra változtatta. A Net-3 további fejlesztésen ment keresztül, amikor a Linux 1.2 és a Linux 2.0 megjelent. A 2.2 és újabb verziójú rendszermagok a hálózati támogatáshoz a Net-4 változatot alkalmazzák, mely jelenleg is a hivatalosan ajánlott változat.

A Net-4 Linux Network az eszközmeghajtók széles skáláját tartalmazza és fejlett tulajdonságkészletet kínál. A szabványos Net-4 protokollok között a következőket találjuk: SLIP és PPP (soros vonalon bonyolított hálózati forgalomhoz), PLIP (párhuzamos vonalakhoz), IPX (Novell rendszerű hálózatokhoz), Appletalk (Apple hálózatokhoz), valamint az AX.25, a NetRom és a Rose (rádióamatőr-hálózatokhoz). A szabványos Net-4 támogatja továbbá az IP tűzfalat (melyről a 7. fejezetben szólunk), az IP nyomkövetést (8. fejezet), az IP álcázást (9. fejezet), az IP továbbítás különböző változatait és a fejlett szabályrendszerű útválasztást. Nagyszámú Ethernet eszköz vezérlésére képes, és az FDDI, a Token Ring, a Frame Relay, az ISDN és az ATM kártyákat is támogatja.

Mindemellett a Linux rugalmasságát számos további lehetőség biztosítja; ilyen például a Microsoft Windows hálózati környezettel való együttműködés képessége a Samba segítségével (16. fejezet), illetve a Novell NCP (NetWare Core Protocol) implementációja.*

Különböző fejlesztési törekvések

Az elmúlt években a Linux hálózatfejlesztési kísérletei különböző irányokat követtek. Miután a Net-2Debugged hivatalos hálózati implementációvá vált, Fred tovább folytatta a fejlesztését. Ez vezetett a Net-2e-hez, ami a hálózati réteg alapos átalakítását vonta maga után. Fred célja egy szabványosított eszközmeghajtó interfész (Device Driver Interface, DDI) létrehozása volt, a Net-2e munkálatai azonban mára abbamaradtak.

A TCP/IP hálózatkezelés egy másik megvalósítása Matthias Urlichs nevéhez fűződik, aki ISDN meghajtót készített a Linux és a FreeBSD rendszerekhez. A meghajtóhoz a Linux rendszermag BSD hálózatkezelő kódjának egyes részeit integrálta. Ma már ez a fejlesztés is leállt.

A Linux rendszermag hálózatkezelésében számos gyors változás zajlott le, és a fejlesztést ma is a változások jellemzik. Ennek következtében persze változik a többi szoftver is, például a hálózati konfigurációs eszközök. Noha jelenleg ez már nem okoz akkora gondot, mint régen, előfordulhat, hogy a rendszermag frissítésekor a hálózati konfigurációs eszközöket is frissítenünk kell. Szerencsére egyszerű a feladatunk, ha figyelembe vesszük az elérhető Linux-disztribúciók nagy számát.

A szabvány jelenleg a Net-4 implementáció, amelyet a világon valóban nagyon sok hálózati hely alkalmaz. A Net-4 implementáció teljesítményének javítására hatalmas erő-

* A Novell állomány- és nyomtatásszolgáltatásai az NCP protokollon alapulnak.

feszítéseket tesznek, így ma már az adott hardverplatformok legjobban sikerült másfajta megvalósításaival is felveszi a versenyt. A Linux jelenleg egyre jobban terjed az internetszolgáltatói környezetben, mert az ilyen tevékenységet folytató szervezetek gyakran futtatják a Linuxot olcsó és megbízható World Wide Web-, levelező- vagy hírkiszolgálóként. A Linux tökéletesítése ma már elegendő fejlesztőt vonz ahhoz, hogy a rendszer napra készen kövesse a hálózatkezelési technológia változásait. A Linux rendszermag legújabb kiadásaiban jelenleg szabványos összetevőként jelenik meg az IP protokoll legújabb generációja, az IPv6, mellyel részletesen a 13. fejezetben ismerkedünk meg.

Hol szerezhető be a program?

Utólag visszatekintve különösnek tűnik, hogy a Linux hálózatkezelő szoftverének korai fejlesztési szakaszában a szabványos rendszermaghoz hatalmas méretű javítócsomagot kellett letöltenünk, ha a hálózatkezelési lehetőséget biztosítani kívántuk. Ma már a hálózati fejlesztés a Linux kernelfejlesztési folyamatának alapvető része. A legújabb stabil Linux rendszermagokat megtaláljuk az *ftp://ftp.kernel.org* címen, a */pub/linux/kernel/v2.x/* könyvtárban, ahol az *x* mindig egy páros szám. A legújabb kísérleti Linux kernelek az *ftp://ftp.kernel.org* címen, a */pub/linux/kernel/v2.y/* könyvtárban pihennek, ahol az *y* mindig egy páratlan szám. A kernel.org disztribúciókat HTTP-n keresztül a *http://www.kernel.org* útvonalon találjuk meg. A Linux rendszermagok forráskódját szerte a világon tükröroldalakról is elérhetjük.

A rendszer karbantartása

A könyv jelentős része telepítési és beállítási kérdésekkel foglalkozik. Az adminisztráció azonban ennél többet jelent – egy szolgáltatás beindítása után az üzemeltetésről is gondoskodnunk kell. A legtöbb szolgáltatás csupán egy kis figyelmet igényel, mások pedig, mint például a levelezés napról napra ismétlődő teendők elvégzését kívánják. E feladatokkal a későbbi fejezetekben találkozunk.

A karbantartási feladatok abszolút minimuma, hogy a rendszernaplók és az egyes alkalmazások naplóállományainak szisztematikus felülvizsgálatával feltárjuk a hibák körülményeit és a szokatlan eseményeket. A feladatot gyakran néhány adminisztratív héjszkripttel oldjuk meg, amelyeket a *cron*-ból rendszeresen futtatunk. A nagyobb alkalmazások forrásállományában gyakran találunk ilyen szkripteket. Elegendő, ha ezeket módosítjuk, figyelembe véve igényeinket és elvárásainkat.

Bármely *cron* feladat kimenetét érdemes egy adminisztratív fiókba postázni. Sok alkalmazás a hibajelentéseket, a felhasználási statisztikákat és a naplóállományok összefoglalóit alapértelmezés szerint a *root* fiókba küldi. Ennek csak akkor van értelme, ha gyakran jelentkezünk be *root* felhasználóként; sok esetben érdemesebb a *root* levelezését saját személyes fiókunkba továbbítani egy alias beállításával – erről a 11. és 12. fejezetben olvashatunk bővebben.

Bármennyire gondosan állítottuk is be hálózati helyünket, Murphy törvénye értelmében valami gondunk előbb-utóbb *biztosan* akad. Ezért a rendszer karbantartása magában foglalja a panaszok és panaszosok meghallgatását is. A felhasználók rendszerint elvárják, hogy a rendszergazda elektronikus levélben elérhető legyen a *root* néven, de a

karbantartás specifikus feladatainak megfelelően más gyakran használt címek is léteznek a felelős személyek megkeresésére. Így például a hibás levelezőprogramról a *postmaster* címet értesítjük, a hírrendszer gondjairól a *newsmaster* vagy *usenet* címet tájékoztatjuk. A *hostmaster* leveleit ahhoz a személyhez irányítjuk, aki a gazdagép alapvető hálózati szolgáltatásaiért és – ha névkiszolgálót futtatunk – a DNS névszolgáltatásért felel.

A rendszerbiztonságról

Hálózati környezetben a rendszeradminisztráció másik fontos feladata a rendszer és a felhasználók védelme a behatolók ellen. A gondatlanul kezelt rendszerek számos támadási felületet kínálnak a támadóknak. A támadások „műfaja" a jelszavak visszafejtésétől az Ethernet kifürkészéséig terjedhet, az okozott kár pedig egy meghamisított e-mail üzenettől kezdve az adatvesztésig vagy felhasználóink személyes állományainak megsértéséig többféle kártétel lehet. A támadások körülményeit tárgyalva kitérünk majd néhány valóságos esetre és az elhárítás lehetőségeire is.

A következőkben a rendszerbiztonsággal kapcsolatban láthatunk néhány példát és elemi eljárást. Természetesen a tárgyalt témakörök nem foglalkozhatnak részletesen valamennyi biztonsági kérdéssel, csupán azokat a nehézségeket illusztrálják, melyekkel nap mint nap szembesülünk. Ezért feltétlenül olvassunk el néhány jó biztonsági szakkönyvet is, különösen ha hálózati rendszerrel dolgozunk.

A rendszerbiztonság alapja a jó rendszeradminisztráció. Ebbe beletartozik a létfontosságú állományok és könyvtárak tulajdonjogának és engedélyeinek ellenőrzése, és a különleges jogosultságokkal rendelkező felhasználói fiókok tevékenységének megfigyelése. A COPS program például ellenőrzi fájlrendszerünket, valamint az általános konfigurációs állományokat, szokatlan jogosultságok és egyéb anomáliák után kutatva. Egy másik eszköz, a Jay Beale által fejlesztett és a *http://www.bastille-linux.org* címről letölthető Bastille Linux olyan szkripteket és programokat tartalmaz, melyekkel lezárhatjuk a Linux rendszert. Érdemes továbbá erős felhasználói jelszavakat megkövetelő jelszócsomagot használni, hogy pusztán találgatással ne lehessen azokat feltörni. Ma már a Linux terjesztések állandó összetevője az árnyékjelszó csomag, amely megköveteli, hogy egy jelszó legalább öt betűt tartalmazzon, legyenek benne kis- és nagybetűk, valamint nem alfabetikus karakterek.

Amikor egy szolgáltatást a hálózaton keresztül teszünk elérhetővé, a lehető legkevesebb jogosultságot biztosítsuk a számára; ne adjunk neki olyan engedélyeket, amelyek a tervezett működéshez nélkülözhetők. Így például a **root** vagy más privilegizált felhasználó birtokában lévő programok suid bitjét csak akkor állítsuk be, ha valóban szükséges. Ha egy szolgáltatást csak korlátozott alkalmazásra szánunk, ne habozzunk az alkalmazás által megengedett legerősebb korlátozásokat beállítani. Ha például engedélyezni szeretnénk, hogy a lemez nélküli kiszolgálók a mi gépünket használják rendszerindításra, biztosítanunk kell számukra a Trivial File Transfer Protocolt (TFTP), mellyel az alapvető konfigurációs állományokat letölthetik */boot* könyvtárunkból. A korlátozások nélkül alkalmazott TFTP azonban lehetővé teszi, hogy a felhasználók a világ bármely pontjáról letölthessék a rendszerünk bárki által olvasható állományait. Ezt elkerülendő, korlátozzuk a TFTP szolgáltatást a */boot* könyvtárra (erre a 10. fejezetben visszatérünk). Az egyes szolgáltatásokat külön-külön is korlátozhatjuk, hogy csak bizonyos gazdagépek férhessenek hozzájuk, mondjuk a helyi hálózatunkról. A *tcpd*-vel, mely ezt teszi le-

hetővé különböző hálózati alkalmazások esetén, a 10. fejezetben ismerkedünk meg. Az adott gazdagépekhez vagy szolgáltatásokhoz történő hozzáférés korlátozásának kifinomultabb eljárásaival a 7. fejezetben foglalkozunk.

Fontos szempont a „veszélyes" programok elkerülése. Természetesen bármely programunk veszélyessé válhat, hiszen lehetnek olyan hibái, melyeket kihasználva az éles eszű támadók behatolhatnak rendszerünkbe. Ez sajnos bármikor megtörténhet, és nincs ellene megbízható védekezés. A probléma a szabad és a kereskedelmi szoftvereket egyaránt érinti.* A különleges jogosultságokat igénylő programok azonban eredendően veszélyesebbek a többinél, mert bármely „kiskapunak" drasztikus következményei lehetnek.** Ha setuid programot telepítünk hálózati célra, kétszer is olvassuk el a leírását, nehogy véletlenül biztonsági rést nyissunk rendszerünkön.

Figyelnünk kell az olyan programokra is, amelyek korlátozott hitelesítés mellett lehetővé teszik a bejelentkezést vagy a parancsok futtatását. Az *rlogin, rsh* és a *rexec* parancsok nagyon hasznosak, de a hívó fél azonosítása szempontjából rendkívül gyengék. A hitelesítés a hívó fél névkiszolgálóról beszerzett gépnevébe vetett bizalmon alapul (erről még lesz szó), a gazdagép neve azonban hamisítható. Az *r* parancsokat mindig és teljes egészében iktassuk ki, *ssh* eszközökkel helyettesítve azokat. Az *ssh* eszközök megbízhatóbb hitelesítési eljárásokat használnak és más szolgáltatásokat is biztosítanak, például a titkosítást és a tömörítést.

Mindig számítsunk arra, hogy elővigyázatosságunk nem hozza meg a várt eredményt, függetlenül attól, milyen óvatosak voltunk. Ezért a behatolás észlelése kivételesen fontos. Kezdeti lépésként ellenőrizzük a rendszer naplóállományait, noha a behatoló valószínűleg elég óvatos, hogy erre felkészüljön, így minden nyilvánvaló nyomot eltüntet maga után. Az olyan eszközök azonban, mint például a Gene Kim és Gene Spafford által fejlesztett *tripwire* lehetővé teszik számunkra, hogy a létfontosságú rendszerállományokat megvizsgálva felismerjük, ha tartalmukat vagy jogosultságaikat megváltoztatták. A *tripwire* különféle megbízható ellenőrzőösszegekkel figyeli az említett állományokat, az eredményeket pedig adatbázisban tárolja. Az egymást követő futtatások alkalmával az ellenőrzőösszegeket ismételten kiszámítja és összeveti a tárolt adatokkal, hogy észlelje az esetleges módosításokat.

A biztonság alapja a megelőzés. Fontos, hogy a futtatott programok frissítéseit és javításait nyomon kövessük a levelezőlistákon, és mindig naprakészek legyünk. Ha elmulasztjuk az Apache, az OpenSSL vagy egy hasonló jelentőségű szoftver frissítését, mulasztásunk közvetlenül a rendszer kompromittálásához vezethet. Időszerű példája ennek a Linux Slapper féreg, mely az OpenSSL sebezhetőségét kihasználva terjedt. Bár a frissítések követése fárasztó és időrabló feladatnak látszik, azok a rendszergazdák, akik idejében frissítették OpenSSL szoftverüket, rengeteg időt takarítottak meg azzal, hogy nem kellett a kompromittált rendszer helyreállításával bíbelődniük!

* Léteztek olyan, a kereskedelmi forgalomban hatalmas összegért kapható Unix rendszerek, melyek setuid root héjszkripttel kerültek forgalomba, így a felhasználók **root** jogosultsághoz juthattak egy egyszerű trükk segítségével.

** 1988-ban az RTM féreg az internet nagy részén váratlan leállást okozott, részben bizonyos programok, például a *sendmail* biztonsági rései miatt. A rést azóta természetesen „betömték".

2

A TCP/IP hálózatkezelés kérdései

A fejezetben azokat a beállítási kérdéseket tárgyaljuk, melyekkel a Linux számítógép TCP/IP hálózatra csatlakoztatásakor találkozunk; szó lesz többek között az IP címekről, a gépnevekről és az útválasztásról. A fejezetből elsajátíthatjuk a beállításhoz szükséges ismereteket, a következő fejezetekben pedig a felhasznált eszközökről szólunk.

A TCP/IP hátteréről és elvéről bővebben Douglas R. Comer háromkötetes könyvében olvashatunk: *Internetworking with TCP/IP* (kiadója a Prentice Hall). A TCP/IP hálózatok kezeléséhez részletes útmutatót kínál Craig Hunt *TCP/IP Network Administration* című könyve (kiadója az O'Reilly).

A hálózatkezelő interfészek

Azért, hogy a hálózati környezet különböző berendezéseit a felhasználók elől elrejtse, a TCP/IP egy absztrakt interfészt definiál; a hardvert ezen keresztül érhetjük el. Az interfész kínálta műveletek minden hardvertípus esetén azonosak, és elsősorban a csomagok küldésére és fogadására alkalmasak.

A rendszermagban valamennyi periferikus hálózatkezelő eszközhöz tartozik interfész. A Linuxban például az Ethernet interfészeket *eth0* és *eth1*, a PPP interfészeket (lásd a 6. fejezetet) *ppp0* és *ppp1*, az FDDI interfészeket pedig *fddi0* és *fddi1* néven nevezzük. Az interfész neve konfigurációs célokat szolgál: akkor használjuk, amikor egy konfigurációs parancsban egy bizonyos fizikai eszközre szeretnénk hivatkozni – ezen túlmenően semmilyen szerepük nincs.

Mielőtt a TCP/IP hálózatkezelésben hasznukat vehetnénk, az interfészekhez IP címet kell rendelnünk; ezzel azonosítjuk, amikor a hálózat egyéb részeivel kommunikálunk. Ez a cím nem azonos az előbb említett interfésznévvel; ha az interfész az ajtó, a cím a rászegezett névtábla.

Az eszköz más paramétereit is beállíthatjuk, például hogy egy hardveregység milyen nagy datagram feldolgozására lehet képes. Ez utóbbinak a neve *Maximum Transfer Unit* (MTU). A későbbiekben további paraméterekkel is megismerkedünk. Szerencsére a legtöbb paraméternek van megfelelő alapértelmezett beállítása.

Az IP címek

Ahogy az 1. fejezetben már említettük, az IP hálózatkezelő protokoll 32 bites számokként értelmezi a címeket. A hálózati környezetben minden kiszolgálónak egyedi címe van. Amennyiben olyan lokális hálózatot üzemeltetünk, amely nem bonyolít TCP/IP forgalmat más hálózatokkal, a számokat tetszésünk szerint jelölhetjük ki. Az IP címek bizonyos tartományait pontosan az ilyen hálózatok számára tartják fenn. A 2.1. táblázat a címtartományokat foglalja össze. Az internetre csatlakozó hálózati helyek számait azonban egy központi felügyeleti szerv, a *Network Information Center* (NIC) osztja ki.

A könnyebb olvashatóság érdekében az IP címeket négy darab nyolc bites részre, úgynevezett *oktetre* osztjuk. Példának okáért, a **quark.physics.groucho.edu** IP címe **0x954C0C04**, amit **149.76.12.4** alakban írunk fel. E formátumot *pontozott négyes jelölésnek* is nevezzük.

A jelölés másik oka, hogy az IP címek két részből állnak: az első oktetekben szereplő *hálózatszámból*, és a maradékot kitevő *gazdagépszámból*. Amikor a NIC-től IP címet kérünk, nem kapunk valamennyi tervbe vett gazdagéphez egy-egy címet; helyette egyetlen hálózatszámot kapunk, és e tartományban azután már szabadon oszthatunk ki érvényes IP címeket a hálózatunk gazdagépei számára.

A cím gazdagépre eső mérete a hálózat méretétől is függ. Ahhoz, hogy a különböző igényeket ki lehessen elégíteni, több hálózatosztályt definiáltak eltérő IP cím felosztással.

A osztály

Az A osztály az **1.0.0.0** és **127.0.0.0** közötti hálózatokat tartalmazza. A hálózatszámot az első oktet adja. Az osztályban a gazdagépszámra 24 bit jut, így közel 1,6 millió állomás csatlakozhat ilyen hálózatokra.

B osztály

A B osztály a **128.0.0.0** és **191.255.0.0** közötti hálózatokat foglalja magában. A hálózatszámot az első két oktet adja. Az osztály 16 320 hálózatot tesz lehetővé, egyenként 65 024 állomás csatlakoztatása mellett.

C osztály

A C osztály a **192.0.0.0** és **223.255.255.0** közötti hálózatokat tartalmazza. A hálózatszámot az első három oktet határozza meg, lehetőséget adva közel 2 millió, egyenként legfeljebb 254 állomást magában foglaló hálózat létesítésére.

D, E és F osztályok

A **224.0.0.0** és **254.0.0.0** tartományba eső címek vagy kísérletiek, vagy különleges célra fenntartottak és nem jelölnek hálózatot. E tartományból választják ki a címeket a többcímes üzenetszórás, az IP Multicast részére, amellyel az internet számos pontjára egyidejűleg küldhetünk adatokat.

Az 1. fejezet példájához visszatérve azt találjuk, hogy a **quark** **149.76.12.4** címe a **149.76.0.0** B osztályú hálózat **12.4** állomását jelenti.

Észrevehettük, hogy az előbbi listában a gazdagépszám oktetjeiben nem minden lehetséges szám megengedett. Ennek oka, hogy a **0** és a **255** oktetek különleges célt szol-

gálnak. Az a cím, amelyben a gazdagépszám csupa 0 bitet tartalmaz, a hálózatot jelzi, az a cím pedig, amelyben a gazdagépszám csupa 1 értékű bitből áll, a *csoportos üzenetszóró cím* (broadcast address), amellyel egyidejűleg küldhetünk adatokat az adott hálózat valamennyi állomására. Ilyen értelemben tehát a **149.76.255.255** nem érvényes gazdagépcím, hanem egyszerre utal a **149.76.0.0** hálózat valamennyi gazdagépére.

Különleges célokat szolgáló hálózati címek is léteznek. A **0.0.0.0** és a **127.0.0.0** is ilyen különleges cím – az előbbit *alapértelmezett útvonalnak,* az utóbbit *visszacsatolási címnek* hívjuk. Az alapértelmezett útvonal helyfoglaló a lokális hálózatot a külvilággal összekötő útválasztó számára.

A **127.0.0.0** hálózat a gazdagépünk számára helyinek minősülő IP forgalomat bonyolítja. A **127.0.0.1** címet rendszerint a gazdagépünk egy különleges interfészéhez, a *visszacsatoló interfészhez* rendeljük, amely egy zárt áramkörhöz hasonlóan működik. A TCP-ről vagy UDP-ről erre az interfészre jutó IP csomagok úgy viselkednek, mintha egy hálózatról érkeztek volna. Ezért az interfész lehetővé teszi a hálózatkezelő szoftverek fejlesztését és tesztelését anélkül, hogy „valódi" hálózatra volna szükségünk. A visszacsatoló hálózat segítségével ráadásul önálló gazdagépeken is futtathatunk hálózatkezelő szoftvereket. Nem is annyira ritka ez, mint első hallásra tűnik: a MySQL-hez hasonló szolgáltatások például, amelyeket csak a kiszolgáló egyéb rezidens alkalmazásai használhatnak, a helyi gazdagép interfészén keresztül további biztonsági réteget alakíthatnak ki.

A címtartományok bizonyos részeit minden hálózatosztályban „fenntartott" vagy „privát" címtartománynak nyilvánították. Az RFC-1918 címként is ismert címeket a magánhálózatok használják, ezekre útválasztáskor nem irányul forgalom az internetről. Általában szervezetek alkalmazzák intranetek kiépítésére, de a kis hálózatok számára is értékes lehet. A fenntartott hálózati címeket a 2.1. táblázat foglalja össze.

2.1. táblázat. Privát hasznosításra fenntartott IP címtartományok

Osztály	*Hálózatok*
A	10.0.0.0 és 10.255.255.255 között
B	172.16.0.0 és 172.31.0.0 között
C	192.168.0.0 és 192.168.255.0 között

Útválasztás osztályok nélkül

A 4. fejezetben részletesen is tárgyalt osztályokat nem alkalmazó, tartományok közötti útválasztás (Classless Inter-Domain Routing, CIDR) az IP címek kiosztásának újabb és hatékonyabb módja. A CIDR segítségével a hálózati adminisztrátorok akár két IP címet tartalmazó hálózatokat is létrehozhatnak, ellentétben a korábbi eljárással, ahol egy C blokkhoz a teljes 254 cím járt. A CIDR kifejlesztésének több oka is volt, melyek közül a legfontosabb az IP címek gyors kimerülése és a globális útvonalválasztó táblázatok különféle, kapacitással kapcsolatos kérdései.

A CIDR címeket már új jelöléssel írjuk, ennek neve CIDR blokkjelölés. Lássunk egy példát! A **172.16.0.0/24** a **172.16.0.0** és a **172.16.0.255** közé eső címtartomány. A jelölésben a 24 azt jelenti, hogy 24 címbit van beállítva, így a 32 bites IP címből 8 bitet használhatunk fel. Ha például szeretnénk csökkenteni a címek számát a tartományban, a címbitekhez hozzáadhatunk még hármat, így a **172.16.0.0/27** hálózati címhez jutunk. Ennek

megfelelően most 5 felhasználható gazdagépbittel rendelkezünk, vagyis összesen 32 címmel. A CIDR címek mindemellett a C osztálynál nagyobb tartományok létrehozására is alkalmasak. Az előbbi 24 bites hálózatból például két bit elvételével a **172.16.0.0/22** hálózatot kapjuk, ahol a hálózati térben 1024 címet oszthatunk ki, ami négyszer több a hagyományos C osztályú térnél. A 2.2. táblázat néhány gyakori CIDR konfigurációt ismertet.

2.2. táblázat. Gyakori CIDR blokkjelölések

CIDR blokk prefixum	*Gazdagépekre jutó bitek*	*Címek száma*
/29	3 bit	8
/28	4 bit	16
/27	5 bit	32
/25	6 bit	128
/24	8 bit	256
/22	10 bit	1024

Címfeloldás

Most, hogy láttuk, hogyan készülnek az IP címek, felmerülhet a kérdés, hogyan használhatjuk ezeket Ethernet vagy Token Ring hálózatokon a különböző gazdagépek címzésére. Végül is e protokollok saját címzési rendszerrel azonosítják a gazdagépeket, és ugye ennek a rendszernek semmi köze az IP címekhez? Így is van.

Egy mechanizmusra van szükség, amely az IP címeket a mögöttes hálózat címeire képezi le. A mechanizmus neve *Address Resolution Protocol* (ARP). Az ARP tulajdonképpen nem korlátozódik Ethernet vagy Token Ring hálózatokra, más hálózattípusokon is alkalmazzák, például a rádióamatőr-hálózatok AX.25 protokollján. Az ARP mögött rejtőző elképzelés pontosan azonos azzal, amit a legtöbb ember gondolna, ha szeretné megtalálni X urat egy 150 fős tömegben: jó hangosan elkiáltaná X úr nevét, hogy mindenki hallja, és várna, feltételezve, hogy ha X úr jelen van, válaszolni fog. A válasz alapján már azonosítani tudná X urat.

Egy adott IP címhez tartozó Ethernet cím azonosításához az ARP az Ethernet egyik lehetőségét, a *csoportos üzenetszórást* (broadcasting) aknázza ki, melynek lényege, hogy a hálózat összes állomásának egyszerre küldünk el egy datagramot. Az ARP szétszórt datagramja tartalmazza az IP címre vonatkozó kérést. Az üzenetet fogadó valamennyi gazdagép összeveti a kérésben szereplő címet a saját IP címével, és egyezés esetén ARP választ küld az érdeklődőnek. Az érdeklődő gazdagép a válaszból most már kiemelheti a küldő Ethernet címét. Ha szeretnénk megismerni hálózatunk ARP címeit, jól jöhet az *arp* segédprogram. Kapcsolók nélkül futtatva a parancs a következőhöz hasonló kimenettel tér vissza:

```
vbrew root # arp
Address           HWtype    HWaddress           Flags Mask      Iface
172.16.0.155      ether     00:11:2F:53:4D:EF   C               eth0
172.16.0.65       ether     00:90:4B:C1:4A:E5   C               eth0
vlager.vbrew.com  ether     00:10:67:00:C3:7B   C               eth1
172.16.0.207      ether     00:0B:DB:53:E7:D4   C               eth0
```

A segédprogrammal a hálózat egy meghatározott gazdagépének ARP címét is lekérhetjük, de arra is alkalmas, hogy segítségével a hálózati adminisztrátorok a gyorsítótárban tárolt ARP bejegyzéseket módosítsák, azokat eltávolítsák vagy újat fűzzenek hozzájuk.

Beszéljünk még egy kicsit az ARP-ról! Miután a gazdagép kiderítette a kívánt Ethernet címet, eltárolja azt saját ARP gyorsítótárában, hogy legközelebb, amikor ugyanarra a gazdagépre szeretne datagramot küldeni, ne kelljen a címet ismét lekérdeznie. Annak azonban nincs értelme, hogy az információt a végtelenségig raktározza, hiszen ha a távoli gazdagép Ethernet kártyáját - például technikai okok miatt - kicserélik, az ARP bejegyzés érvényét veszti. Ezért az ARP gyorsítótár bejegyzései idővel törlődnek, így a gazdagépnek ismét le kell kérdeznie az IP címet.

Esetenként szükség lehet egy Ethernet címhez tartozó IP cím kiderítésére is. Ez történik akkor, amikor egy lemez nélküli gép a hálózat egyik kiszolgálójáról végzi a rendszerindítást, ami a LAN hálózatokon megszokott jelenség. A lemez nélküli ügyfél azonban gyakorlatilag semmit nem tud önmagáról - Ethernet címe kivételével! Ezért egy üzenetet szór szét, melyben kéri a rendszerindítást végző kiszolgálót, hogy jelöljön ki számára egy IP címet. Ezt a feladatot látja el a *Reverse Address Resolution Protocol* (RARP), amely BOOTP protokollal egyetemben arra szolgál, hogy a hálózat lemez nélküli ügyfeleinek indítási eljárását szabályozza.

IP útválasztás

Térjünk most át arra a kérdésre, hogyan találhatjuk meg IP címük alapján a datagramok címzettjeit! A cím egyes részeit eltérő módon kezeli a rendszer; a mi feladatunk az egyes részek kezelési módját meghatározó állomány létrehozása.

IP hálózatok

Amikor levelet írunk, a címzésben feltüntetjük az országot, a várost és az irányítószámot. Bedobjuk a levelesládába, ahonnan a postahivatal feladata eljuttatni a címzetthez: a levél a jelzett országba kerül, ahol a nemzeti postaszolgálat viszi el a kívánt városba és utcába. A hierarchikus séma előnye nyilvánvaló: bárhová küldünk is levelet, a postahivatal tudja, milyen irányban kell a levelet elindítania, de azzal már nem kell foglalkoznia, hogy a kívánt országot elérve milyen útvonalon kézbesítik ki.

Az IP hálózatok felépítése hasonló. Az internet rengeteg önálló hálózatból épül fel, ezeket *autonóm rendszereknek* hívjuk. Az egyes rendszerek a gazdagépeik közötti útválasztást belsőleg oldják meg, így a datagram célba juttatásának feladata a címzett gazdagép hálózatához vezető útvonal kiderítésére korlátozódik. Ebből következik, hogy amint egy datagram egy adott hálózat *bármely* gazdagépéhez érkezik, a további feldolgozást már kizárólag ez a hálózat végzi.

Alhálózatok

Az IP címek korábban tárgyalt felosztása gazdagépszámra és hálózatszámra ezt a szerkezetet tükrözi. Alapértelmezés szerint a rendeltetési hálózat az IP cím hálózati részéből származtatható, vagyis az azonos IP hálózatszámmal rendelkező gazdagépek egyazon hálózaton helyezkednek el.*

* Az autonóm rendszerek kicsit általánosabbak, több IP hálózatot is tartalmazhatnak.

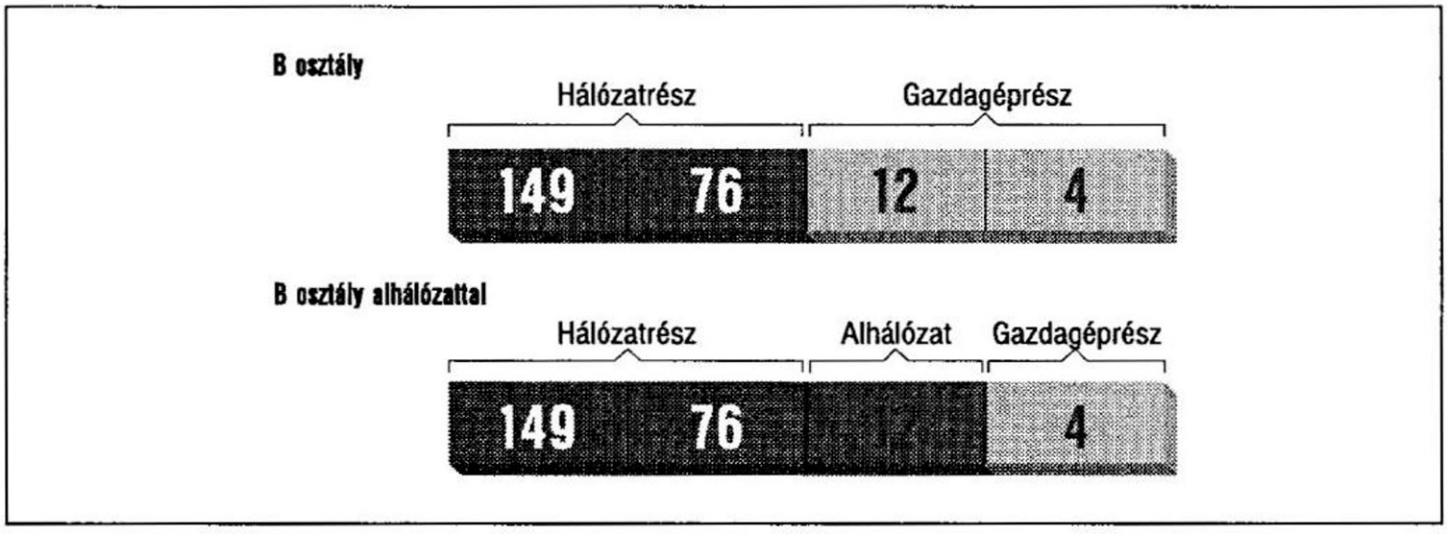
A hálózaton *belül* is érdemes hasonló sémát követni, hiszen a hálózat akár több száz kisebb hálózatot is magában foglalhat, ahol a legkisebb fizikai egységek olyan hálózatok, mint például az Ethernet. Ezért az IP protokoll lehetővé teszi, hogy az IP hálózatokat *alhálózatokra* osszuk fel.

Egy alhálózat az IP címek egy bizonyos tartományáért felel, feladata a datagramok célba juttatása e tartományban. Tulajdonképpen az A, B és C osztályoknál megismert bitmezőfelosztás koncepciójának kiterjesztéséről van szó, noha a hálózatrész most a gazdagéprész egyes bitjeit is tartalmazza. Az alhálózatszámként értelmezett bitek számát az úgynevezett *alhálózatmaszk* vagy *hálózatmaszk* adja meg. Ez szintén egy 32 bites szám, amely meghatározza az IP cím hálózati részének a bitmaszkját.

A Groucho Marx Egyetem (GME) hálózata jól példázza ezt a hálózatot: B osztályú hálózatszáma **149.76.0.0**, így a hálózatmaszk **255.255.0.0**.

A GME egyetemi hálózata belsőleg több kisebb hálózatból épül fel, például a különféle tanszékek lokális hálózataiból (LAN). Ezért az IP címek tartománya 254 alhálózatra bomlik, **149.76.1.0**-től **149.76.12.0**-ig. Az Elméleti fizika tanszék címe például **149.76.12.0**. Az egyetemi gerinchálózat önmagában teljes értékű hálózat, melynek címe **149.76.1.0**. Az alhálózatok azonos IP hálózatszámon osztoznak, megkülönböztetésük pedig a harmadik oktetre hárul, ezért az alhálózatmaszkjuk **255.255.255.0**.

A 2.1. ábra a **quark** címének, a **149.76.12.4**-nek eltérő értelmezését szemlélteti, ha hagyományos B hálózati osztályként vagy ha alhálózatként használjuk.



2.1. ábra. B osztályú hálózat alhálózattal

Nem árt megjegyeznünk, hogy az *alhálózat* csak a hálózat *belső felosztása*. Az alhálózatokat a hálózat tulajdonosa (vagy a rendszergazda) hozza létre. Az alhálózatokat gyakran úgy állítják fel, hogy már létező határokat tükrözzenek, legyen az fizikai (két Ethernet között), adminisztratív (két tanszék között) vagy földrajzi határ (két helyszín között), és az egyes alhálózatok irányításával egy kapcsolattartó személyt bíznak meg. E szerkezet azonban csak a hálózat belső viselkedésére van hatással, a külvilág számára teljesen láthatatlan.

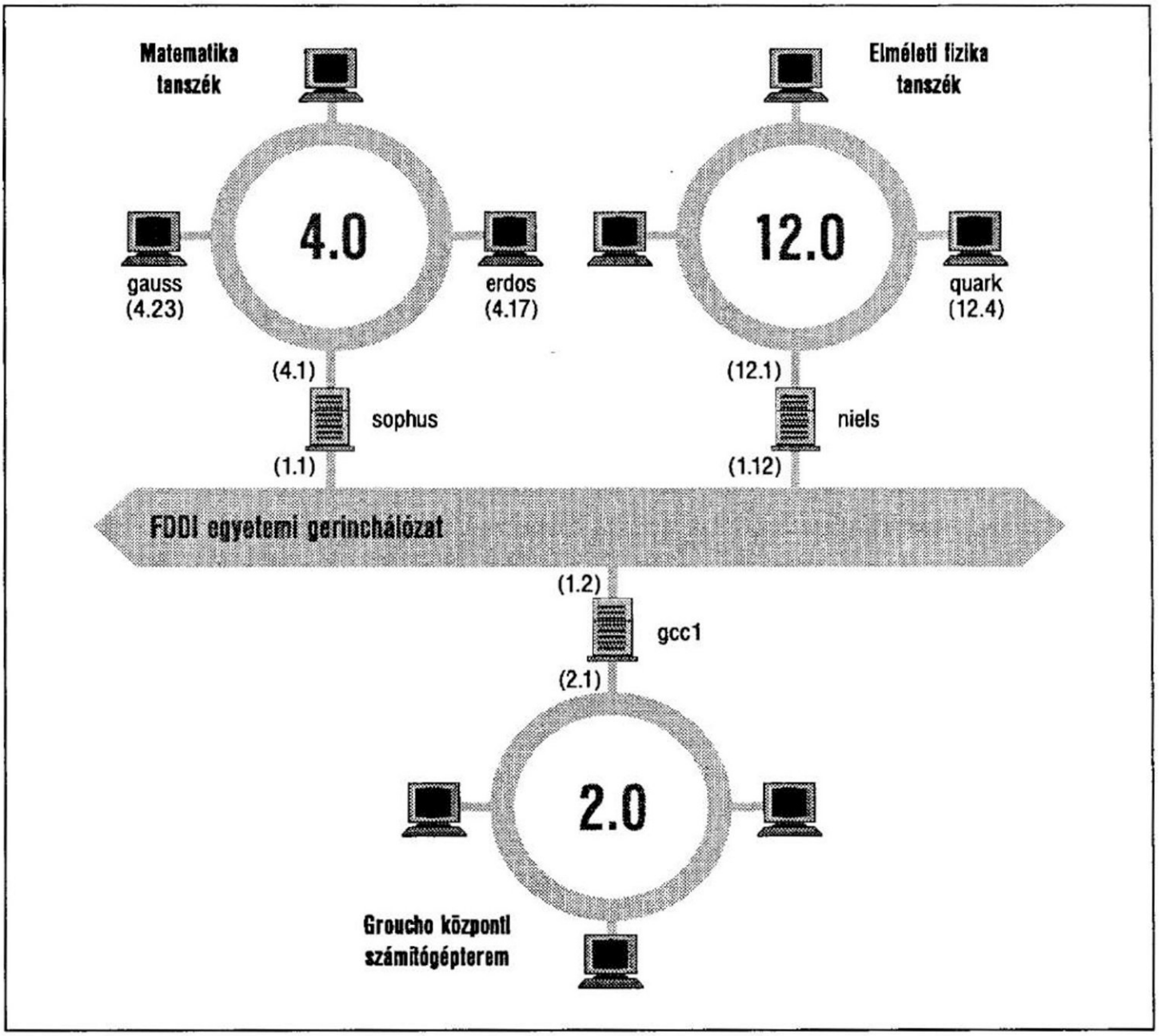
Átjárók

Az alhálózat nem csak jobb szervezési lehetőséget biztosít, de gyakran a hardverek jelentette határok természetes következménye. Egy adott fizikai hálózat, például egy

Ethernet valamely gazdagépének nézőpontja roppant korlátozott: csupán azokkal a gazdagépekkel képes érintkezni, amelyek vele azonos hálózaton vannak. Minden más gazdagépet csak különleges célú számítógépeken, *átjárókon* keresztül tud elérni. Az átjáró olyan gazdagép, amely egyidejűleg egy vagy több fizikai hálózatra csatlakozik, és feladata a csomagok továbbítása e hálózatok között.

A 2.2. ábra a GME hálózati tipológiájának részletét mutatja. Az egyidejűleg két alhálózatra is kapcsolódó gazdagépeket a két cím jelzi.

A különböző fizikai hálózatoknak különböző IP hálózatokhoz kell tartozniuk, hogy az IP felismerhesse, ha egy gazdagép a helyi hálózaton van. Így például a 149.76.4.0 hálózatszámot a Matematika tanszék lokális hálózata számára tartják fenn. Amikor datagramot küldünk a **quark** gépre, az **erdos** hálózati szoftvere a 149.76.12.4 IP címből azonnal látja, hogy a címzett kiszolgáló más fizikai hálózaton van, ezért csak egy átjárón keresztül érhető el (alapértelmezésben ez a **sophus**).



2.2. ábra. A Groucho Marx Egyetem hálózati topológiájának részlete

A **sophus** maga két, egymástól független alhálózatra kapcsolódik: a Matematika tanszékre és az egyetemi gerinchálózatra. A kettőt két különböző interfészen keresztül éri el, ezek az *eth0,* illetve az *fddi0.* Akkor milyen IP címet jelöljünk ki a számára? A cím a 149.76.1.0 vagy a 149.76.4.0 alhálózaton legyen?

A válasz: mindkettőn. A 149.76.1.0 hálózaton a **sophus** címe 149.76.1.1, a 149.76.4.0 hálózaton pedig a 149.76.4.1. Egy átjáróhoz minden hálózaton, amelyre kapcsolódik, ki kell jelölnünk egy IP címet. E címek – a megfelelő hálózatmaszkkal együtt – az alhálózathoz kapcsolódó interfészhez tartoznak. A **sophus** interfész- és címleképzését a 2.3. táblázat szemlélteti.

2.3. táblázat. Példa az interfészre és a címre

Interfész	*Cím*	*Hálózatmaszk*
eth0	149.76.4.1	255.255.255.0
fddi0	149.76.1.1	255.255.255.0
lo	127.0.0.1	255.0.0.0

Az utolsó sorban a *lo* (loopback) visszacsatoló interfészt láthatjuk, melyről a fejezet korábbi részében esett szó.

Általában nincs lényeges különbség aközött, hogy egy címet egy gazdagéphez vagy annak interfészéhez rendelünk. Az **erdos**hoz hasonlóan egyetlen hálózatra kapcsolódó gazdagépek esetén azt mondjuk, hogy a gép IP címe ez és ez, noha szigorúan véve ez az Ethernet interfész IP címe. A különbségtétel csak akkor válik lényegessé, amikor átjárókra hivatkozunk.

Az útvonalválasztó táblázat

Nézzük meg, hogyan választ az IP átjárót, amikor egy datagramot kíván továbbítani egy távoli hálózatra.

Korábban azt láttuk, hogy amikor az **erdos** a **quark**hoz címzett datagramot megkapja, ellenőrzi annak rendeltetési címét és felismeri, hogy az nem része a helyi hálózatnak. Ekkor a datagramot az alapértelmezett **sophus** átjáróhoz küldi, amely most hasonló kérdéssel néz szembe: a **quark** olyan hálózaton van, amelyhez a **sophus** nem kapcsolódik közvetlenül, ezért továbbításához egy másik átjárót kell találnia. A megfelelő választás a **niels** volna, a Fizika tanszék átjárója. Ahhoz, hogy a **sophus** a rendeltetési hálózatot egy megfelelő átjáróval társíthassa, információra van szüksége.

A feladat megoldására az IP egy, a hálózatok és a hozzájuk vezető átjárók összerendelését tartalmazó táblázatot vesz igénybe. Általában egy minden célra megfelelő bejegyzést *(alapértelmezett útvonalat)* is biztosítanunk kell; ez a 0.0.0.0 hálózattal társított átjáró. Ennek az útvonalnak minden rendeltetési cím megfelel, hiszen a 32 bit egyikének sem kell egyeznie, ezért az ismeretlen hálózatokra címzett csomagok az alapértelmezett útvonalon haladhatnak tovább. A **sophus** útvonalválasztó táblázata a 2.4. táblázathoz hasonló lehet.

2.4. táblázat. Minta útvonalválasztó táblázat

Hálózat	*Hálózatmaszk*	*Átjáró*	*Interfész*
149.76.1.0	255.255.255.0	-	*eth1*
149.76.2.0	255.255.255.0	149.76.1.2	*eth1*
149.76.3.0	255.255.255.0	149.76.1.3	*eth1*
149.76.4.0	255.255.255.0	-	*eth0*
149.76.5.0	255.255.255.0	149.76.1.5	*eth1*
0.0.0.0	0.0.0.0	149.76.1.2	eth1

Amikor olyan hálózathoz választunk útvonalat, amelyre a **sophus** közvetlenül kapcsolódik, nincs szükségünk átjáróra; a táblázat átjáró oszlopa ekkor kötőjelet tartalmaz.

A szükséges információt az útvonalválasztó táblázatból a *route* parancs és a -n opció segítségével kaphatjuk meg. A parancs az IP címekkel tér vissza, nem a DNS nevekkel.

Egy matematikai művelettel ellenőrizhető, hogy adott rendeltetési cím megfelel-e egy útvonalnak. A folyamat egészen egyszerű, azonban ismernünk kell hozzá a bináris aritmetika és logika szabályait: egy útvonal akkor felel meg egy rendeltetési címnek, ha a hálózatcímen és a hálózatmaszkon végzett logikai ÉS művelet eredménye pontosan megegyezik a rendeltetési címen és a hálózatmaszkon elvégzett logikai ÉS művelet eredményével.

Magyarul: egy útvonal megfelelő, ha a hálózatcímnek a hálózatmaszkban meghatározott számú bitje (kezdve a bal oldali bittel, a cím első bájtjának legmagasabb értékű bitjével) megegyezik a rendeltetési cím azonos számú bitjével.

Amikor az IP implementáció egy rendeltetési helyhez vezető legjobb útvonalat keresi, számos, a rendeltetési címnek megfelelő útvonalbejegyzést találhat. Tudjuk például, hogy az alapértelmezett útvonal minden rendeltetési címnek megfelel, a helyi hálózatokra címzett datagramok azonban a helyi útvonalnak is megfelelnek. Hogyan dönti el az IP, melyik útvonalat válassza? A döntésben a hálózatmaszk játszik fontos szerepet. Noha a rendeltetési címnek mindkét útvonal megfelel, valamelyik útvonal hálózatmaszkja nagyobb, mint a másik. Korábban említettük, hogy a hálózatmaszkot a címtér kisebb hálózatokra történő feldarabolására használjuk. Minél nagyobb a hálózatmaszk, annál pontosabban egyezik a rendeltetési cím; a datagramok útválasztásánál mindig a legnagyobb hálózatmaszkkal rendelkező útvonalat kell választanunk. Az alapértelmezett útvonal hálózatmaszkja nulla bitekből áll, az előbb vázolt konfigurációban pedig a helyi kapcsolt hálózatok hálózatmaszkja 24 bites. Ha a datagram megfelel egy helyi hálózatnak, nem az alapértelmezett útvonalra, hanem a megfelelő eszközre kerül, hiszen a helyi hálózat útvonala több bitben megegyezik. Az alapértelmezett útvonalra csak azok a datagramok jutnak, amelyek más útvonalnak nem felelnek meg.

Útvonalválasztó táblázatokat többféleképpen is készíthetünk. Kisebb lokális hálózatok esetén általában érdemes manuálisan összeállítanunk, majd rendszerindításkor a route paranccsal az IP számára átadnunk (lásd a 4. fejezetet). Nagyobb hálózatokon futásidejű útválasztó daemonok hozzák létre és tartják karban; a daemonok a hálózat központi kiszolgálóján futnak, és a taghálózatok közötti „optimális" útvonalak kiszámításához szükséges útvonal-információt információcsere révén szerzik be.

A hálózat méretétől függően különböző útválasztó protokollokra lehet szükségünk. Autonóm rendszereken belüli útválasztáshoz (például a Groucho Marx Egyetemen) *belső útválasztású protokollokat* futtatunk. Közülük is kiemelkedik a *Routing Information Protocol* (RIP), melyet a BSD routed daemon valósít meg. Autonóm rendszerek közötti útválasztásra *külső útválasztó protokollokat* használunk; ilyenek például az *External Gateway Protocol* (EGP) vagy a *Border Gateway Protocol* (BGP). E protokollokat, köztük a RIP-et is a Cornell Egyetem gated daemonában érhetjük el.

A metric értékek

A megcímzett gazdagéphez vagy hálózathoz vezető legjobb útvonal kiválasztásában az *ugrások* (hop) számán alapuló dinamikus útválasztásra támaszkodunk. Az ugrások a datagramnak a kiszolgálóhoz vagy a hálózathoz vezető útján érintett átjárók számát jelzik – ennyi átjárón halad keresztül a datagram. Minél rövidebb az útvonal, a RIP annál jobbnak értékeli. A 16 vagy több ugrást számláló útvonalakat használhatatlannak tekinti és elveti a rendszer.

Noha lokális hálózatunk belső útvonal-információját a RIP kezeli, a *gated* programnak minden gazdagépen futnia kell. Rendszerindításkor a gated az összes aktív hálózati interfészt megkeresi. Ha egy gazdagépen több aktív interfészt is talál (a visszacsatoló interfészt leszámítva), feltételezi, hogy a gazdagép több hálózat között kapcsol csomagokat, ennek megfelelően aktívan cseréli és közvetíti az útválasztási információt. Ellenkező esetben a gazdagép csak passzív fogadója a RIP frissítéseknek, és csak a helyi útvonalválasztó táblázatot kezeli.

Amikor a helyi útvonalválasztó táblázatból közvetít információt, a *gated* az útvonalválasztó táblázat bejegyzéséhez kapcsolódó úgynevezett *metric értékből* számítja ki az útvonal hosszát. A metric értékét a rendszergazda állítja be az útvonal konfigurálásakor; az érték szerepe, hogy kifejezze az útvonal használatának valós költségeit.* Ezért a kiszolgálóval közvetlen összeköttetésben álló alhálózatokhoz vezető útvonal metric értékét nullában, a két átjárót érintő útvonal metric értékét pedig kettőben érdemes meghatározni. Ha nem használjuk a RIP-et vagy a gated programot, a metric értékekkel nem szükséges foglalkoznunk.

Az internet vezérlőüzenet protokoll (ICMP)

Az IP egy társprotokollal is rendelkezik, amiről még nem esett szó. Ez az *internet vezérlőüzenet protokoll* (Internet Control Message Protocol, ICMP), amelyet a rendszermag hálózatkezelő része arra használ, hogy hibaüzeneteket juttasson el más gazdagépekhez. Tegyük fel például, hogy ismét az **erdos** előtt ülünk és telnetezni szeretnénk a **quark** 12345 portjára, de a porton nincs figyelő folyamat. Amikor a portra irányított első TCP csomag megérkezik a **quark**ra, a hálózatkezelő réteg észleli az érkezést, és ICMP üzenetben azonnal tájékoztatja az **erdost**, hogy a port nem elérhető.

* Egyszerű hálózaton a cél eléréséhez szükséges ugrások számát is tekinthetjük az útvonalhasználat költségének (vagy költségarányos szorzószámnak), összetett hálózatokban azonban a pontos költségek kiszámítása már-már művészet.

Az ICMP különféle üzeneteket hordoz, melyek közül sok a hibakörülményeket írja le. Van azonban egy roppant érdekes üzenet, a Redirect (átirányítás). Ezt az útválasztó modul akkor generálja, amikor észleli, hogy egy másik gazdagép átjárónak használja, holott sokkal rövidebb útvonal is létezik. Tegyük fel például, hogy a rendszerindítást követően a **sophus** útvonalválasztó táblázata hiányos, csak a Matematika tanszék hálózatához, a FDDI gerinchálózathoz és a Groucho központi számítógéptermének átjárójához (**gcc1**) tartó alapértelmezett útvonalakat tartalmazza. Ez esetben a **quark** számára küldött csomagok a **gcc1**-re érkeznek a Fizika tanszék átjárója, a **niels** helyett. Az ilyen datagram fogadásakor a **gcc1** észleli, hogy az útválasztás bizony nem szerencsés, és a csomagot a **niels** átjáróra továbbítja, miközben ICMP Redirect üzenetben informálja a **sophust** a rövidebb útvonalról.

Az is eszünkbe juthatna, hogy akkor elegendő csupán a legalapvetőbb útvonalakat manuálisan beállítanunk. Vigyázzunk azonban, mert a dinamikus útválasztásra támaszkodni – legyen az RIP vagy ICMP Redirect üzenet – nem mindig jó választás. Az ICMP Redirect és a RIP egyáltalán nem vagy csak alig nyújt lehetőséget, hogy ellenőrizzük az útválasztási információ hitelességét. Ilyen körülmények között rosszindulatú semmirekellők a teljes hálózati forgalmat megszakíthatnák, vagy akár rosszabb dolgokat is művelhetnének. Következésképpen a Linux hálózatkezelő programja a hálózati útválasztásra vonatkozó (Network) Redirect üzenetet nem általános érvénnyel, hanem egyedi módon kezeli, mintha csak az adott gazdagép átirányítására vonatkozna (Host Redirect). Ezzel minimalizálni lehet a támadásokból származó kárt, mivel csak egyetlen kiszolgálót érint, és nem a teljes hálózatot. A helyzet fonákja, hogy szabályszerű körülmények között is kissé nagyobb forgalom keletkezik, mert minden gazdagép esetén egy ICMP Redirect üzenetet kell generálnunk. Jelenleg már általánosan elfogadott, hogy igen rossz üzlet bármit is az ICMP átirányításra bízni.

A gazdagépnevek meghatározása

A fejezet korábbi részében már volt szó arról, hogy a TCP/IP hálózatokon a címzés – legalábbis az IP 4-es verziója esetén – a 32 bites számok körül forog. Nehéz volna azonban ezek közül akár csak egy párat is fejben tartanunk. Ezért a gazdagépekre általában „hétköznapi" nevekkel hivatkozunk, például **gauss** vagy **strange** néven. Az alkalmazás feladata, hogy a nevekhez tartozó IP címeket kiderítse. Ezt a folyamatot nevezzük a *gazdagépnév meghatározásának* (hostname resolution). Amikor egy alkalmazás egy adott kiszolgáló IP címét meg kívánja szerezni, a *gethostbyname(3)* és *gethostbyaddr(3)* könyvtárfüggvényekre támaszkodhat. Ezeket és más kapcsolódó eljárásokat hagyományosan önálló könyvtárban, a *feloldó könyvtárban* (resolver library) gyűjtik össze; a Linuxon mindegyik a szabványos *libc* része. Köznyelven a függvénygyűjteményt egyszerűen „feloldóként" is emlegetik. A feloldó könyvtár beállításával az 5. fejezetben foglalkozunk.

Kisebb hálózaton, például egy Ethernet hálózaton vagy Ethernetek csoportján nem okoz különösebb nehézséget a gazdagépneveket címekre leképező táblázatok fenntartása. Ezt az információt rendszerint az */etc/hosts* állomány tárolja. Gazdagépek hozzáadásakor vagy eltávolításakor, illetve a címek újrakiosztásakor csupán a gazdagépek *hosts* állományát szükséges frissítenünk. Nyilvánvaló, hogy olyan hálózaton, amely egy maroknyi gépnél többet számlál, a feladat már nehézkessé válik.

A címekkel kapcsolatos információt eredetileg az interneten is egyetlen *HOSTS.TXT* adatbázis tárolta. Az állomány a NIC kezelésében állt, és minden kapcsolódó hálózati helyre le kellett tölteni, majd telepíteni. A hálózat növekedésével a rendszerben egyre több gond mutatkozott. A *HOSTS.TXT* állomány rendszeres telepítésével együtt járó adminisztratív többletmunkától eltekintve az állományt elosztó kiszolgálók is túlterheltté váltak. A helyzetet tovább súlyosbította, hogy a NIC-nek minden nevet regisztrálnia kellett, nehogy többször is kiadják ugyanazt.

Ez az oka, hogy 1994-ben új névmeghatározó rendszert fogadtak el, a *Domain Name Systemet.* A Paul Mockapetris tervezte DNS rendszer a két problémára egyidejűleg kínált megoldást. A Domain Name System az 5. fejezet témája.

3

A soros hardver beállítása

Az internet hihetetlen sebességgel növekszik. A növekedés jórészt azoknak az internetfelhasználóknak tulajdonítható, akik olcsó és könnyen kezelhető DSL, kábel vagy más, gyors és permanens hálózati hozzáféréssel rendelkeznek, illetve akik napi e-mail- és híradagjukat a PPP és más hasonló protokollokon keresztül, az internetszolgáltatójukat feltárcsázva szerzik be.

A fejezet célja, hogy segítséget nyújtson a külvilággal modemen keresztül kapcsolatot tartó felhasználóknak. Nem foglalkozunk azonban a modemek beállításával, mert erről bőségesen találunk irodalmat a weben fellelhető számos, modemekről szóló HOWTO leírásokban. Foglalkozunk viszont a soros portot használó eszközök kezelésének legtöbb Linux vonatkozású kérdésével. Szólunk a soros kommunikációs szoftverekről, a soros eszközállományok létrehozásáról, a soros hardverekről és a soros eszközök beállításáról a *setserial* és *stty* programokkal. David Lawyer Serial HOWTO leírásában sok egyéb kapcsolódó témakörről is olvashatunk.

A modemkapcsolatok kommunikációs szoftverei

A Linuxhoz számos kommunikációs csomag elérhető. Sok közülük *terminálprogram,* vagyis a segítségével a felhasználó úgy hívhat fel egy másik számítógépet, mintha csak egyszerű terminál előtt ülne. A Unix-szerű környezetek hagyományos terminálprogramja a **kermit,** amely azonban őskövületnek számít, használata nehézkes. Ma már kényelmesebb megoldásokat is találunk, amelyek támogatják a telefonkönyveket, a szkriptnyelveket – ezekkel automatizálhatjuk a távoli számítógépek hívását és a bejelentkezést –, vagy a különféle kapcsolási protokollokat. Ilyen program az egyik legnépszerűbb DOS terminálprogram alapján készült **minicom.** Az X11 felhasználók sem csalódhatnak: a **seyon** teljes tulajdonságkészletű, X11 alapú kommunikációs program.

A terminálprogramokkal azonban még nem ér véget a soros kommunikációra alkalmas szoftverek sora. Léteznek programok, amelyekkel a kiszolgálónkra kapcsolódva elektronikus leveleinket egyetlen kötegben tölthetjük le, hogy később a nekünk megfelelő időben elolvassuk és válaszoljunk rájuk. Ezzel időt takarítunk meg, ami különösen fontos, ha szerencsétlenségünkre a kapcsolatunkat a hálózaton eltöltött idő alapján számlázzák. A levelek olvasására és a válaszokra nem a hálózaton töltött időből ál-

dozunk, hanem csak akkor kapcsolódunk ismét a hálózatra, amikor elkészültünk, és válaszainkat egyetlen kötegben töltjük fel.

A PPP köztes protokoll, mert lehetővé teszi az interaktív és a nem interaktív felhasználást is. Sokan arra használják, hogy egyetemi hálózatukat vagy más internetszolgáltatót feltárcsázva kapcsolódjanak az internetre. A PPP-t (pontosabban a PPPoE-t) azonban állandó vagy félig állandó kapcsolatokon is alkalmazzák, például kábel vagy DSL modemeken. A PPPoE a 7. fejezet témája.

Bevezetés a soros eszközök ismeretébe

A Unix rendszermagjában a soros eszközök elérésére szolgáló eszközöket tipikusan *tty* eszközöknek nevezzük (betűnként ejtjük, tehát T-T-I).

Ez a *Teletype eszköz* rövidítése – a Unix korai időszakában a Teletype volt a terminál eszközök egyik legjelentősebb gyártója. A kifejezést ma a karakter alapú adatterminálokra alkalmazzuk. A fejezetben kizárólag a Linux eszközállományokat értjük rajta, nem a fizikai terminált.

A Linux rendszereken a tty eszközöknek három osztálya van: a soros eszközök, a virtuális terminálok (melyeket a helyi konzol billentyűzetéről az Alt-F1 – Alt-F *nn* gombok lenyomásával érhetünk el) és a pszeudoterminálok (melyek az X11 és a hasonló alkalmazások kétutas adatcsatornájára emlékeztetnek). Az előbbieket azért nevezték tty eszközöknek, mert az eredeti karakter alapú terminálok soros kábelen vagy telefonvonalon és modemen keresztül csatlakoztak a Unix számítógéphez. A két utóbbi azért lett tty eszköz, mert úgy alakították ki őket, hogy a programozás szempontjából hasonló módon viselkedjenek.

A PPP-t általában a rendszermagban valósítják meg. A rendszermag a tty eszközt valójában nem hálózati eszköznek tekinti, amit az Ethernet eszközöknél használt *ifconfig* típusú parancsokkal lehetne manipulálni. Ehelyett olyan felületként kezeli, amelyhez hálózati eszközök kapcsolhatók. Ezért a rendszermag megváltoztatja a tty eszköz úgynevezett *vonali üzemmódját* (line discipline). Eszerint a PPP is vonali üzemmódnak minősül, amelyet a tty eszközökön bekapcsolhatunk. Az elképzelés lényege, hogy a kapott adatokat a soros illesztőprogram különbözőképpen dolgozza fel a beállított vonali üzemmódtól függően. Az alapértelmezett vonali üzemmódban az illesztőprogram egyszerűen továbbítja az egymás után beérkező karaktereket. Ha a rendszer PPP vonali üzemmódot jelöl ki, az illesztőprogram adatblokkot olvas be, különleges fejlécbe burkolja – ennek segítségével képes a távoli végpont az adatblokkok azonosítására az adatfolyamban –, majd az új adatblokkot küldi tovább. Ne aggódjunk, ha még nem világos minden; a következő fejezetben részletesen foglalkozunk a PPP-vel, de az egész folyamat egyébként is automatikusan zajlik.

A soros eszközök elérése

Mint a Unix rendszer minden eszközét, a soros eszközöket is a */dev* könyvtárban található speciális eszközállományokon keresztül érhetjük el. A soros illesztőprogramokhoz két különböző eszközállomány tartozik, és minden port rendelkezik a két típussal. Az eszköz viselkedése némileg eltérő attól függően, hogy melyik eszközállományt nyitjuk

meg. Érdemes a különbségeket megvizsgálnunk, mert így könnyebben megérthetjük az egyes beállításokat és azokat a tanácsokat, amelyeket a soros eszközökről kapunk, noha a gyakorlatban elegendő a két típus egyike. A jövőben a másik talán teljesen el is tűnik.

A soros eszközök két osztálya közül a 4-es elsődleges eszközazonosító számú (főszámú) a jelentősebb, amelynek eszközállományait *ttyS0, ttyS1* stb. eszközállománynak hívjuk. A másik változat főszáma 5, és akkor használjuk, amikor porton keresztül tárcsázunk; az eszközállományok neve *cua0, cua1* stb. A Unix világában a számozás általában nullával kezdődik, noha a laikusok hajlamosak eggyel kezdeni. Ez sokakat összezavar, mert a `COM1:` megfelelője a */dev/ttyS0,* a `COM2:` megfelelője a */dev/ttyS1* stb. Az IBM PC-vel kompatibilis rendszerek ismerői tudják, hogy a `COM3:` és a nála magasabb számozásokat amúgy sem szabványosították soha.

A *cua* vagy „kimenő hívás" (callout) eszközöket azért hozták létre, hogy elkerülhető legyen a soros eszközök ütközése a bejövő és kimenő kapcsolatokat is támogató modemek esetében. Sajnos a cua eszközöknek is megvannak a maguk hibái, ezért valószínűleg eltűnnek majd. Nézzük meg röviden a problémát.

A Linux a Unixhoz hasonlóan lehetővé teszi, hogy valamely eszközt vagy bármely állományt több folyamat egyidejűleg megnyisson. Sajnálatos módon a tty eszközök esetében ez csak ritkán kecsegtet sikerrel, minthogy a két folyamat szinte bizonyosan összeütközésbe kerül egymással. Szerencsére a mechanizmust úgy tervezték, hogy egy folyamat mindig ellenőrizheti, használja-e már egy másik folyamat az adott tty eszközt. Ehhez úgynevezett *zárolási állományokat* (lock files) vesz igénybe. Lényegük, hogy mielőtt egy folyamat megnyitna egy tty eszközt, előbb egy erre a célra fenntartott könyvtárban ellenőrzi, van-e olyan állomány, amely a megnyitni kívánt eszköz nevét viseli. Ha az állomány nem létezik, létrehozza a folyamat, majd megnyitja a tty eszközt. Ha létezik az állomány, a folyamat feltételezi, hogy az eszközt egy másik folyamat már használja, és ennek megfelelően fut tovább. A zárolásállomány-rendszer működtetéséhez még egy trükkre van szükség, nevezetesen, hogy a zárolási állományt létrehozó folyamat beírja az állományba saját azonosítóját, a folyamatazonosítót (pid), amiről később még szólunk.

A zárolási állományok tökéletesen működnek, feltéve hogy az állományok meghatározott helyen vannak és minden program tudja, hol keresse azokat. Nos, a Linuxon nem mindig volt így. Amíg a Linux Filesystem Standard nem definiálta a tty zárolási állományok szabványos helyét, sok hiba adódott. Volt idő, amikor a szoftverfejlesztők négy különböző könyvtárt is használtak e célra: */usr/spool/locks/, /var/spool/locks/, /var/lock/* és */usr/lock/*. Az egységesség hiánya fejetlenséghez vezetett. A programok a különböző helyeken egyre-másra nyitották meg a zárolási állományokat, amelyek elméletileg egyetlen tty eszköz szabályozására szolgáltak; az egész olyan volt, mintha az állományok nem is léteztek volna.

A cua eszközöket azért tervezték, hogy a hibát elhárítsák. Ahelyett, hogy a soros eszközre várakozó programok ütközésének elkerülésében a zárolási állományokra támaszkodnának, a fejlesztők úgy vélték, hogy a rendszermag egyszerűbben eldöntheti, ki kapjon hozzáférést. Ha a ttyS eszköz már nyitva van, a cua megnyitására tett kísérlet hibát eredményez, ami a program számára azt jelenti, hogy az eszközt már használják. Ha a cua eszköz már nyitva van, és kísérlet történik a ttyS megnyitására, a kérést a rendszer blokkolja, vagyis „várólistára" teszi, amíg a másik folyamat be nem zárja a cua eszközt. Mindez remekül működik, ha csak egyetlen, a bejövő hívásokra konfigurált modemünk van, és csak ritkán fordul elő, hogy ugyanazon az eszközön kimenő hívást is bonyolí-

tunk. Megbízhatatlan azonban az olyan rendszereken, ahol egyazon eszközön több program is kimenő hívást kezdeményezhet. A pörlekedést csak a zárolási állományok oldhatják meg! Vissza az első mezőre.

Itt elegendő megemlíteni, hogy a megoldást végül is a Linux Filesystem Standard hozta meg, amely ma már előírja, hogy a zárolási állományokat a */var/lock* könyvtárban kell tárolni, és hogy elfogadott elnevezésük például a ttyS1 esetében a *LCK..ttyS1*. A cua zárolási állományokat szintén e könyvtárban kell elhelyezni, a cua eszközök használata azonban ma már nem javasolt.

A cua eszközök ennek ellenére egy ideig kompatibilitási okokból még velünk maradnak, idővel azonban eltűnnek majd. Ha kétségeink vannak, mit használjunk, érdemes a ttyS eszközhöz ragaszkodnunk és Linux FSSTND kompatibilis rendszert futtatnunk, de legalábbis meggyőződnünk arról, hogy a zárolási állományok helyének meghatározásában a soros eszközöket használó programok megegyeznek. A soros tty eszközökkel dolgozó legtöbb szoftver fordításkor lehetőséget kínál az állományok helyének meghatározására. Ezt legtöbbször a *Makefile* állomány, illetve a konfigurációs fejlécállomány `LOCKDIR` változójában adhatjuk meg. Ha magunk fordítjuk a szoftvert, érdemes az FSSTND-ben meghatározott helyet beállítanunk. Ha előre fordított bináris állománnyal dolgozunk és nem vagyunk biztosak benne, hova írja a program a zárolási állományokat, a következő paranccsal tájékozódhatunk:

```
strings bináris állomány | grep lock
```

Ha a talált könyvtár eltér a rendszerünkben használttól, az idegen program zárolási állományait szimbolikus linkkel irányíthatjuk át a */var/lock/* könyvtárba. Nem szép megoldás, de működik.

A soros eszközök speciális eszközállományai

A másodlagos eszközazonosító számok (alszám) mindkét típusú soros eszköznél azonosak. Ha modemünk a `COM1:` – `COM4:` portok valamelyikén van, alszámát úgy kapjuk meg, hogy a COM port számához hozzáadunk 63-at. Ha különleges soros hardvert, például nagy teljesítményű többportos soros vezérlőt alkalmazunk, szükségünk lehet speciális eszközállományok létrehozására; a hagyományos eszközillesztők valószínűleg nem működnek megfelelően. A *Serial HOWTO* dokumentációban megtaláljuk a részleteket.

Tegyük fel, hogy modemünk a `COM2:` porton van. Alszáma ekkor 65, főszáma normál üzemben 4. Léteznie kell tehát egy ttyS1 eszköznek azonos számokkal. Listázzuk ki a */dev/* könyvtár soros ttys eszközeit. A fő- és alszámokat az 5., illetve a 6. oszlopban látjuk:

```
$ ls -l /dev/ttyS*

0 crw-rw---   1 uucp     dialout    4,  64 Oct 13  1997 /dev/ttyS0
0 crw-rw---   1 uucp     dialout    4,  65 Jan 26 21:55 /dev/ttyS1
0 crw-rw---   1 uucp     dialout    4,  66 Oct 13  1997 /dev/ttyS2
0 crw-rw---   1 uucp     dialout    4,  67 Oct 13  1997 /dev/ttyS3
```

Ha nem találunk 4-es főszámú és 65-ös alszámú eszközt, nekünk kell azt létrehoznunk. Rendszergazdai jogosultságokkal gépeljük be a következőket:

```
# mknod -m 666 /dev/ttyS1 c 4 65
# chown uucp.dialout /dev/ttyS1
```

A különböző Linux terjesztések némileg eltérően közelítik meg a soros eszközök tulajdonlásának kérdését. Esetenként a root a tulajdonos, máskor más felhasználó a birtokos. A terjesztések többsége külön csoportot tart fenn a kimenő hívást bonyolító eszközök számára, amelybe minden felhasználó bekerül, aki használhatja az eszközöket.

Egyesek azt javasolják, hogy a */dev/modem* legyen a modem eszközre mutató szimbolikus link, hogy az alkalmi felhasználónak ne kelljen fejben tartania a kissé nehézkes *ttyS1* elnevezést. Nem lehet azonban az egyik programban a *modem* szót, a másikban pedig az eszközállomány valódi nevét megadni, mert a zárolási állományok neve eltérő lenne és a zárolási mechanizmus nem működne.

A soros hardver

A PC világban a soros kommunikáció legelterjedtebb szabványa jelenleg az RS-232, amely számos áramkörrel biztosítja az önálló bitek átvitelét és a szinkronizálást. További vonalakkal megoldható a modemek által használt vivő (carrier) jelenlétének jelzése és a kézfogás (handshake) is. A Linux széles körben támogatja az RS-232 szabványon alapuló soros kártyákat.

A hardverkézfogás opcionális, de nagyon hasznos. Lehetővé teszi, hogy két állomás kölcsönösen jelezze egymásnak, mikor állnak készen a további adatok fogadására, illetve ha időt kérnek a bejövő adatok feldolgozásához. Az e célt szolgáló vonalak neve *küldésre kész* (Clear to Send, CTS), illetve *küldés kérése* (Request to Send, RTS), ami magyarázatot ad a hardverkézfogás közkeletű nevére is: RTS/CTS.

Ismerős lehet a kézfogás másik típusa is, a XON/XOFF handshake, amely két kijelölt karakterrel, hagyományosan a Ctrl-S és Ctrl-Q karakterekkel jelzi a távoli végpontnak, hogy indítsa el, illetve szüneteltesse az adatátvitelt. Noha az eljárás egyszerűen megvalósítható és a „buta" terminálokhoz alkalmazható, zavart okozhat a bináris adatok átvitelekor, mert előfordulhat, hogy az adatfolyam részeként e karaktereket is szeretnénk elküldeni, és nem akarjuk, hogy átviteli vezérlőkarakterként értelmeződjenek. Emellett a kézfogás sebessége is elmarad kissé a hardverkézfogástól. A hardverkézfogás egyszerű és gyors – ha van választási lehetőségünk, részesítsük előnyben a XON/XOFF kézfogással szemben.

Az eredeti IBM PC számítógépben az RS-232 interfészt egy UART lapka vezérelte, a 8250-es. A 486-os processzor korának személyi számítógépeiben az UART újabb változatát, a 16450-est találjuk, mely a 8250-esnél jóval gyorsabb volt. Szinte az összes Pentium számítógépbe ennél is újabb UART lapkát szereltek, a 16550-est. Egyes márkák (különösen a Rockwell lapkakészletet tartalmazó belső modemek) teljesen eltérő lapkát használnak, amely a 16550-es viselkedését utánozza és ahhoz hasonlóan kezelhető. A Linux szabványos soros portos illesztőprogramja mindegyiket támogatja.*

* A WinModemTM más lapra tartozik! A WinModemek roppant egyszerű hardvere a valódi munka elvégzésére számítógépünk központi processzorát veszi igénybe a dedikált hardver helyett. Javasoljuk, hogy ilyen modemet ~~ne~~ vásároljunk; szerezzünk be valódi modemet. De van remény akkor is, ha nincs más választásunk: a *http://linmodems.org* címen illesztőprogramokat, némi útbaigazítást és *LINMODEM HOWTO* leírást találunk.

A 16550-es jelentős javulást hozott a 8250-eshez és a 16450-eshez képest, mert 16 bájtos FIFO puffert tartalmazott. A 16550-es tulajdonképpen az UART eszközök egyik családja, amely magában foglalja a 16550-eset, a 16550A-sat és a 16550AFN-eset (ezt később PC16550DN-re nevezték át). A különbségek abból adódnak, hogy a FIFO működik-e; a 16550AFN az egyetlen, amelyiken biztosan működik. Létezett egy NS16550 jelölésű is, ennek FIFO puffere azonban szintén nem működött.

A 8250 és a 16450 UART lapkák egyszerű, 1 bájtos puffert rejtenek. Ez azt jelenti, hogy a 16450 lapka minden egyes átvitt vagy fogadott karakter után megszakításkérést generál. Minden megszakításkérés kiszolgálása időt vesz igénybe; a késlekedés miatt a 16450 megbízható maximális bitsebessége korlátozott egy tipikus ISA buszos rendszerben, értéke körülbelül 9600 bps.

Az alapértelmezett konfiguráció szerint a rendszermag ellenőrzi a négy szabványos soros portot, a `COM1:`-től a `COM4:`-ig. Arra is képes, hogy automatikusan felismerje az egyes soros portokhoz használt UART lapkákat, és ha elérhető, a 16550-es lapka kibővített puffertárát is igénybe veszi.

A konfigurációs segédprogramok használata

Töltsünk el most egy kis időt a soros eszközök beállítására használt két leghasznosabb eszköz, a *setserial* és az *stty* társaságában!

A setserial parancs

A rendszermag bizonyára minden tőle telhetőt megtesz, hogy helyesen állapítsa meg a soros hardver beállításait, de a soros eszközök beállítási lehetőségei annyira sokfélék, hogy a gyakorlatban ez nagyon nehéz feladat. Arra, hogy hol okozhat ez gondot, jó példát mutatnak a korábban tárgyalt belső modemek. Ezek UART lapkájának FIFO puffere 16 bájtos, de a rendszermag eszközillesztője 16450-es UART lapkának ismeri fel: ha nem mondjuk meg pontosan az illesztőnek, hogy a port egy 16550-es eszköz, a rendszermag nem használja a kiterjesztett puffert. További példa lehet az egyszerű 4 portos kártya, amely engedélyezi egyetlen IRQ megosztását több soros eszköz között. Előfordulhat, hogy nekünk kell a rendszermagnak megmondanunk, melyik IRQ portot használja, és hogy az IRQ-n esetleg többen is osztoznak.

A *setserial* parancsot arra hozták létre, hogy segítségével futás közben állítsuk be a soros illesztőprogramot. Egyes terjesztések a programot a rendszer indításakor futtatják az *rc.serial* szkriptből, noha ebben lehetnek eltérések. A szkript feladata a soros illesztőprogram inicializálása, mégpedig olyan módon, hogy illeszkedjen a rendszer nem szabványos vagy nem megszokott soros hardvereihez is.

A *setserial* általános szintaxisa a következő:

```
setserial eszköz [paraméterek]
```

Az eszköz valamely soros eszköz, például a ttyS0.
A *setserial* számos paramétert fogadhat. A leggyakoribbakat a 3.1. táblázat foglalja összse. A további paraméterekről a *setserial* súgóoldalán olvashatunk.

3.1. táblázat. A setserial parancssori paraméterei

Paraméter	*Leírás*
`port` *portszám*	A soros eszköz I/O portjának címét határozza meg. A portszámot hexadecimális alakban adjuk meg, például `0xf8`.
`irq` *szám*	A soros eszköz megszakításkérési vonalát határozza meg.
`uart` *uart_típus*	A soros eszköz UART típusát határozza meg. Szokásos értékei a `16450`, `16550` stb. A `none` értéket megadva kikapcsoljuk a soros eszközt.
`fourport`	A paraméter tudatja a kernel soros illesztőprogramjával, hogy a port egy AST Fourport kártya egyik portja.
`spd_hi`	Az UART beállítására szolgál: ha egy processz 38,4 kbps-t igényel, a sebesség 57,6 kbps lesz.
`spd_vhi`	Az UART beállítására szolgál: ha egy processz 38,4 kbps-t igényel, a sebesség 115 kbps lesz.
`spd_normal`	Az UART beállítására szolgál: a sebesség a kért 38,4 kbps alapértelmezett sebesség lesz. A paraméterrel a soros eszköznek kiadott `spd_hi` és `spd_vhi` paraméterek hatását állíthatjuk vissza.
`auto_irq`	A paraméter utasítja a rendszermagot, hogy automatikusan próbálja meghatározni a parancsban megadott eszköz IRQ-ját. A meghatározás nem mindig megbízható; egy kicsit olyan, mintha a rendszermag találgatna. Ha ismerjük az eszköz IRQ-ját, helyesebb, ha megadjuk az `irq` paraméterrel.
`autoconfig`	A paramétert csak a `port` paraméterrel együtt adhatjuk meg. Ilyenkor a `setserial` utasítja a rendszermagot, hogy automatikusan próbálja meghatározni a parancsban megadott portcímen található UART típusát. Ha az `auto_irq` paramétert is megadjuk, a rendszermag az IRQ automatikus meghatározására is kísérletet tesz.
`skip_test`	A paraméter arra utasítja a rendszermagot, hogy ne végezze el az UART típusának tesztelését az automatikus beállítás során. Erre akkor van szükség, ha a rendszermag helytelenül ismeri fel az UART típusát.

A rendszerindításkor a soros port beállítására szolgáló tipikus és egyszerű *rc* állományt mutat be a 3.1. példa. A legtöbb Linux terjesztésben ennél valamelyest kifinomultabb állományt találunk.

3.1. példa. Az rc.serial állomány setserial parancsai

```
# /etc/rc.serial - a soros vonal konfigurációs szkriptje.
#
# A soros eszköz beállítása
/sbin/setserial /dev/ttyS0 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS1 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS2 auto_irq skip_test autoconfig
/sbin/setserial /dev/ttyS3 auto_irq skip_test autoconfig
```

3.1. példa. Az rc.serial állomány setserial parancsai (folytatás)

```
#
# A soros eszköz beállításainak megjelenítése
/sbin/setserial -bg /dev/ttyS*
```

Az utolsó parancs `-bg /dev/ttyS*` argumentuma arra szolgál, hogy az összes aktív soros eszköz hardverbeállítását szépen formázott összefoglalóban jelenítse meg. Kimenete a 3.2. példához hasonló.

3.2. példa. A setserial -bg /dev/ttyS parancs kimenete

```
/dev/ttyS0 at 0x03f8 (irq = 4) is a 16550A
/dev/ttyS1 at 0x02f8 (irq = 3) is a 16550A
```

Az stty parancs

Az *stty* elnevezés valószínűleg a „set tty", vagyis a tty beállítása kifejezésből származik, noha arra is alkalmas, hogy egy terminál konfigurációját megjelenítse. A beállítható jellemzők számát tekintve az *stty* még a *setserial* parancson is túltesz. A legfontosabbakkal hamarosan megismerkedünk, a maradék leírását pedig megtaláljuk az *stty* súgóoldalán.

Az *stty* paranccsal elsősorban a terminálparamétereket állíthatjuk be, például a karakterek visszhangozását, vagy hogy milyen karakter generáljon break jelet. Szó volt róla, hogy a soros eszközök tty eszközök, ezért az *stty* parancs ugyanúgy alkalmazható rájuk is.

A soros eszközökkel kapcsolatban az *stty* egyik fontos feladata a hardverkézfogás bekapcsolása az eszközön. A fejezetben a kérdését már érintettük. A soros eszközökön a hardverkézfogás az alapértelmezett beállítás szerint kikapcsolt állapotban van. E beállítás teszi lehetővé, hogy a „háromeres" soros kábelek működjenek; ezek a kézfogáshoz szükséges jeleket nem támogatják, így ha az be volna kapcsolva, nem tudnánk a kikapcsolásához szükséges karaktert elküldeni.

Meglepő módon egyes soros kommunikációs programok nem kapcsolják be a kézfogást, ezért ha modemünk támogatja, nekünk kell erre utasítani (a szükséges parancsot megtaláljuk a modem kézikönyvében), illetve a soros eszközt is nekünk kell beállítanunk. A kézfogást egy eszközön az *stty* parancs `crtscts` jelzőbitjével kapcsolhatjuk be, ezt fogjuk használni. A parancsot érdemes a rendszerindításkor az *rc.serial* (vagy azzal egyenértékű) állományból kiadni; ezt mutatja be a 3.3. példa.

3.3. példa. Az rc.serial állomány stty parancsai

```
#
stty crtscts < /dev/ttyS0
stty crtscts < /dev/ttyS1
stty crtscts < /dev/ttyS2
stty crtscts < /dev/ttyS3
#
```

Az *stty* parancs alapértelmezés szerint az aktuális terminálra vonatkozik, de a héj bemenetátirányítási (<) lehetőségét kihasználva bármely tty eszközt manipulálhatjuk vele. Mivel sokan összekeverték a < és a > jeleket, az *stty* parancs újabb változataiban a szintaxis jóval világosabb. A 3.4. példában az új szintaxis szerint átírt minta konfigurációt láthatjuk.

3.4. példa. Az rc.serial állomány stty parancsai az új szintaxissal

```
#
stty crtscts -F /dev/ttyS0
stty crtscts -F /dev/ttyS1
stty crtscts -F /dev/ttyS2
stty crtscts -F /dev/ttyS3
#
```

Említettük, hogy az *stty* utasítással bármely tty eszköz terminálparamétereit megjeleníthetjük. Adott tty eszköz összes aktív beállítását a következő utasítással írathatjuk ki:

```
$ stty -a -F /dev/ttyS1
```

A parancs kimenete, melyet a 3.5. példa tartalmaz, megadja az eszköz összes jelzőbitjének állapotát; ha a jelzőbit előtt számítástechnikai mínuszjel van (például `-crtscts`), a jelzőbitet kikapcsoljuk.

3.5. példa. Az stty -a parancs kimenete

```
speed 19200 baud; rows 0; columns 0; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
        eol2 = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R;
        werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
-parenb -parodd cs8 hupcl -cstopb cread clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr -icrnl -ixon
        -ixoff -iuclc -ixany -imaxbel
-opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0
        bs0 vt0 ff0
-isig -icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop
        echoprt echoctl echoke
```

A legfontosabb jelzőbitek leírását a 3.2. táblázat foglalja össze. A jelzőbiteket úgy kapcsolhatjuk be, hogy az *stty* parancs után írjuk azokat, és úgy kapcsolhatjuk ki, hogy mínuszjelet teszünk elébük. Ezért a ttys0 eszköz hardverkézfogás opciójának kikapcsolásához a következő parancsot adnánk ki:

```
$ stty -crtscts -F /dev/ttyS0
```

3.2. táblázat. A soros eszközök beállításánál leggyakrabban használt stty jelzőbitek

Jelzőbit	*Leírás*
N	A vonal sebességét N bit/másodpercre állítja.
`crtscts`	A hardverkézfogás be- és kikapcsolása.
`ixon`	A XON/XOFF vezérlés be- és kikapcsolása.
`clocal`	A modem vezérlőjeleinek, például a DTR/DTS és a DCD jeleknek a be- és kikapcsolása. „Háromeres" soros kábelnél szükséges, mert az ilyen kábel nem támogatja ezeket a jeleket.
`cs5 cs6 cs7 cs8`	Az adatbitek számát 5, 6, 7, illetve 8 bitre állítja.
`parodd`	Páratlan paritást állít be. A jelzőbit kikapcsolásával páros paritást állíthatunk be.
`parenb`	A paritásbit ellenőrzése. Ha a jelzőbit elé mínuszjelet teszünk, nem használunk paritásbitet.
`cstopb`	Bekapcsolja a két stopbit használatát karakterenként. Ha a jelzőbit elé mínuszjelet teszünk, karakterenként csak egy stopbitet használunk.
`echo`	A fogadott karakter visszhangozásának (visszaküldésének a küldőhöz) be- és kikapcsolása.

A következő példában több jelzőbit kombinálásával a ttyS0 eszközt a következő értékekre állítjuk: 19200 bps, 8 adatbit, nincs paritásbit, bekapcsolt hardverkézfogás visszhangozás nélkül.

```
$ stty 19200 cs8 -parenb crtscts -echo -F /dev/ttyS0
```

Soros eszközök és a bejelentkezés: a parancskérő jel

Valaha elterjedt gyakorlat volt, hogy a Unix-telepítés tartalmazott egy szervert és számos „buta", karakteres üzemmódban működő terminált vagy betárcsázó modemet. Ma ez már ritkábban fordul elő, ami jó hír azok számára, akik ilyen rendszert szeretnének üzemeltetni, mert olcsón hozzájuthatnak a terminálokhoz. A betárcsázó modemes konfigurációk ma is elterjedtek, azonban az egyszerű bejelentkezés helyett inkább a PPP bejelentkezést támogatják (lásd a 6. fejezetet). Ettől eltekintve mindkét konfiguráció egyaránt hasznát veheti egy egyszerű szoftvernek, a *getty* programnak.

A *getty* valószínűleg a „get tty", a tty lekérése kifejezés rövidítése. A program megnyit egy soros eszközt, megfelelően konfigurálja – esetleg a modemmel együtt –, majd várakozik a kapcsolat létrejöttére. A soros eszközök aktív kapcsolatát általában a hívott soros eszköz *Data Carrier Detect* (DCD) tűje jelzi. A kapcsolat észlelésekor a *getty* egy `login:` parancskérő jelet hoz létre, majd a rendszerbe történő beléptetés elvégzésére meghívja a bejelentkezést kezelő programot. A Linux rendszereken minden egyes virtuális terminálhoz (például */dev/tty1)* tartozik egy *getty* program.

A *getty* több különböző változatban létezik; mindegyiket úgy tervezték, hogy bizonyos konfigurációkhoz jobban megfelelnek, mint másokhoz. Mi az *mgetty* programmal foglalkozunk. A meglehetősen népszerű program sikerét annak köszönheti, hogy sok olyan lehetőséget kínál – például automatikus faxprogramok és hangmodemek támogatását –, amelyek különösen alkalmassá teszik a modemek kezelésére. Elsősorban arra fordítjuk a figyelmünket, hogyan állíthatjuk be az *mgetty* programot a hagyományos adathívások fogadására, a további lehetőségeket pedig ki-ki maga derítheti fel.

Az mgetty daemon beállítása

Az *mgetty* daemon szinte minden Linux terjesztésben megtalálható előre csomagolt formában. A daemon a legtöbb *getty* változattól különbözik, mert kimondottan az AT parancskészletet támogató modemekhez készült.

Ennek ellenére támogatja a közvetlen terminálkapcsolatokat, leginkább mégis a betárcsázó alkalmazásokhoz megfelelő. A bejövő hívások érzékelésére nem a DCD vonalat használja, hanem a `RING` üzenetet figyeli, amelyet a korszerű modemek a bejövő hívás észlelésekor generálnak, ha nincsenek automatikus válaszadásra állítva.

A fő futtatható állomány neve */usr/sbin/mgetty,* a fő konfigurációs állomány neve */etc/mgetty/mgetty.config.* Vannak más bináris és konfigurációs állományai is, ezek az *mgetty* egyéb tulajdonságaiért felelnek.

A legtöbb rendszeren a beállításhoz elegendő, ha módosítjuk az */etc/mgetty/mgetty.config* állományt, és elhelyezzük az *mgetty* automatikus futtatásához szükséges bejegyzéseket az */etc/inittab* állományban.

A 3.6. példában roppant egyszerű *mgetty* konfigurációs állományt ismertetünk. A példa két soros eszközt állít be. Az első, a */dev/ttyS0* egy Hayes-kompatibilis modemet támogat 38 400 bps sebességen. A második, a */dev/ttyS1* egy közvetlenül kapcsolt VT100 terminált támogat 19 200 bps sebességen.

3.6. példa. Minta /etc/mgetty/mgetty.config állomány

```
#
# mgetty konfigurációs állomány
#
# minta konfigurációs állomány; részletek az mgetty.info állományban
#
# a megjegyzést "#" karakter nyitja meg, az üres sorok figyelmen kívül
maradnak
#
# —- globális szakasz —-
#
# Idehelyezzük a globális érvényű alapértelmezett értékeket
#
# a modem(ek) sebessége legyen 38400 bps
speed 38400
#
# a hibakeresési szint legyen "4" (a policy.h alapértelmezett értéke)
debug 4
#
# —- az önálló portokra vonatkozó szakasz —-
```

3.6. példa. Minta /etc/mgetty/mgetty.config állomány

```
#
# Itt helyezzük el az adott vonal beállításait - ezek más vonalakra nem
érvényesek
#
#
# a ttyS0-ra kapcsolt Hayes modem: nincs fax, korlátozott naplózás
#
port ttyS0
  debug 3
  data-only y
#
# közvetlen kapcsolatú VT100 terminál; a DTR megszakítás tiltása
#
port ttyS1
  direct y
  speed 19200
  toggle-dtr n
#
```

A konfigurációs állományban globális, illetve csak adott portra érvényes opciókat is megadhatunk. Példánkban a 38 400 bps sebesség beállítására globális opciót használtunk. A ttyS0 port ezt az értéket örökli, mint ahogy az összes *mgetty* programmal konfigurált port is ezt a sebességet használja, ha az egyes portok esetében külön nem írjuk felül az értéket, ahogy a ttyS1 beállításánál tettük.

A `debug` kulcsszó az *mgetty* naplóinak részletességét szabályozza. A ttyS0 konfigurációjában látható `data-only` kulcsszó hatására az *mgetty* a modem összes faxtulajdonságát figyelmen kívül hagyja, és egyszerű adatmodemként viselkedik. A ttyS1 konfigurációjának `direct` kulcsszava utasítja az mgetty programot, hogy ne kísérelje meg a modem inicializálását a porton. És végül a `toggle-dtr` kulcsszó arra utasítja a programot, hogy ne szakítsa meg a vonalat a soros interfész *Data Terminal Ready* (DTR) tűjének megszakításával, mert egyes terminálok ezt rosszul viselik.

Ha nekünk úgy tetszik, az *mgetty.cofig* állományt üresen is hagyhatjuk, és a legtöbb paramétert parancssori argumentumként is megadhatjuk. Az alkalmazás dokumentációjában részletes leírást találunk az *mgetty* konfigurációs állományának paramétereiről és a parancssori argumentumokról. Lásd a következő példát.

A beállítás aktiválásához két bejegyzést helyezünk el az */etc/inittab* állományban. Az *inittab* a Unix System V *init* parancsának konfigurációs állománya, amely a rendszer inicializálásáért felel, és lehetőséget ad arra, hogy a rendszer indításakor programokat futtassunk, illetve ismételten végrehajtsuk azokat, amikor futásuk véget ér. A *getty* futtatásához pontosan erre van szükségünk.

```
T0:23:respawn:/sbin/mgetty ttyS0
T1:23:respawn:/sbin/mgetty ttyS1
```

Az */etc/inittab* állomány minden sorában négy mezőt látunk, melyeket kettőspont választ el. Az első mező egy azonosító, amely az állományon belül egyedi címkével ruházza föl a bejegyzést; ez hagyományosan két karakter, bár az új változatok négy karaktert

engedélyeznek. A második mező azon futási szintek listája, ahol a bejegyzésnek aktívnak kell lennie. A futási szintekkel alternatív számítógép-konfigurációkat biztosíthatunk; ehhez rendszerint faszerkezetbe rendezett indítási szkripteket használunk. A szkriptek könyvtárakban helyezkednek el, például */etc/rc1.d*, /etc/rc2.d stb. néven. Használatuk nagyon egyszerű, bejegyzéseinket elkészíthetjük az állományban található más bejegyzések vagy a rendszer dokumentációja alapján. A harmadik mező azt írja le, hogy a művelet mikor hajtódik végre. A *getty* futtatásához ebben a mezőben a `respawn` értéket adjuk meg, ami azt jelenti, hogy a parancsot automatikusan újra kell futtatni, ha leáll. Más opciók is léteznek, számunkra azonban ezek most lényegtelenek. A negyedik mező maga a végrehajtani kívánt parancs: itt adjuk meg az *mgetty* programot és argumentumait. Példánkban egyszerűen elindítjuk és szükség esetén újraindítjuk a programot, valahányszor a rendszer kettes vagy hármas futási szinten üzemel, és argumentumként csupán a használni kívánt eszköz nevét adjuk át. Az *mgetty* az eszközt automatikusan a */dev/* könyvtárban keresi, ezért nem kell megadnunk.

A fejezetben röviden bemutattuk az *mgetty* programot és hogy miként biztosíthatunk bejelentkezési promptot a soros eszközök számára. A *Serial HOWTO* átfogó információt közöl a témáról.

A konfigurációs állományok szerkesztése után az init programot ismét be kell töltenünk ahhoz, hogy a módosítások érvénybe lépjenek. Ehhez egyszerűen szakítsuk meg az init folyamatot; az init folyamatazonosítója mindig 1, így biztonsággal kiadhatjuk a következő parancsot:

```
# kill -HUP 1
```

4

A TCP/IP hálózatkezelés beállítása

A fejezetben végigjárjuk a TCP/IP hálózatkezelés beállításának lépéseit az IP címek kiosztásától a TCP/IP hálózati interfészek beállításáig, miközben megismerünk néhány olyan eszközt, amelyek kapóra jönnek a hálózat telepítési hibáinak felderítésében.

A fejezet legtöbb feladatát általában csak egyszer szükséges elvégeznünk. Ezt követően a konfigurációs állományokhoz akkor kell újból hozzányúlnunk, ha hálózatunkra új rendszert kapcsolunk, vagy azt teljesen átkonfiguráljuk. A TCP/IP beállítását szolgáló néhány parancsnak azonban a rendszer minden indulásakor le kell futnia, ezért általában az */etc/rc* rendszerszkriptekből hívjuk meg ezeket.

Az eljárás hálózatra vonatkozó részét többnyire egyetlen szkript tartalmazza, amelynek neve a különböző Linux terjesztésekben eltérő lehet. Sok régebbi Linux kiadásban ez az *rc.net* vagy *rc.inet.* Előfordulhat, hogy két szkripttel is találkozunk: az *rc.inet1* és *rc.inet2* szkriptekkel; az előbbi a hálózatkezelés rendszermaghoz tartozó részét inicializálja, az utóbbi az alapvető hálózatkezelő szolgáltatásokat, alkalmazásokat indítja el. Az újabb terjesztésekben az *rc* állományok felépítése kifinomultabb elrendezést követ: az */etc/init.d/* (vagy */etc/rc.d/init.d/)* könyvtárban bukkanhatunk a hálózati eszközöket létrehozó szkriptekre, illetve a hálózati alkalmazásokat futtató egyéb *rc* állományokra. A könyv példái az utóbbi elrendezést követik.

A fejezetben a hálózati interfészeket beállító szkript egyes részeit taglaljuk valamivel bővebben. A fejezet végére olyan parancskészlet birtokába jutunk, melynek segítségével beállíthatjuk számítógépünk TCP/IP hálózatkezelését. Ezután már csak a minta konfigurációs szkriptek parancsait kell a sajátjainkra kicserélnünk, majd – meggyőződve róla, hogy a szkriptet a fő *rc* szkript a rendszer indításakor meghívja – újraindíthatjuk a számítógépet. A kedvenc Linux terjesztésünkben található hálózatkezelő *rc* szkript példái megfelelő alapul szolgálnak a munkához.

A /proc fájlrendszer értelmezése

A Linux 2.4 disztribúciók a rendszermaggal a */proc* fájlrendszeren keresztül tartanak kapcsolatot, míg a 2.6 verzió az új sysfs interfészt használja. Mindkét interfész egy fájlrendszerre emlékeztető mechanizmussal teszi lehetővé a rendszermag futásidejű adatainak lekérdezését. A fejezetben a */proc* fájlrendszerrel ismerkedünk meg, minthogy jelenleg ez az elterjedtebb. A felcsatolás (mount) után a */proc* fájlrendszer a többi fájlrendszerhez hasonlóan állományokat jelenít meg tartalmukkal együtt. Tipikus állományok pél-

dául a *loadavg*, amely a rendszer átlagos terhelését mutatja, és a *meminfo*, amely az aktuális központi memória és a csereállomány (swap) felhasználásáról tájékoztat.

A hálózatkezelő program ehhez hozzáteszi még a *net* könyvtárt, amely különböző állományokat tartalmaz. Ezekből megismerhetjük például a rendszermag ARP tábláit, a TCP kapcsolatok állapotát és az útvonalválasztó táblázatokat. A hálózati adminisztrációs alkalmazások többsége ezekre az állományokra támaszkodik.

A *proc* fájlrendszer (vagy más néven *procfs)* felcsatolása általában a rendszer indításakor a */proc* csatolási pontra történik. A legjobb megoldás, ha a következő sort elhelyezzük az */etc/fstab* állományban:

```
# procfs mount point:
none /proc proc defaults
```

Ezután futtassuk a `mount /proc` utasítást az */etc/rc* szkriptből.
Ma már a legtöbb rendszermagban alapértelmezésként beállítják a *procfs-t.*

Az eszközök telepítése

Az előre összeállított Linux terjesztések tartalmazzák a jelentősebb hálózati alkalmazásokat és segédprogramokat, valamint magukban foglalnak egy összefüggő mintaállomány-készletet is. Csak akkor szükséges új segédprogramokat beszereznünk, ha a rendszermag új verzióját telepítjük. Ezekben néha megváltoztatják a hálózatkezelő réteget, így az alapvető konfigurációs eszközök is frissítésre szorulnak. A frissítés legalábbis újrafordítást jelent, de esetenként a legújabb bináris állományokat is be kell szereznünk. A bináris állományokat a hivatalos webhelyről, az *ftp://ftp.inka.de/pub/comp/Linux/networking/NetTools/* címről tölthetjük le archív állomány formájában, melynek neve *net-tools-XXX.tar.gz*, ahol az *XXX* a verziószámot jelenti.

Ha a szabványos TCP/IP hálózati alkalmazásokat magunk szeretnénk fordítani és telepíteni, a forrásokat a legtöbb Linux FTP szerveren megtaláljuk. A korszerű Linux terjesztések szép számmal tartalmaznak TCP/IP hálózati alkalmazásokat, például internetes böngészőket, Telnet és FTP programokat, valamint más hálózati alkalmazásokat, például a talk programot. Ha olyasmire bukkanunk, amit nekünk kell fordítanunk, a Linux rendszereken ezt valószínűleg minden nehézség nélkül megtehetjük a forrásból, az ott mellékelt instrukciókat követve.

A gazdagépnév beállítása

A legtöbb – ha nem az összes – hálózati alkalmazás ránk bízza, hogy a helyi gazdagép nevét valamilyen ésszerű értékre állítsuk. A beállítást általában a rendszer indításakor végezzük a hostname paranccsal. Ha a gazdagépnevet a *név* értékre szeretnénk állítani, írjuk be:

```
# hostname név
```

Általános gyakorlat, hogy a minősítetlen gazdagépnevet adjuk meg a tartománynév nélkül. Ha van például egy hálózati helyünk, melynek neve Virtuális Sörgyár (egy kita-

lált, de tipikus kis hálózat, mellyel a könyv több fejezetében is találkozunk), a kiszolgáló neve lehet **vale.vbrew.com** vagy **vlager.vbrew.com**. Ezek a hivatalos, *teljes minősített tartománynevek* (fully qualified domain name, FQDN). A helyi gazdagép neve adja a név első részét, például **vale**. Mivel azonban a helyi gazdagép nevét gyakran használják a kiszolgáló IP címének kikeresésére, meg kell győződnünk róla, hogy a feloldó könyvtár megtalálja a gazdagép IP címét. Ez rendszerint azt jelenti, hogy a nevet elhelyezzük az */etc/hosts* állományban.

Ahhoz, hogy az FQDN fennmaradó részének tartománynév voltáról a rendszermagnak fogalma legyen, a beállításhoz egyesek a *domainname* parancsot javasolják. Ily módon a *hostname* és *domainname* kimenetének kombinálásával ismét az FQDN-hez jutnánk. Ez azonban a legjobb esetben is csak féligazság. A *domainname* parancsot általában a gazdagép NIS tartományának beállítására használjuk, és ez teljesen eltérhet attól a DNS tartománytól, amelyhez a gazdagép tartozik. Azért, hogy a gazdagépnév rövid formájában is feloldható legyen a *hostname* parancs minden korábbi verziójával, érdemes inkább a következő két megoldás egyikét választani. Helyezzünk el egy bejegyzést a helyi tartománynév-kiszolgálónkon. Írjuk be a teljes minősített tartománynevet az */etc/hosts* állományba. Ekkor a *hostname* parancshoz az `-fqdn` argumentumot hozzáfűzve megkapjuk a kimenetben a teljes minősített tartománynevet.

Az IP címek kiosztása

Ha gazdagépünk hálózatkezelő szoftverét önálló üzemre állítjuk, ezt a részt nyugodtan átugorhatjuk, mert csak a visszacsatoló interfész IP címére lesz szükségünk, és az mindig 127.0.0.1.

A valódi hálózatokon, például az Ethernet hálózaton a helyzet némileg összetettebb. Amennyiben gazdagépünket egy már létező hálózatra kívánjuk kapcsolni, a rendszergazdájától kell hálózati IP címet kérnünk - noha ez sem mindig igaz. Sok hálózat ma már dinamikusan osztja ki az IP címeket; az eljárás neve *Dynamic Host Configuration Protocol* (DHCP), erről a következő részben szólunk. Amikor a hálózatot mi alakítjuk ki, az IP címeket vagy magunk adjuk ki, vagy egy DHCP kiszolgálót állítunk üzembe. Ha számítógépünk közvetlenül kapcsolódik az internetre, az IP címet az ISP, DSL vagy kábelhálózati kiszolgálónktól kell kérnünk.

Egy helyi hálózat gazdagépei általában egyetlen logikai IP hálózat címein osztoznak, vagyis IP címük első oktetjei rendszerint azonosak. Ha több fizikai hálózattal rendelkezünk, akkor vagy különböző hálózatszámokat jelölünk ki számukra, vagy alhálózatokkal osztjuk fel több részre az IP címtartományt. Az alhálózatokkal még foglalkozunk a fejezet *Alhálózatok létrehozása* című részében.

Ha hálózatunk nem kapcsolódik az internetre vagy a kapcsolódáshoz hálózati címfordítást alkalmaz, bármilyen érvényes hálózatcímet választhatunk. Csak arra ügyeljünk, hogy belső hálózatunkból ne jussanak csomagok az internetre. Legyünk azért biztosítva is, hogy akkor sem történjék semmi baj, ha csomagok *mégis* kikerülnének. Ennek érdekében a személyes felhasználásra fenntartott hálózatszámok közül válasszunk egyet. Az *Internet Assigned Numbers Authority* (IANA) az A, B és C osztályokból több hálózatszámot is elkülönített, ezeket regisztráció nélkül használhatjuk. Ilyen címek csak saját hálózatunkon belül érvényesek, és a valódi internet hálózati helyei nem érhetik el.

A számokat az RFC 1918 határozza meg, és a 2. fejezet 2.1. táblázatában soroljuk fel. Vegyük észre, hogy a második és harmadik blokk 16, illetve 256 hálózatot tartalmaz.

De nemcsak az internettől teljesen elszigetelt hálózatok esetén érdemes és lehet e hálózatszámok egyikéből címet választanunk, hiszen például egy átjáróként üzemeltetett gazdagéppel létrehozhatunk korlátozott hozzáférést az internethez. Helyi hálózatunkon az átjáró a belső IP címén keresztül érhető el, míg a külvilág a hivatalosan bejegyzett cím alapján ismeri fel (a címet a szolgáltatónk jelöli ki). Az IP álcázással kapcsolatban erre még visszatérünk a 9. fejezetben.

A könyv hátralevő részében feltételezzük, hogy a Virtuális Sörgyár hálózatának rendszergazdája egy B osztályú hálózatszámot választott, ez a **172.16.0.0**. Természetesen egy C osztályú hálózatszám is megfelelne mind a sörgyár, mind a borászat hálózatához. A B osztályú hálózatot az egyszerűség kedvéért választottuk; a fejezet alhálózatokkal foglalkozó része így érthetőbbé válik.

Az IP címek kiosztása a DHCP rendszerrel

Jelenleg sok hálózat alkalmazza a *Dynamic Host Configuration Protocol* (DHCP) rendszert. A protokoll a kettes hálózati rétegen fut, és a DHCP kéréseket figyeli. A DHCP kiszolgáló rendelkezik egy előre meghatározott listával a rendszergazda által kiosztott és a felhasználóknak kiadható IP címekről. Amikor a DHCP-hez IP cím kérés érkezik, a rendszer DHCP szerződés kibocsátásával válaszol. A szerződés azt jelenti, hogy a kérelmező gazdagép megszabott időre megkapja az IP címet. Az erősen terhelt hálózatokon a szerződést gyakran meghatározott óraszámra adják ki, megakadályozva ezzel, hogy egy tétlen gép sokáig lefoglalhasson egy címet. Egyes hálózatokon ez az érték nagyon rövid, akár két óra is lehet. Kisebb hálózatokon a DHCP szerződés ideje hosszabbra is állítható, egy napra, vagy akár egy hétre. Az értéket – a hálózat terheltségét figyelembe véve – a rendszergazda határozza meg.

Egy hálózaton a DHCP szerződés kéréséhez a *dhcpcd* szoftver szükséges, melynek legújabb verzióját a *http://www.phystech.com/download/dhcpcd.html* címről tölthetjük le. A legfrissebb verziókon kívül itt megtaláljuk a leírását is. A legtöbb korszerű Linux terjesztésben a szoftvert előre telepítik, és már a rendszer kezdeti beállításakor is lehetőséget biztosít arra, hogy az interfészeket a DHCP-vel konfiguráljuk.

A címkérés a DHCP segítségével egyszerű, és a következő paranccsal végezhető:

```
vlager# dhcpcd eth0
vlager#
```

A daemon e ponton újrakonfigurálja az *eth0* interfészt, és nem csak egy IP címet rendel hozzá, de az alhálózat megfelelő beállításáról is gondoskodik. Sok DHCP kiszolgáló alapértelmezett útvonalat és DNS információt is biztosít. Ez utóbbi esetben */etc/resolv.conf* állományunkat a daemon felülírja a frissített DNS-kiszolgálóinformációval. Ha valamilyen okból ezt el kívánjuk kerülni, a parancssorban adjuk meg a `-R` kapcsolót. A *dhcpcd* további parancssori argumentumokat is fogadhat, amelyekre bizonyos feltételek mellett lehet szükség. Ezekről a *dhcpcd* súgóoldalon olvashatunk. A *resolv.conf* állományról bővebben a DNS-ről szóló fejezetben tájékozódhatunk.

A DHCP kiszolgáló futtatása

Nagyobb, dinamikusabb hálózatokon a DHCP elengedhetetlen. Működéséhez azonban az szükséges, hogy az ügyfelek az IP címeket egy DHCP kiszolgálóról kapják. Noha sok útválasztó, tűzfal és más hálózati eszköz kínál ilyen funkcionalitást, a hálózati adminisztrátornak érdemes fontolóra venni egy linuxos számítógép üzemeltetését erre a célra, mert a Linux DHCP kiszolgálók a beállítások terén általában rugalmasabbak. A DHCP kiszolgálóknak több változata is létezik. Az egyik népszerű és kedvező választás az ISC ajánlata, melyet az *ftp://ftp.isc.org/isc/dhcp/* címen érhetünk el. Beállítása és telepítése szabványos, az ismert *automake* konfigurációs szkripttel végezhető. Fordítását és telepítését követően máris konfigurálhatjuk a szoftvert.

Először azonban győződjünk meg arról, hogy hálózati interfészeinken beállítottuk a többcímes üzenetküldés (multicast) támogatását; a beállítást a legegyszerűbben az *ifconfig* paranccsal végezhetjük el:

```
ticktock root # ifconfig
eth0      Link encap:Ethernet  HWaddr C0:FF:EE:C0:FF:EE
          inet addr:172.16.1.1  Bcast:172.16.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:80272 errors:0 dropped:0 overruns:0 frame:0
          TX packets:55339 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:8522502 (8.1 Mb)  TX bytes:9203192 (8.7 Mb)
          Interrupt:10 Base address:0x4000
```

Ha a kimenetben nem jelenik meg a `MULTICAST`, újra kell konfigurálnunk a rendszermagot. Ennek valószínűsége azonban csekély, mert a legtöbb rendszermagban a multicast alapértelmezett beállítás.

Most már készen állunk a *dhcpd.conf* állomány elkészítésére. Lássunk egy minta *dhcpd.conf* állományt:

```
# Minta konfiguráció DHCP kiszolgálóhoz
option domain-name "vbrew.com";
option domain-name-servers ns1.vbrew.com, ns2.vbrew.com;
default-lease-time 1600;
max-lease-time 7200;
log-facility local7;
# Nagyon egyszerű alhálózat kijelölése:
subnet 172.16.1.0 netmask 255.255.255.0 {
  range 172.16.1.10 172.16.1.50;
  option routers router1.vbrew.com;
}
```

A konfiguráció a **172.16.1.0** hálózaton hoz létre és oszt ki címeket; összesen 40 IP címet oszthat ki a **172.16.1.10** és a **172.16.1.50** közé eső tartományban. Az option routers és domain-name-servers parancsokkal az ügyfelek részére biztosított alapértelmezett útválasztót és DNS kiszolgálókat adhatjuk meg.

A DHCP kiszolgáló beállítási opciói közül lássuk most a fontosabbakat!

option domain-name
Zárójelek között megadhatjuk hálózatunk tartománynevét. Elhagyható, de hatékonyabbá teheti a helyi nevek kikeresését.

option domain-name-servers
Noha sokan gondolják, hogy elhagyható, a legtöbb esetben meg kell adnunk. Itt soroljuk fel az IP címeket vagy az FQDN tartománynév-kiszolgálókat.

default-lease-time
Amikor egy gazdagép szerződést kér, de nem határozza meg, hogy mennyi időre, ez az érték lesz az időtartam. Másodpercben adjuk meg.

max-lease-time
A szerződés legfeljebb erre az időtartamra adható ki.

fixed-address
Az opció lehetőséget ad fix IP cím kijelölésére egy adott ügyfél számára. Általában a MAC címszűrés opcióval együtt használjuk.

hardware ethernet
Ezzel az opcióval a hálózati adminisztrátor megadhatja, mely MAC címekhez lehet IP címet kiosztani. Az opcióval biztonságossá tehetünk egy DHCP tartományt, illetve MAC címeket kapcsolhatunk IP címekhez.

A DHCP kiszolgáló az ügyfél MAC címét mint eljárást az IP címek kiosztására vagy korlátozására használhatja. Az ilyen típusú beállítás erős biztonságot igénylő környezetben lehet indokolt, ahol csak az ismert számítógépek kaphatnak címet. A következő példában azt látjuk, hogyan rendel a DHCP kiszolgáló egy adott címet egy gazdagéphez annak MAC címe alapján; vegyük észre, hogy a range direktívát is alkalmazhatnánk:

```
host vale {
hardware ethernet 0:0f:d0:ee:ag:4e;
fixed-address 172.16.1.55;
}
```

Ügyeljünk rá, hogy a DHCP címtartományokban ne használjunk statikusan kiosztott címeket – ha nem figyelünk erre, címkonfliktusokra számíthatunk.

Alhálózatok létrehozása

Több Ethernet hálózat üzemeltetéséhez hálózatunkat alhálózatokra osztjuk. Az alhálózatok azonban csak akkor kívánatosak, ha egynél több üzenetszóró (broadcast) hálózatunk van – a pont-pont kapcsolatok nem számítanak. Ha például van egy Ethernet és egy vagy több PPP kapcsolatunk a külvilághoz, nincs szükség alhálózatokra. Ezzel részletesen a 6. fejezetben foglalkozunk.

A két Ethernet hálózat (a sörgyár és a borászat) üzemeltetését latolgatva a sörgyár hálózati igazgatója elhatározta, hogy a cím gazdagépre eső részének 8 bitjét kiegészítő alhálózati bitként fogja használni. Így 8 bit jut a gazdagéprészre, vagyis minden alhálózat 254 gazdagépet tartalmazhat. Ezután az 1-es alhálózatszámot a sörgyárhoz, a 2-es alhálózatszámot a borászathoz rendelte. A kapott hálózatcímek a **172.16.1.0** és a **172.16.2.0**. Az alhálózati maszk a **255.255.255.0**.

A két hálózat közötti átjáró a **vlager**, melynek gazdagépszáma mindkét hálózaton 1, IP címei pedig ennek megfelelően **172.16.1.1** és **172.16.2.1**.

Ne feledjük, hogy a példában a dolgokat egyszerűsítve B osztályú hálózattal dolgozunk, noha egy C osztályú hálózat valószerűbb volna. Az új hálózatkezelési kódnál a bájthatárok nem korlátozzák az alhálózatokat, vagyis akár egy C osztályú hálózatot is több alhálózatra oszthatnánk. Például ha a hálózatmaszkhoz a gazdagéprész két bitjét foglaljuk le, négy lehetséges alhálózatot hozhatunk létre, egyenként 64 állomással.*

A hosts és networks állományok

Miután hálózatunkat felosztottuk alhálózatokra, az */etc/hosts* állomány segítségével fel kell készülnünk valamifajta egyszerű gazdagépnév-meghatározásra. Ha nem DNS vagy NIS címfeloldást alkalmazunk, minden gazdagépet fel kell tüntetnünk a hosts állományban.

De még ha DNS-t futtatunk is normál üzemben, az */etc/hosts* állománynak legalább a gazdagépek egy részét tartalmaznia kell, mert valamilyen névmeghatározásra akkor is szükség lehet, amikor a hálózati interfészek nem futnak, például a rendszer indításakor. Ez nem egyszerűen kényelmi kérdés, noha lehetővé teszi, hogy szimbolikus gépneveket használjunk a hálózat *rc* szkriptjében. Így az IP címek megváltozásakor elegendő, ha a frissített *hosts* állományt átmásoljuk az összes számítógépre és újraindítjuk a rendszert, ahelyett hogy egy csomó *rc* állományt külön-külön módosítanánk. Általában az összes helyi gazdagépnevet és címet elhelyezzük a *hosts* állományban, kiegészítve a használt átjárók és NIS kiszolgálók nevével.

Győződjünk meg róla, hogy feloldónk a kezdeti ellenőrzést kizárólag a *hosts* állomány alapján végzi – a DNS szoftverrel kapott mintaállományok furcsa eredményre vezethetnek. Azért, hogy minden alkalmazás csak az */etc/hosts* állományt használhassa egy gazdagép IP címének kikeresésére, módosítsuk az */etc/host.conf* állományt. Helyezzünk megjegyzésbe minden `order` kulcsszóval kezdődő sort úgy, hogy egy kettős kereszt karaktert írunk elébük, majd adjuk az állományhoz a következő sort:

```
order hosts
```

A feloldó könyvtár beállításával a 6. fejezetben ismerkedünk meg.

A *hosts* állomány minden sora egyetlen bejegyzést tartalmaz, amely egy IP címből, a gazdagép nevéből és a gazdagépnév hivatkozási neveinek (alias) opcionális listájából áll. A mezőket szóköz vagy tabulátor választja el, a cím mező az első oszlopban kezdődik.

* Minden alhálózat első száma az alhálózatszám, míg az utolsó száma egy fenntartott csoportos üzenetszórási cím, vagyis valójában 62 állomásról lehet szó alhálózatonként.

A kettős kereszt (#) után álló adatok megjegyzésnek minősülnek és figyelmen kívül maradnak.

A gazdagépnevek vagy teljes minősített nevek, vagy a helyi tartománynévhez viszonyítottak. A **vale** esetében a *hosts* állományban érdemes megadnunk a teljes minősített nevet, **vale.vbrew.com**, valamint a **vale** nevet önmagában is, hogy egyaránt ismert legyen a hivatalos és a helyben alkalmazott rövidebb név is.

Példaként lássuk most a Virtuális Sörgyár *hosts* állományát. Két különleges név szerepel benne, a **vlager-if1** és **vlager-if2**, melyek a **vlager** átjáró két interfészének címét határozzák meg.

```
#
# A Virtuális Sörgyár/Virtuális Borászat hosts állománya
#
# IP              FQDN                        alias nevek
#
127.0.0.1         localhost
#
172.16.1.1        vlager.vbrew.com            vlager vlager-if1
172.16.1.2        vstout.vbrew.com            vstout
172.16.1.3        vale.vbrew.com              vale
#
172.16.2.1        vlager-if2
172.16.2.2        vbeaujolais.vbrew.com       vbeaujolais
172.16.2.3        vbardolino.vbrew.com        vbardolino
172.16.2.4        vchianti.vbrew.com          vchianti
```

Éppúgy, mint a gazdagép IP címénél, esetenként a hálózatszámokhoz is szükség lehet szimbolikus nevekre. Ezért a *hosts* állományhoz tartozik egy */etc/networks* állomány is, amely a hálózatneveket hálózatszámokra képezi le, és fordítva. A Virtuális Sörgyár *networks* állománya például így fest:*

```
# A Virtuális Sörgyár /etc/networks állománya
brew-net      172.16.1.0
wine-net      172.16.2.0
```

Az interfészek beállítása az IP használatára

Miután a 3. fejezetben bemutatott módon összeállítottuk hardverünket, be kell azt „mutatnunk" a rendszermag hálózatkezelő szoftverének. A hálózati interfészek beállítását és az útvonalválasztó táblázat inicializálását néhány utasítással elvégezhetjük. A feladatot rendszerint a hálózat inicializációs szkriptjéből futtatjuk a rendszer indításakor. A folyamathoz használt két alapvető eszköz az *ifconfig* (ahol az „if" az interfészre utal) és a *route*.

Az *ifconfig* feladata, hogy az interfészt elérhetővé tegye a rendszermag hálózatkezelő rétege számára. Ebbe beletartozik az IP cím és más paraméterek kijelölése és az in-

* Figyelem! A *networks* állomány nevei nem egyezhetnek meg a *hosts* állomány gazdagépneveivel, mert ez egyes programokban zavart okozhat.

terfész aktiválása. Az interfész aktív állapota azt jelenti, hogy a rendszermag IP datagramokat küldhet és fogadhat rajta keresztül. Legegyszerűbben így aktiválhatunk egy interfészt:

```
ifconfig interfész ip cím
```

A parancs az *interfész*hez *rendeli az* `ip cím` értékét és aktiválja az interfészt. Minden más paraméter értéke az alapértelmezett érték. Az alapértelmezett hálózatmaszk például az IP cím hálózati osztályából származik, és egy B osztályú cím esetén **255.255.0.0**. Az *ifconfig* programról bővebben is szó lesz a fejezetben.

A *route* paranccsal útvonalat vehetünk fel vagy távolíthatunk el a rendszermag útvonalválasztó táblázatából. Meghívása így történhet:

```
route [add|del] [-net|-host] cél [if]
```

Az `add` és `del` argumentumok határozzák meg, hogy a *cél*hoz vezető útvonalat felvenni (`add`) vagy törölni (`del`) kívánjuk. A `-net` és `-host` argumentumok megmondják route parancsnak, hogy a cél egy hálózat vagy egy gazdagép. Az *if* argumentum szintén opcionális, és segítségével kijelölhetjük azt a hálózati interfészt, amelyre az útvonalat irányítani szeretnénk - ha elhagyjuk, a Linux rendszermagja dönti el, melyiket használja. A következőkben alaposabban is megismerkedünk a témával.

A visszacsatoló interfész

A legelőször aktivált interfész szinte mindig a visszacsatoló interfész:

```
# ifconfig lo 127.0.0.1
```

Az IP cím helyén találkozhatunk néha a **localhost** szimbolikus névvel is. Az *ifconfig* a *hosts* állományban keresi a nevet, ezért egy bejegyzésben megadjuk a **127.0.0.1** címet és mellette gazdagépnévként a localhost kifejezést:

```
# A localhost minta /etc/hosts bejegyzése
localhost    127.0.0.1
```

Ha szeretnénk egy interfész beállításait megtekinteni, hívjuk meg az **ifconfig** parancsot, egyedül az interfész nevét adva át argumentumként:

```
$ ifconfig lo
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          Collisions:0
```

Láthatjuk, hogy a visszacsatoló interfész hálózatmaszkja a **255.0.0.0**, mivel a **127.0.0.1** egy A osztályú hálózatcím.

Lassan megszületik minihálózatunk. Már csak az hiányzik, hogy létrehozzunk az útvonalválasztó táblázatban egy bejegyzést, amely megmondja az IP-nek, hogy ezt az interfészt útvonalként használhatja a **127.0.0.1** rendeltetési helyhez. Írjuk meg a bejegyzést:

```
# route add 127.0.0.1
```

Az IP cím helyett természetesen most is állhatna a localhost, feltéve hogy felvettük az */etc/hosts* állományba.

Eljött az ideje, hogy ellenőrizzük a munkánkat. Ehhez használhatjuk például a *ping* programot, amely a hanglokátor megfelelője a hálózatkezelésben. Az utasítással megvizsgálhatjuk, hogy egy adott cím elérhető-e, illetve mérhetjük azt is, hogy mennyi időbe telik, amíg egy datagram eljut a címzetthez, majd onnan visszaérkezik. Ezt az időtartamot gyakran „körbejárási időnek" (round-trip time) is nevezik:

```
# ping localhost
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=0.4 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=255 time=0.4 ms
^C
--- localhost ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.4/0.4 ms
#
```

Amikor a bemutatott módon meghívjuk a *ping* programot, a végtelenségig küldeni fogja a csomagokat, ha a felhasználó nem szakítja meg a futását. A ^C azt jelzi, hogy itt megnyomtuk a Ctrl-C billentyűket.

A példában azt látjuk, hogy a **127.0.0.1** címre küldött csomagok rendben megérkeznek és szinte azonnal visszatérnek a *ping*-hez, vagyis sikeresen beállítottuk első hálózati interfészünket!

Ha a *ping* kimenete nem hasonlít a példa kimenetére, bajban vagyunk. Vizsgáljuk meg a hibaüzeneteket, hátha kiderül, hogy egyik vagy másik állományt rosszul telepítettük. Ellenőrizzük, hogy a használt **ifconfig** és *route* bináris állományok kompatibilisek-e a futtatott rendszermaggal, valamint a rendszermag fordításakor bekapcsoltuk-e a hálózatkezelést (ezt a */proc/net* könyvtár meglétéből tudhatjuk). Ha a hibaüzenet arról árulkodik, hogy a hálózat nem elérhető („Network unreachable"), valószínűleg rosszul adtuk ki a **route** parancsot. Győződjünk meg róla, hogy ugyanazt a címet adtuk meg, mint az **ifconfig** parancsban.

Most már eleget tudunk ahhoz, hogy önálló gazdagépen hálózatkezelő alkalmazásokat futtassunk. Miután a korábban említett sorokat elhelyeztük a hálózat inicializáló szkriptjében és meggyőződtünk róla, hogy a sorok a rendszer indításakor végrehajtódnak, újraindíthatjuk a gépet és kipróbálhatunk számos alkalmazást. Az *ssh localhost* például *ssh* kapcsolatot hoz létre a gazdagéphez, megjelenítve az SSH bejelentkezési promptot.

A visszacsatoló interfész azonban nem csak arra jó, hogy a hálózatkezeléssel foglalkozó könyvekben példaként hozzuk fel, vagy a próbakő szerepét játssza a fejlesztés alatt, hiszen sok alkalmazás a rendes működéséhez is igényli.* Ezért mindig állítsuk be, függetlenül attól, hogy számítógépünk csatlakozik-e valamilyen hálózatra vagy sem.

Az Ethernet interfészek

Az Ethernet interfész beállítása szinte teljesen megegyezik a visszacsatoló interfész beállításával, csupán egy-két paraméterrel többet kell megadnunk, amikor alhálózatokat használunk.

A Virtuális Sörgyár az eredetileg B osztályú IP hálózatot C osztályú alhálózatokra osztotta fel. Ahhoz, hogy az interfész ezt felismerje, az *ifconfig* hívásánál így kell eljárnunk:

```
# ifconfig eth0 vstout netmask 255.255.255.0
```

A parancs az *eth0* interfészhez a **vstout (172.16.1.2)** IP címét rendeli. Ha elhagynánk a hálózatmaszkot, akkor az **ifconfig** az IP hálózatosztályból származtatná, így a hibás **255.255.0.0** hálózatmaszkot kapnánk. Egy gyors ellenőrzés:

```
# ifconfig eth0
eth0    Link encap 10Mps Ethernet HWaddr  00:00:C0:90:B3:42
        inet addr 172.16.1.2 Bcast 172.16.1.255 Mask 255.255.255.0
        UP BROADCAST RUNNING  MTU 1500  Metric 1
        RX packets 0 errors 0 dropped 0 overrun 0
        TX packets 0 errors 0 dropped 0 overrun 0
```

Látjuk, hogy az **ifconfig** automatikusan a szokásos értékre állítja be a csoportos üzenetszórási címet (a `Bcast` mező): a gazdagép hálózatszámára, melyben a gazdagép minden bitje be van állítva. Emellett a maximális átviteli egységet (a rendszermag által az interfész számára generált IP datagramok maximális méretét) az Ethernet csomagok maximális méretére, 1500 bájtra állítja. Általában az alapértelmezett értékeket használjuk, de szükség esetén speciális opciókkal minden értéket felülírhatunk; ezeket a fejezet későbbi részében ismertetjük.

Mint a visszacsatoló interfész esetében, egy útválasztó bejegyzés létrehozásával tudatjuk a rendszermaggal, hogy milyen hálózatot érhet el az *eth0* interfészen keresztül. A Virtuális Sörgyár esetén a route programot a következőképpen hívnánk meg:

```
# route add -net 172.16.1.0
```

Első pillantásra mindez misztikusnak tűnhet, hiszen nem világos, honnan tudja a route program, melyik interfészen keresztül vezesse az útvonalat. A megoldás azonban nagyon egyszerű: a rendszermag sorban ellenőrzi a már beállított interfészeket, és a rendeltetési címet (itt **172.16.1.0**) összehasonlítja az interfészcím hálózati részével (vagyis

* Például minden RPC alapú alkalmazás, amely induláskor a visszacsatoló interfészen keresztül regisztrálja magát a *portmapper* daemonnál. Ilyen alkalmazás a NIS és az NFS is.

bitenkénti ÉS műveletet végez az interfészcímen és a hálózatmaszkon). Az egyetlen megfelelő interfész az *eth0*.

És mire való a `-net` opció? Azért adjuk meg, mert a **route** a hálózatokhoz és az önálló gazdagépekhez vezető útvonalakat is képes kezelni (ahogy láthattuk a `localhost` példáján). Amikor pontozott négyes jelöléssel adunk át számára egy címet, a **route** a gazdagéprész bitjeiből megkísérli kitalálni, hogy hálózatról vagy gazdagépnévről van-e szó. Amennyiben a cím gazdagéprésze nulla, a **route** feltételezi, hogy hálózatot jelöl; ellenkező esetben egy gazdagép címeként kezeli. Így a **route** azt gondolná, hogy a **172.16.1.0** egy gazdagép címe, nem pedig hálózatszám, hiszen nem tudja, hogy alhálózattal dolgozunk. Ezért a `-net` opcióval kifejezetten meg kell mondanunk, hogy itt hálózatról van szó.

A **route** parancs begépelése azonban fárasztó, és könnyen melléüthetünk. Kényelmesebb, ha az */etc/networks* állományban definiált hálózatneveket használjuk. Így a parancs olvashatóbbá válik, és még a `-net` jelzőt is elhagyhatjuk, hiszen a **route** tudja, hogy a **172.16.1.0** hálózatot jelöl:

```
# route add brew-net
```

Elvégezve az alapvető beállításokat, szeretnénk most látni, hogy az Ethernet interfész valóban vidáman fut. Válasszunk egy kiszolgálót az Ethernetünkről, például a **vlagert**, és írjuk be:

```
# ping vlager
PING vlager: 64 byte packets
64 bytes from 172.16.1.1: icmp_seq=0. time=11. ms
64 bytes from 172.16.1.1: icmp_seq=1. time=7. ms
64 bytes from 172.16.1.1: icmp_seq=2. time=12. ms
64 bytes from 172.16.1.1: icmp_seq=3. time=3. ms
^C
—vstout.vbrew.com PING Statistics—
4 packets transmitted, 4 packets received, 0
round-trip (ms)  min/avg/max = 3/8/12
```

Ha a kimenet nem hasonlít az itt láthatóra, hiba történt. Amennyiben szokatlanul magas a csomagvesztés aránya, a hardver hibájára gyanakodhatunk, például rossz vagy hiányzó lezárókra. Ha egyáltalán nem kapunk választ, ellenőrizzük az interfész beállítását a fejezetben korábban tárgyalt *netstat* programmal. Az *ifconfig* csomagstatisztikájából megtudhatjuk, hogy egyáltalán elhagyták-e az interfészt a csomagok. Ha a távoli gazdagéphez is hozzáférünk, menjünk át oda és vizsgáljuk meg ott is a statisztikát. Ily módon pontosan kideríthetjük, hol vesztek el a csomagok. Emellett a route programmal az útválasztási információt is írassuk ki, és nézzük meg, hogy az útvonalbejegyzés mindkét gazdagépen megfelelő-e. A *route* a rendszermag teljes útvonalválasztó táblázatát kinyomtatja, ha argumentumok nélkül hívjuk meg (a `-n` csupán arra szolgál, hogy a címeket ne a gazdagépek nevével, hanem pontozott négyes felosztással jelenítse meg).

```
# route -n
Kernel routing table
```

```
Destination  Gateway   Genmask          Flags Metric   Ref Use  Iface
127.0.0.1    *         255.255.255.255  UH    1        0     112 lo
172.16.1.0   *         255.255.255.0    U     1        0      10 eth0
```

A mezők részletes jelentéséről később még szó lesz. A `Flags` oszlop az egyes interfészekhez beállított kapcsolókat szedi listába. Az `U` az aktív interfészek esetén mindig bekapcsolt állapotban van, a `H` pedig azt jelenti, hogy a rendeltetési cím egy gazdagép címe. Ha a `H` kapcsoló olyan útvonalhoz van beállítva, amelyet hálózati útvonalnak szántunk, ismételten adjuk ki a route parancsot a `-net` opcióval. Azt, hogy a megadott útvonal egyáltalán használatban van-e, úgy vizsgálhatjuk, hogy megnézzük, növekszik-e a `Use` mező (jobbról a második) értéke a ping két meghívása között.

Útválasztás átjárókon keresztül

Az előző részben csupán egyetlen Ethernet hálózat egyetlen gazdagépének beállításával foglalkoztunk. Meglehetősen gyakran találkozunk azonban átjárókon keresztül összekapcsolt hálózatokkal. Az átjárók esetleg csak két vagy alig néhány Ethernet hálózatot kötnek össze, de a külvilághoz, például az internethez is biztosíthatnak kapcsolatot. Az átjáró alkalmazásához a hálózatkezelő rétegnek kiegészítő információra van szüksége az útvonallal kapcsolatban.

A Virtuális Sörgyár és a Virtuális Borászat Ethernet hálózatait pontosan ilyen átjáró köti össze, a **vlager**. Feltételezve, hogy a **vlager** beállítását már elvégeztük, csupán egyetlen bejegyzést kell a **vstout** útvonalválasztó táblázatához adnunk, hogy a rendszermagot tájékoztassuk: a borászat hálózatának kiszolgálóit a **vlager**en keresztül érheti el. A **route** megfelelő hívását itt láthatjuk; a `gw` kulcsszó arra utal, hogy az utána következő argumentum egy átjáró (gateway):

```
# route add wine-net gw vlager
```

Természetesen ahhoz, hogy a borászat hálózatán lévő minden gazdagéppel kommunikálhassunk, valamennyinek rendelkeznie kell egy útvonalbejegyzéssel a sörgyár hálózatához. Ellenkező esetben, bár a sörgyár hálózatáról küldhetnénk adatokat a borászat hálózatára, az ottani gazdagépek nem tudnának válaszolni.

A példában egy átjárót figyeltünk meg, amely két izolált Ethernet hálózat között kapcsolt össze adatokat. Most tegyük fel, hogy a **vlager** az internethez is kapcsolódik (mondjuk egy további SLIP kapcsolaton keresztül). Azt szeretnénk, hogy a sörgyár hálózatán kívül *minden más* hálózatra küldött datagramot a **vlager** kapjon meg. Ezt úgy érhetjük el, hogy a **vstout** alapértelmezett átjárójává nevezzük ki:

```
# route add default gw vlager
```

A **default** hálózatnév a 0.0.0.0 rövidítése, és az alapértelmezett útvonalat jelöli. Az alapértelmezett útvonal minden rendeltetési helynek megfelel, és ha rajta kívül nincs más megfelelő, speciális útvonal, a rendszer ezt használja. Nevét nem szükséges az */etc/networks* állományba felvennünk, mert be van építve a **route** programba.

Amikor egy gazdagépet a ping programmal egy vagy több átjárón keresztül szondázunk és a csomagvesztés aránya magas, az erősen terhelt hálózatra utalhat. A csomagvesztés rendszerint nem technikai hiányosságnak tudható be, hanem az ideiglenesen túlterhelt továbbító gazdagépek okozzák, melyek a bejövő datagramokat késleltethetik, illetve véglegesen el is „dobhatják".

Az átjárók beállítása

Egy gazdagépet elég egyszerűen beállíthatunk, hogy két Ethernet hálózat között csomagokat kapcsoljon. Tegyük fel, hogy ismét ott ülünk a **vlager** előtt, amely két Ethernet kártyával kapcsolódik a két hálózatra. Mindössze a két interfészt kell beállítanunk, IP címet és megfelelő útvonalat rendelve hozzájuk, és már készen is vagyunk.

Érdemes a két interfészről bejegyzést készíteni a *hosts* állományban, mert így könnyen kezelhető nevekhez jutunk:

```
172.16.1.1    vlager.vbrew.com     vlager vlager-if1
172.16.2.1    vlager-if2
```

A két interfészt beállító parancsok a következők:

```
# ifconfig eth0 vlager-if1
# route add brew-net
# ifconfig eth1 vlager-if2
# route add wine-net
```

Ha a parancsok nem működnek, ellenőrizzük, hogy a rendszermagba belefordítottuk az IP továbbítást is. Az egyik lehetőségünk, hogy megbizonyosodjunk róla, ha a */proc/net/snmp* második sorában mint első számot az 1-est látjuk.

A pont-pont interfész

A két gép összekapcsolására alkalmas PLIP kapcsolat kissé eltér az Ethernettől. A PLIP kapcsolatok *pontok közötti* (point-to-point) kapcsolatok, vagyis a kapcsolat mindkét végén egy-egy gazdagép áll. Az Ethernethez hasonló hálózatok ezzel szemben úgynevezett *üzenetszóró* (broadcast) hálózatok. A pontok közötti kapcsolatok beállítása különbözik, mert az üzenetszóró hálózatoktól eltérően a pontok közötti rendszerek nem támogatnak saját hálózatot.

A PLIP olcsó és hordozható kapcsolatot biztosít a számítógépek között. Példaként vegyük a Virtuális Sörgyár egyik alkalmazottjának laptop számítógépét, amely PLIP kapcsolattal csatlakozik a **vlager** gépre. A laptop neve **vlite**, és csak egyetlen párhuzamos porttal rendelkezik. A rendszer indulásakor a portot *plip1* néven regisztráljuk. A kapcsolat aktiválásához a következő parancsokkal állítjuk be a *plip1* interfészt:*

* A **pointopoint** kifejezést – nem tévedés – valóban így írjuk!

```
# ifconfig plip1 vlite pointopoint vlager
# route add default gw vlager
```

Az első parancs az interfészt állítja be, tájékoztatva a rendszermagot, hogy pont-pont kapcsolatról van szó, ahol a távoli gép címe a **vlager**. A második az alapértelmezett útvonalat határozza meg, átjáróként a **vlager**t jelölve meg. A **vlager** gépen egy hasonló **ifconfig** utasítással aktiválhatjuk a kapcsolatot (a **route** meghívása nem szükséges):

```
# ifconfig plip1 vlager pointopoint vlite
```

Figyeljük meg, hogy a **vlager** *plip1* interfészéhez nem kell önálló IP címet rendelnünk, ennek is adhatjuk a **172.16.1.1** címet. A pontok közötti kapcsolatok nem támogatnak közvetlenül hálózatot, így az interfésznek semmilyen hálózaton nem kell címet biztosítanunk. Az esetleges hibák megelőzésére a rendszermag az útvonalválasztó táblázat interfész információját használja.* Beállítottuk tehát a laptoptól a sörgyár hálózatához vezető útvonalat; hiányzik azonban az útvonal, amelyen keresztül a sörgyár gazdagépeiről a **vlite** gépet elérhetjük. Az egyik lehetséges, de igen fárasztó megoldás, hogy minden gazdagép útvonalválasztó táblázatában elhelyezünk egy bejegyzést, a **vlager**t nevezve meg átjáróként a **vlite**-hoz:

```
# route add vlite gw vlager
```

Az ideiglenes útvonalakhoz sokkal jobb megoldás a dinamikus útválasztás. Használhatjuk például a *gated* útválasztó daemont, melyet a hálózat összes kiszolgálójára fel kell telepítünk, hogy az útvonal-információt dinamikusan kezelhesse. A legegyszerűbb megoldást azonban egy *proxy ARP* (Address Resolution Protocol) jelenti. Proxy ARP esetén a **vlager** minden **vlite**-ra irányuló kérésre saját Ethernet címét küldi el válaszul. A **vlite**-ra címzett csomagok így a **vlager**hez érkeznek, és ez továbbítja azokat a laptophoz. A proxy ARP-re még visszatérünk a fejezet *Az ARP táblák ellenőrzése* című részében.

A *net-tools* új kiadásai tartalmaznak egy *plipconfig* nevű eszközt, mellyel a PLIP egyes időzítéssel kapcsolatos paramétereit állíthatjuk be. A printer porthoz tartozó IRQ értékét az *ifconfig* paranccsal szabályozhatjuk.

A PPP interfész

Noha a PPP kapcsolatok a PLIP-hez hasonlóan két pont közötti kapcsolatok, érdemes róluk bővebben is szólni. A PPP a 6. fejezet témája lesz.

Az IP hivatkozási nevek (alias)

Van a Linuxnak olyan tulajdonsága, az alias, amely kiválthatja a régi hamis (dummy) interfészeket, és más hasznos funkciókat is támogat. Az IP alias nevek lehetővé teszik, hogy

* Óvatosságból csak akkor állítsuk be a PLIP kapcsolatot, ha az útvonalválasztó táblázat bejegyzéseit már létrehoztuk Ethernet hálózataink számára. Egyes régebbi rendszermagok esetén ugyanis előfordulhat, hogy a hálózati útvonal végül a két pont közötti kapcsolatra mutat majd.

egyetlen fizikai eszközhöz több IP címet rendeljünk. Az esetek többségében a gazdagépünket állítjuk be úgy, hogy több, saját IP címmel rendelkező gazdagépnek látsszon. Az eljárást *virtuális gépkezelésnek* (virtual hosting) is nevezik, noha műszaki szempontból egy sor más feladatra is alkalmas.*

Amikor egy interfészhez alias nevet szeretnénk megadni, először is ellenőriznünk kell, hogy a rendszermag fordítása az IP Alias támogatásával történt (ha igen, akkor létezik a */proc/net/ip_alias* állomány; ha nem, akkor újra kell fordítanunk a rendszermagot). Egy IP alias és egy valódi hálózati eszköz beállítása gyakorlatilag azonos módon történik; mindössze a különleges névvel jelezzük, hogy alias nevet hozunk létre. Például:

```
# ifconfig eth0:0 172.16.1.1
```

A parancs az *eth0* interfészhez hoz létre egy 172.16.1.1 című alias nevet. Amikor az IP aliasra hivatkozunk, a hálózati eszközhöz a *:n* kifejezést fűzzük, ahol az „n" egy egész szám. Példánkban az *eth0* hálózati eszközhöz hozunk létre egy aliast, melynek száma nulla. Ily módon egyetlen fizikai eszköz több alias nevet is támogathat.

Az alias nevet önálló eszközként kezelhetjük, és a rendszermag IP szoftvere számára valóban önálló is az eszköz; a hardvert azonban egy másik interfésszel osztja meg.

Bővebben az ifconfig utasításról

Az *ifconfig* utasítás az eddig tárgyalt paramétereknél sokkal többet képes fogadni. Rendes hívása a következő:

```
ifconfig interfész [cím [paraméterek]]
```

Az *interfész* az interfész neve, a *cím* az interfészhez kiosztott IP cím. Az utóbbi lehet egy pontozott négyes jelöléssel megadott IP cím, vagy egy név, melyet az *ifconfig* az */etc/hosts* állományból keres ki.

Amikor az *ifconfig* utasításban csupán az interfész nevét adjuk meg, a kimenetben az interfész beállításairól kapunk listát. Ha egyáltalán nem adunk meg paramétert, a kimenet a már beállított összes interfészt megjeleníti; a `-a` opcióval az inaktív interfészeket is kiírathatjuk. Nézzük meg az *eth0* Ethernet interfészhez meghívott utasítás mintakimenetét:

```
# ifconfig eth0
eth0    Link encap 10Mbps Ethernet  HWaddr 00:00:C0:90:B3:42
        inet addr 172.16.1.2 Bcast 172.16.1.255 Mask 255.255.255.0
        UP BROADCAST RUNNING  MTU 1500  Metric 0
        RX packets 3136 errors 217 dropped 7 overrun 26
        TX packets 1752 errors 25 dropped 0 overrun 0
```

* Pontosabban fogalmazva, az IP hivatkozási nevek használatát a hálózati réteg virtuális gépkezelésének is nevezik. A WWW és STMP világában gyakoribb az alkalmazási réteg virtuális gépkezelése, ahol ugyanazt az IP címet osztjuk ki az egyes virtuális gazdagépeknek, de minden alkalmazási rétegből származó kéréshez különböző gazdagépnevet társítunk.

Az MTU és Metric mezők a maximális átviteli egység aktuális méretét és az interfész metric értékét mutatják. Az operációs rendszerek hagyományosan a metric értékből számítják ki egy útvonal költségét.

Az RX és TX sorok jelzik, hogy hány csomag érkezett vagy távozott hibátlanul, hány hiba történt, hány csomagot dobott el az interfész (például mert kevés volt a memória), és hány veszett el túlcsordulás miatt. A csomagok fogadásakor általában akkor történik túlcsordulás, amikor a csomagok gyorsabban érkeznek, mint ahogy a rendszermag a megszakításkérést kiszolgálja. Az ifconfig kimenetében látható kapcsolók többé-kevésbé megfelelnek a parancssori opciók neveinek; ezekről még lesz szó.

A következő listában összefoglaljuk az *ifconfig* által értelmezett paramétereket, a nekik megfelelő kapcsolók nevével együtt. A funkciók bekapcsolására szolgáló paraméterek egyben ki is kapcsolják azokat, ha eléjük vonást (dash, számítástechnikai mínuszjel) írunk.

up
: Elérhetővé tesz egy interfészt az IP réteg számára. Ha a parancssorban a *címet* is megadjuk, azzal az opciót automatikusan bekapcsoljuk. A down opcióval ideiglenesen kikapcsolt interfészek újbóli bekapcsolására is alkalmas.
Az opció az UP és RUNNING kapcsolóknak felel meg.

down
: Elérhetetlenné tesz egy interfészt az IP réteg számára. Gyakorlatilag minden IP forgalmat leállít az interfészen. Jegyezzük meg, hogy az opció automatikusan törli az interfészt használó összes útvonalbejegyzést is.

netmask *maszk*
: Alhálózati maszkot jelöl ki az interfész számára. Megadhatjuk 32 bites hexadecimális számmal (előtte a 0x kifejezéssel), vagy decimális számokkal, pontozott négyes jelöléssel. Míg a pontozott négyes forma gyakoribb, addig a hexadecimális formával sokszor könnyebb dolgozni. A hálózatmaszkok alapvetően bináris számok, és a bináris-hexadecimális átváltás egyszerűbb, mint a bináris-decimális.

pointopoint *cím*
: Az opciót a két kiszolgálót tartalmazó, két pont közötti IP kapcsolatokhoz használjuk, például a SLIP és PLIP interfészek beállítására. Ha beállítottunk egy két pont közötti címet, az *ifconfig* megjeleníti a POINTOPOINT kapcsolót.

broadcast *cím*
: Az üzenetszórási cím általában a hálózatszámból jön létre úgy, hogy a gazdagéprész összes bitjét beállítjuk. Egyes IP implementációk (például a BSD 4.2-ből származó rendszerek) eltérő módszert alkalmaznak, és a gazdagéprész összes bitjét törlik. A broadcast opció alkalmazkodik e különleges környezetekhez. Ha az üzenetszórási címet beállítottuk, az ifconfig megjeleníti a BROADCAST kapcsolót.

`irq`
Segítségével megadhatjuk az adott eszközökkel használt IRQ vonalat. Különösen a PLIP esetén fontos, de egyes Ethernet kártyáknál is jól jöhet.

`metric` *szám*
Az opció az interfészhez tartozó útvonalválasztó táblázat bejegyzéséhez rendel egy metric értéket. A RIP a metric alapján építi fel a hálózat útvonalválasztó táblázatát.* Az *ifconfig* alapértelmezett metric értéke a nulla. Ha nem futtatjuk a RIP daemont, erre az opcióra nincs szükségünk, de ha futtatjuk is, a metric értéket csak ritkán kell megváltoztatnunk.

`mtu` *bájt*
A maximális átviteli egységet (Maximum Transmission Unit, MTU), vagyis az egy tranzakcióban az interfész által kezelhető oktetek maximális számát állítja be. Ethernet hálózatokon az MTU alapértelmezett értéke 1500 (az Ethernet csomag legnagyobb megengedett mérete); SLIP interfészeken ez az érték 296. (A SLIP kapcsolatok MTU értékének nincs felső határa; az érték egyszerűen egy ésszerű kompromisszum.)

`arp`
Az opció kizárólag az olyan üzenetszóró hálózatokra vonatkozik, amilyen az Ethernet vagy a csomagrádió. Lehetővé teszi az ARP használatát a hálózatra kapcsolt gazdagépek fizikai címének észleléséhez. Üzenetszóró hálózatokon alapértelmezésben bekapcsolt állapotban van. Ha az ARP ki van kapcsolva, az *ifconfig* a `NOARP` kapcsolóval jelzi.

`-arp`
Az opció kikapcsolja az ARP használatát az interfészen.

`promisc`
Az interfészt lehallgató (promiscuous) üzemmódba állítja. Üzenetszóró hálózatokon ennek hatására az interfész minden csomagot fogad, függetlenül attól, hogy az adott gazdagépre szánták-e vagy sem. Így lehetővé válik a hálózati forgalom elemzése a csomagszűrők és egyéb eszközök segítségével; ezt szaglászásnak (snooping) is nevezik. Rendszerint alkalmas az olyan hálózati hibák felderítésére, melyeket egyébként csak nehezen észlelhetnénk. Olyan eszközök használják, mint például a tcpdump.

Másfelől az opció lehetővé teszi, hogy a támadók valóban csúnya dolgokat műveljenek, például „lefölözzék" hálózatunk forgalmát, és így jelszavak után kutassanak. Az ilyen támadások ellen úgy védekezhetünk, hogy senkit nem engedünk számítógépével a hálózatunkra kapcsolódni. Emellett használhatunk biztonságos hitelesítési pro-

* A RIP egy adott gazdagéphez vezető optimális útvonalat az út „hossza" alapján határozza meg, amit az egyes gazdagép-gazdagép kapcsolatok önálló metric értékeinek összegéből számít ki. Alapértelmezés szerint egy ugrás hossza 1, de bármilyen 16-nál kisebb pozitív egész szám lehet. (A 16-os útvonalhossz egyenlő a végtelennel. Az ilyen útvonalakat a rendszer használhatatlannak tekinti.) A **metric** paraméter ezt az ugrási költséget állítja be, amit azután az útválasztó daemon küld szét.

tokollokat is, például a Kerberost vagy a secure shell bejelentkezési csomagot.* Az opciónak a PROMISC kapcsoló felel meg.

`-promisc`
Kikapcsolja a lehallgató üzemmódot.

`allmulti`
A többcímes (multicast) szórási címek hasonlítanak az Ethernet csoportos (broadcast) szórási címekre azzal a különbséggel, hogy a többcímes szórási címre küldött csomagokat nem kapja meg mindenki automatikusan, csupán azok a kiszolgálók, melyeket e csomagok figyelésére állítottak be. Ennek olyan alkalmazások vehetik hasznát, mint például egy Ethernet alapú videokonferencia vagy hálózati audioátvitel, melyekre csak az érdekelt felek figyelnek. A legtöbb – bár nem az összes – Ethernet illesztőprogram támogatja a többcímes szórási címeket. Bekapcsolása esetén az interfész fogadja és feldolgozásra továbbítja a többcímes csomagokat. Az opció az ALLMULTI kapcsolónak felel meg.

`-allmulti`
Kikapcsolja a többcímes szórást.

A netstat parancs

A *netstat* remek eszköz a hálózat beállításainak és működésének ellenőrzésére. Valójában egy több eszközből álló gyűjteményről van szó. Egyes feladatait a következőkben vizsgáljuk meg.

Az útvonalválasztó táblázat kiíratása

Ha a *netstat* parancsot a `-r` kapcsolóval meghívjuk, megjeleníthetjük a rendszermag útvonalválasztó táblázatát, ahogy a route programnál tettük. A **vstout** esetén a következő eredményt kapjuk:

```
# netstat -nr
Kernel IP routing table
Destination  Gateway         Genmask           Flags   MSS Window  irtt Iface
127.0.0.1    *               255.255.255.255   UH        0 0          0 lo
172.16.1.0   *               255.255.255.0     U         0 0          0 eth0
172.16.2.0   172.16.1.1      255.255.255.0     UG        0 0          0 eth0
```

A -n opció hatására a **netstat** a címeket pontozott négyes IP számokkal jeleníti meg a szimbolikus gazdagép- és hálózatnevek helyett. Különösen hasznos, ha szeretnénk elkerülni, hogy a neveket a hálózaton keresztül kelljen kikeresni (például egy DNS vagy egy NIS kiszolgálóról).

A **netstat** kimenetének második oszlopa azt az átjárót jelöli, amelyre az útvonalbejegyzés mutat. Ha nem használunk útválasztót, helyette a csillag karakter jelenik meg. A harmadik oszlop az útvonal „általános" hálózatmaszkját mutatja. Amikor a rendszermag egy

* Az OpenSSH letölthető az *ftp://ftp.openbsd.org/OpenBSD/OpenSSH/portable* címről.

IP címhez a megfelelő útvonalat keresi, végigveszi az útvonalválasztó táblázat bejegyzéseit, és a címen és a genmaskon bitenkénti ÉS műveletet végez, mielőtt az útvonal céljával összevetné azokat; végül a legjobban illeszkedő találatot használja fel.

A negyedik oszlop az útvonalat leíró kapcsolókat jeleníti meg, a következőket:

G	Az útvonal átjárót használ.
U	A szükséges interfész aktív.
H	Az útvonalon keresztül csak egyetlen kiszolgáló érhető el. Ez a helyzet például a 127.0.0.1 visszacsatoló bejegyzéssel.
D	Az útvonal dinamikusan jön létre. Akkor van beállítva, ha a tábla bejegyzését egy útválasztó daemon, például a *gated,* vagy egy ICMP átirányító üzenet (lásd a 2. fejezetet) hozta létre.
M	Ez az útvonal akkor van beállítva, ha a tábla bejegyzését egy ICMP átirányító üzenet módosította.
I	Ez az útvonal egy elutasító útvonal, a datagramokat a rendszer eldobja.

A következő három oszlop az adott útvonal TCP kapcsolatának MSS, Window és irtt értékeit jelzi. Az MSS jelentése Maximum Segment Size, és arra a legnagyobb datagramra utal, amelyet a rendszermag ezen az útvonalon az átvitelhez létrehoz. A Window az a legnagyobb adatmennyiség, amelyet egyben elfogad a rendszer egy távoli gazdagéptől. Az *irtt* betűszó jelentése: kezdeti körbejárási idő. A TCP protokoll az adatok megbízható kézbesítését két gazdagép között úgy biztosítja, hogy az elveszett datagramokat ismételten elküldi. Ehhez folyamatosan méri, hogy mennyi időt vesz igénybe egy datagram eljuttatása a távoli véghez és a fogadást megerősítő üzenet visszaérkezése, így tudja, mikor szükséges ismételten elküldenie a datagramot. A körbejárási idő az az idő, amíg a datagram eljut a célgéphez és a nyugta visszaérkezik a vételről. Ez a normál üzenetvételi folyamat. A kezdeti körbejárási idő az az érték, melyet a TCP protokoll a kapcsolat első létrehozásakor feltételez. A legtöbb hálózattípusnál az alapértelmezett érték elfogadható, de a lassú hálózatokon, különösen a csomagrádió-hálózatok bizonyos típusain ez az idő túl rövid, így a protokoll feleslegesen küldi el újra a csomagokat. Az irtt értékét a *route* paranccsal állíthatjuk be. A mezők nulla értéke arra utal, hogy az alapértelmezett értékek érvényesek.

Végül az utolsó mező az útvonalhoz tartozó hálózati interfészt jeleníti meg.

Az interfészstatisztika kiíratása

Ha a `-i` kapcsolóval meghívjuk a *netstat* parancsot, a már beállított hálózati interfészekről jelenít meg statisztikát. Ha a `-a` opciót is megadjuk, a rendszermagban lévő *összes* interfészről beszámol, nem csak a beállítottakról. A **vstout** esetén a **netstat** kimenete így fest:

```
# netstat -i
Kernel Interface table
Iface MTU Met  RX-OK RX-ERR RX-DRP RX-OVR  TX-OK TX-ERR TX-DRP TX-OVR Flags
lo      0   0  3185      0      0      0  3185      0      0      0 BLRU
eth0 1500   0 972633     17     20    120 628711    217      0      0 BRU
```

Az `MTU` és `Met` mezők az interfész aktuális MTU és metric értékeit mutatják. Az `RX` és `TX` oszlopok azt jelzik, hogy hány csomag érkezett vagy távozott hibátlanul

(RX-OK/TX-OK), illetve sérült meg (RX-ERR/TX-ERR); hány csomagot dobott el az interfész (RX-DRP/TX-DRP); és hány veszett el túlcsordulás miatt (RX-OVR/TX-OVR).

Az utolsó oszlop az interfész beállított kapcsolóit mutatja. A következő karakterek az *ifconfig* utasítással lekérhető interfész-beállítások hosszú kapcsolóneveinek egybetűs rövidítései:

B	Az üzenetszórási cím be van állítva.
L	Az interfész egy visszacsatoló eszköz.
M	Minden csomag fogadása (promiscuous üzemmód).
O	Az ARP ki van kapcsolva az interfészen.
P	Ez egy pont-pont kapcsolat.
R	Az interfész fut.
U	Az interfész aktív.

A kapcsolatok kiíratása

A *netstat* számos opciót támogat az aktív és passzív socketek megjelenítésére. A -t, -u, -w és -x opciók az aktív TCP, UDP, RAW és Unix socketkapcsolatokat mutatják. Ha a -a kapcsolót is megadjuk, a kapcsolatra váró (vagyis figyelő) socketeket is láthatjuk. A kimenet ekkor a rendszeren futó összes kiszolgálót feltünteti.

A **vlager** esetén a *netstat* -ta utasítás a következő eredményt adja:

```
$ netstat -ta
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 localhost:mysql         *:*                     LISTEN
tcp        0      0 localhost:webcache      *:*                     LISTEN
tcp        0      0 *:www                   *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 *:https                 *:*                     LISTEN
tcp        0      0 ::ffff:1.2.3.4:ssh      ::ffff:4.5.6.:49152 ESTABLISHED
tcp        0    652 ::ffff:1.2.3.4:ssh      ::ffff:4.5.6.:31996 ESTABLISHED
```

A kimenetből láthatjuk, hogy a legtöbb kiszolgáló bejövő kapcsolatra vár. A negyedik sorban azonban a **vstoutról** érkező bejövő SMTP kapcsolatunk van, a hatodik sorból pedig megtudjuk, hogy a **vbardolino** kimenő *telnet* kapcsolatot futtat.*

A -a kapcsoló önmagában az összes család összes foglalatát megjeleníti.

Traceroute a kapcsolat ellenőrzésére

A gazdagépek közötti kapcsolatok és az útvonalak ellenőrzésének egyszerű módját kínálja a *traceroute* eszköz. A *traceroute* a csomagok útvonalának meghatározásához UDP (illetve a -I opcióval ICMP) datagramokat használ. A parancsot így hívhatjuk meg:

* Azt, hogy kimenő kapcsolatról van-e szó, a portszámok árulják el. A hívó gazdagép portszáma mindig egy egyszerű egész szám. A jól ismert szolgáltatási portot használó hívott gazdagéphez a *netstat* az */etc/services* állományból vett szimbolikus nevet (például smtp) adja meg. Persze ma már sok alkalmazásban megadhatjuk a forrásportunkat, így ez nem jelent garanciát.

```
# traceroute -n www.oreilly.com
traceroute to www.oreilly.com (208.201.239.37), 30 hops max, 40 byte packets
 1  22.44.55.23  187.714 ms  178.548 ms  177.132 ms
 2  206.171.134.130  186.730 ms  168.750 ms  150.769 ms
 3  216.102.176.193  168.499 ms 209.232.130.82
                                    194.629 ms 209.232.130.28  185.999 ms
 4  151.164.243.121  212.852 ms  230.590 ms  132.040 ms
 5  151.164.240.134  80.049 ms  71.191 ms  53.450 ms
 6  151.164.40.150  45.320 ms  44.579 ms  176.651 ms
 7  151.164.191.82  168.499 ms  194.864 ms  149.789 ms
 8  151.164.248.90  80.065 ms  71.185 ms  88.922 ms
 9  69.22.143.178  228.883 ms  222.204 ms  179.093 ms
10  69.22.143.6  131.573 ms  89.394 ms  71.180 ms
.
.
```

Az ARP táblák ellenőrzése

Bizonyos esetekben jól jöhet, ha megtekinthetjük, esetleg megváltoztathatjuk a rendszermag ARP tábláinak tartalmát – ilyen eset például, amikor azt gyanítjuk, hogy egy szabálytalan időközönként jelentkező hálózati hiba oka egy duplán létező internetcím lehet. Az *arp* eszközt ilyen helyzetekre tervezték. Parancssori opciói a következők:

```
arp [-v] [-t hwtype] -a [gazdagépnév]
arp [-v] [-t hwtype] -s gazdagépnév hwcím
arp [-v] -d gazdagépnév [gazdagépnév]
```

Minden *gazdagépnév* argumentum lehet szimbolikus gazdagépnév vagy pontozott négyes jelöléssel megadott IP cím.

Az első utasításhívás a megadott gazdagépnév vagy IP cím ARP bejegyzését mutatja, illetve a *gazdagépnév* nélküli meghívással az összes ismert gazdagép bejegyzését írja ki. Ha például a **valger** kiszolgálón adjuk ki az *arp* parancsot, az eredmény ehhez hasonló:

```
# arp -e
Address              HWtype    HWaddress           Flags Mask      Iface
172.16.0.1                     (incomplete)                        eth0
172.16.0.155         ether     00:11:2F:38:4E:4F   C               eth0
172.16.0.69          ether     00:90:4B:F1:3A:B5   C               eth0
vale.vbrew.com       ether     00:10:67:30:C5:7B   C               eth1
172.16.0.207         ether     00:0B:DB:1A:C7:E2   C               eth0
```

A kimenet néhány kiszolgáló Ethernet címét tartalmazza.

A `-s` opcióval a *gazdagépnév* Ethernet címét véglegesen az ARP táblához adhatjuk. A `hwcím` argumentum határozza meg a hardvercímet; alapértelmezés szerint ez egy kettőspontokkal elválasztott hat hexadecimális bájtból álló Ethernet cím. A `-t` opcióval más típusú hardver hardvercímét is beállíthatjuk.

Esetenként a távoli gazdagéphez intézett ARP lekérdezés sikertelen, mert például az ARP illesztőprogram hibás, vagy a hálózat egy másik gazdagépe tévesen azonosítja ma-

gát a gazdagép címével; ilyen hiba esetén manuálisan adhatjuk meg az IP bejegyzést az ARP táblához. A magukat másnak kiadó gazdagépek ellen is védekezhetünk, ha az ARP tábla IP címeit előre beállítjuk az Ethernet hálózatunkon, noha ez durva megoldás.

Ha az *arp* utasítást a `-d` kapcsolóval meghívjuk, az adott gazdagépre vonatkozó összes ARP bejegyzést eltávolíthatjuk. A kapcsoló akkor is jó szolgálatot tehet, ha az interfészt rá szeretnénk venni, próbálja meg ismételten kikeresni a kérdéses IP címhez tartozó Ethernet címet. Erre akkor lehet szükség, ha egy hibásan beállított rendszer rossz ARP információt sugárzott (előtte természetesen újra kell konfigurálni a hibás gazdagépet).

A `-s` opcióval *proxy* ARP-t hozhatunk létre. A különleges eljárás lényege, hogy egy gazdagép, például a **gate** az átjáró szerepét játssza egy másik gazdagéphez, mondjuk a **fnord**hoz, miközben úgy tesz, mintha mindkét cím egyazon kiszolgálóhoz, nevezetesen a **gate**-hez tartozna. Ehhez a **fnord** nevében a saját Ethernet interfészére mutató ARP bejegyzést bocsát ki. Így tehát amikor egy gazdagép ARP kérésben a **fnord** után érdeklődik, a **gate** válaszol, és válaszában a saját Ethernet címét küldi el. A kérést intéző gazdagép ekkor minden datagramot a **gate** gazdagépre küld, mely engedelmesen továbbítja azokat a **fnord**hoz.

Erre a csűrés-csavarásra akkor lehet szükségünk, ha például a **fnord**ot egy DOS gépről szeretnénk elérni, amely hibás TCP implementációja miatt nem alkalmas az útválasztásra. Proxy ARP-t használva a DOS gép számára úgy tűnik, mintha a **fnord** a helyi hálózaton volna, így nem szükséges tudnia, hogyan kezelje az átjárókat.

A proxy ARP akkor is kapóra jön, amikor valamely gazdagépünk csupán ideiglenesen játssza az átjáró szerepét egy másik gazdagéphez, például egy betárcsázó kapcsolaton keresztül. A korábban megismert **vlite** laptopunk időről időre PLIP kapcsolaton keresztül csatlakozik például a **vlager** géphez. Természetesen a kapcsolat csak akkor működik, ha annak a gazdagépnek a címe, amelyhez a proxy ARP-t biztosítani szeretnénk, az átjáróval azonos IP alhálózaton van. Ezért a **vstout** alkalmas proxy ARP lenne a sörgyári alhálózat (172.16.1.0) bármely gazdagépe számára, de a borászat alhálózatának (172.16.2.0) gazdagépei számára soha.

A **fnord** géphez a következő utasítással hozhatunk létre proxy ARP-t; természetesen a megadott Ethernet cím a **gate** címe kell legyen:

```
# arp -s fnord 00:00:c0:a1:42:e0 pub
```

A proxy ARP bejegyzést el is távolíthatjuk:

```
# arp -d fnord
```

5

A névszolgáltatás és beállítása

A 2. fejezetben láttuk, hogy a TCP/IP hálózatkezelésben különböző eljárásokkal fordíthatjuk a neveket címekre. A legegyszerűbb módszert az */etc/hosts* állomány gazdagép táblázata nyújtja. Ez azonban csak olyan kisebb méretű, helyi hálózatokon előnyös, amelyeket egyetlen rendszergazda kezel, és IP forgalmuk nem jut ki a külvilágba. A *hosts* állomány szerkezetét a 4. fejezetben ismertettük.

Noha a hosts állományon alapuló megközelítés a kis hálózatokon elfogadható, a rendszergazdák többségének a DNS kiszolgálót is meg kell ismernie. Az IP cím feloldására több szolgáltatást is igénybe vehetünk. Ezek közül a legelterjedtebb a *Berkeley Internet Name Domain* szolgáltatás (BIND) 8.x verziója. Egy ideje már a különféle új tulajdonságokkal felruházott, és a BIND 8.x verzió biztonságát megerősítő 9.x verzió is forgalomban van. A BIND 8 és BIND 9 közötti eltérések azonban nem olyan jelentősek, mint korábban a BIND 4 és BIND 8 közöttiek, mert sok konfigurációs állomány és beállítási lehetőség megegyezik. A BIND beállítása komoly kihívás, de ha egyszer túl vagyunk rajta, a hálózati topológiát könnyen módosíthatjuk. A Linuxon - hasonlóan számos más Unix-szerű rendszerhez - a BIND szolgáltatást a *named* nevű program biztosítja. Induláskor a program a működéséhez szükséges törzsállományokat a belső gyorsítótárába tölti, és megkezdi a várakozást a távoli vagy helyi felhasználói folyamatoktól érkező lekérdezésekre. A BIND beállítására több módszer is létezik, és nem mindegyik igényli, hogy valamennyi gazdagépen névkiszolgálót futtassunk.

Szó lesz egy egyszerűbb és megbízhatóbb megoldásról is, David J. Bernstein djbdns nevű programjáról. A szerző a feloldót úgy írta meg az alapoktól kezdve, hogy mindvégig a biztonságot tartotta szem előtt. A program a kiszolgálók beállítását számos módon egyszerűsíti, elsősorban azzal, hogy elkerüli a sok zavart okozó zónaállományok alkalmazását.

A fejezet célja nem csupán az, hogy felvázolja a DNS működését és a névkiszolgálók üzemeltetését, de megfelelő támogatást szeretne nyújtani a kis lokális hálózattal és internetes kapcsolattal rendelkező olvasóknak is. Aki ennél többre kíváncsi, a BIND és a djbdns forráscsomagokkal érkező dokumentációban megtalálja a súgóoldalakat és a verzióra vonatkozó legfrissebb információkat. A BIND csomagban ott rejtőzik továbbá a *BIND Operator's Guide* (BOG). A név (a. m. ingoványos) senkit ne riasszon el, valójában nagyon hasznos leírás. A DNS és a kapcsolódó témakörök átfogó ismertetését kínálja Paul Albitz és Cricket Liu könyve, a *DNS and BIND* (kiadója az O'Reilly). A DNS vonatkozású kérdéseinkre a *comp.protocols.tcp-ip.domains* hírcsoporton kereshetünk választ. A techni-

kai részletekről a tartománynév-rendszert leíró RFC 1033, 1034 és 1035 szabványokban tájékozódhatunk.

A feloldó könyvtár

A *feloldó* kifejezés nem valami különleges alkalmazásra, hanem a feloldó könyvtárra (resolver library) utal. A könyvtár a szabványos C könyvtárban tárolt – és egy sereg hálózatkezelő alkalmazásban igénybe vett – függvénygyűjtemény. A legfontosabb eljárások a *gethostbyname(2)* és a *gethostbyaddr(2),* melyek feladata egy gazdagépnévhez tartozó összes IP cím kikeresése, és fordítva. Beállításuktól függően a keresést a *hosts* állományban vagy a DNS névkiszolgálókon végzik.

Futtatáskor a feloldó függvények konfigurációs állományokat olvasnak be. Ezekből határozzák meg, hogy mely adatbázisokat és milyen sorrendben kérdezzenek le, de ezekből ismerik meg a rendszer beállításától függő egyéb fontos körülményeket is. A Linux régebbi szabványos könyvtára, a libc mint fő konfigurációs állományt az */etc/host.conf* állományt tartalmazta, a GNU szabványos könyvtárának 2-es verziójától azonban az */etc/nsswitch.conf* állományban találjuk a beállításokat.

Az nsswitch.conf állomány

Az *nsswitch.conf* állományban a rendszergazda a különböző adatbázisok széles skáláját állíthatja be. Most csak azokkal a lehetőségekkel foglalkozunk, amelyek a gazdagépcímek és a hálózati IP címek feloldásával kapcsolatosak. A további lehetőségekről bővebben a GNU szabványos könyvtárának dokumentációjában olvashatunk.

Az *nsswithc.conf* állományban minden opciót külön sorban adunk meg. A mezőket whitespace karakterekkel (szóköz vagy tabulátor) választjuk el egymástól. A kettőskereszt (#) megjegyzést vezet be, amely az újsor karakterig tart. Minden sor egy adott szolgáltatást ír le, például a gazdagépnév feloldását. Minden sor első mezője az adatbázis neve, melyet kettőspont zár le. A gazdagépek címfeloldási adatbázisa a `hosts`. Ehhez kapcsolódik a `networks` adatbázis, melyet akkor használunk, ha hálózatneveket hálózatcímekre oldunk fel. A sor fennmaradó részében adhatjuk meg azokat a beállításokat, amelyek az adott adatbázisban végzendő keresés módját írják le.

A következő opciókat adhatjuk meg:

`dns`
: A címfeloldáshoz a DNS szolgáltatást használja. Csak a gazdagépek címének feloldásakor van értelme, a hálózatcímek esetében nincs. Az eljárás az *fs* állománnyal dolgozik, mellyel a fejezet későbbi részében ismerkedünk meg.

`files`
: A gazdagép vagy hálózat nevét és a hozzá tartozó címet egy helyi állományban keresi. Az opció a hagyományos */etc/hosts* és */etc/networks* állományokra támaszkodik.

A lekérdezendő szolgáltatások megadott sorrendje szabja meg, hogy a névfeloldáskor milyen sorrendben történik a lekérdezés. A sorrendet meghatározó listát az */etc/nss-*

witch.conf állomány szolgáltatásleírása tartalmazza. A szolgáltatások lekérdezése balról jobbra történik, és alapértelmezés szerint a keresés befejeződik, ha a feloldás sikeres.

A gazdagépek és a hálózatok adatbázisának megadását mintázza az 5.1. példa, amely a régebbi libc szabványos könyvtárt használja.

5.1. példa. Minta nsswitch.conf állomány

```
# /etc/nsswitch.conf
#
# Minta GNU Name Service Switch beállítás.
# Az állományról a `libc6-doc' csomagban találunk információt.

hosts:          dns files
networks:       files
```

A rendszer a gazdagépeket először a DNS kiszolgálón, majd ha az sikertelen, az */etc/hosts* állományban keresi. A hálózatnevek kikeresését csak az */etc/networks* állományban kísérli meg.

A keresési művelet pontosabb szabályozását teszik lehetővé a „műveleti elemek", melyekkel megadhatjuk, hogy az előző kikeresés eredményétől függően milyen művelet következik. A műveleti elemek helye a szolgáltatásleírások között szögletes zárójelben van: [ ]. A műveleti utasítás általános szintaxisa:

```
[ [!] állapot = művelet ... ]
```

Két lehetséges művelet közül választhatunk:

`return`
A végrehajtás visszakerül ahhoz a programhoz, amely a névfeloldást megkísérelte. Ha a kikeresés sikeres, a feloldó a részletekkel tér vissza; ellenkező esetben nullával.

`continue`
A feloldó a lista következő szolgáltatásához lép és ismét megkísérli a névfeloldást.

Az opcionális (!) karakter előírja, hogy az állapot értékét tesztelés előtt meg kell fordítani; más szóval a logikai „NEM" megfelelője.

A további műveletet meghatározó állapotértékek a következők:

`success`
A kívánt bejegyzést hiba nélkül sikerült kikeresni. Az értékhez tartozó alapértelmezett művelet a `return`.

`notfound`
Kikeresés közben nem történt hiba, de a keresett gazdagép vagy hálózat nem található. Az értékhez tartozó alapértelmezett művelet a `continue`.

`unavail`
A kért szolgáltatás nem elérhető. Jelentheti azt, hogy a `files` szolgáltatás nem volt képes a *hosts* vagy *networks* állomány olvasására, vagy hogy a névkiszolgáló vagy NIS szerver nem válaszolt a `dns` vagy `nis` szolgáltatásoknak. Az értékhez tartozó alapértelmezett művelet a `continue`.

`tryagain`
Az érték arra utal, hogy a szolgáltatás ideiglenesen nem elérhető. A `files` szolgáltatás esetén ez rendszerint azt jelenti, hogy a releváns állományt egy folyamat zárolta. Más szolgáltatásoknál azt jelezheti, hogy a kiszolgáló ideiglenesen képtelen a kapcsolatok fogadására. Az értékhez tartozó alapértelmezett művelet a `continue`.

Az eljárásra mutat egyszerű mintát az 5.2. példa.

5.2. példa. Minta nsswitch.conf állomány egy műveleti utasítással

```
# /etc/nsswitch.conf
#
# Minta GNU Name Service Switch beállítás.
# Az állományról a `libc6-doc' csomagban találunk információt.

hosts:          dns [!UNAVAIL=return] files
networks:       files
```

A példa a gazdagép nevének feloldását a DNS segítségével kísérli meg. A feloldó minden esetben a kapott eredménnyel tér vissza, bármi legyen is az, kivéve ha a visszatérési állapot a „nem elérhető". Ha a DNS kikeresési kísérlet a nem elérhető értékkel tér vissza, akkor, és csak akkor a feloldó megkísérli a keresést a helyi */etc/hosts* állományban. Más szóval, a *hosts* állományt csak akkor vesszük igénybe, ha valamilyen oknál fogva a névkiszolgáló nem elérhető.

A névkiszolgáló kikereséseinek beállítása a resolv.conf állománnyal

Amikor a gazdagépek kikereséséhez a feloldó könyvtárt a DNS névszolgáltatás használatára utasítjuk, azt is meg kell határoznunk, melyik névkiszolgálót vegye igénybe. Erre egy önálló állomány, a *resolv.conf* szolgál. Ha az állomány nem létezik vagy üres, a feloldó feltételezi, hogy a névkiszolgáló a helyi gazdagépen van.

Azért, hogy a helyi gazdagépünkön névkiszolgálót futtathassunk, külön be kell állítanunk azt - erről később, *A DNS működése* című részben olvashatunk. Ha azonban a helyi hálózaton hozzáférünk egy már létező névkiszolgálóhoz, ragaszkodjunk ahhoz. Ha betárcsázó IP kapcsolattal csatlakozunk az internetre, a *resolv.conf* állományban általában a szolgáltatónk névkiszolgálóját adjuk meg.

A *resolv.conf* legfontosabb opciója a `nameserver`, amely a használni kívánt névkiszolgáló IP címét határozza meg. Amikor a `nameserver` opciót többször is megadjuk, és így több névkiszolgálót jelölünk ki, a megadott sorrendben történik a lekérdezésük. Ezért mindig a legmegbízhatóbb kiszolgálót vegyük előre. A jelenlegi implementáció-

ban akár három `nameserver` utasítást is elhelyezhetünk a *resolv.conf* állományban. Ha egyet sem adunk meg, a feloldó a helyi gazdagép névkiszolgálójához próbál csatlakozni.

Két másik opció, a `domain` és a `search` lehetővé teszi, hogy rövid neveket használjunk a helyi tartomány gazdagépeihez. Ha csupán a tartomány egy gazdagépét szeretnénk elérni, rendszerint nem adjuk meg a teljes minősített gazdagépnevet, csak a rövid nevét írjuk be a parancssorba, például `gauss`, és hagyjuk, hogy a feloldó járjon utána a **mathematics.groucho.edu** résznek.

Pontosan erre szolgál a `domain` utasítás. Segítségével megadhatunk egy alapértelmezett tartománynevet, melyet sikertelen keresés esetén a DNS a gazdagépnévhez fűz. A **gauss** esetén például a feloldó először megkísérli a **gauss** kikeresését a DNS-ben, de kudarcot vall, mert nincs ilyen felső szintű tartomány. Amikor a **mathematics.groucho.edu** tartományt adjuk meg alapértelmezett tartományként, a feloldó ismételten megkísérli a **gauss** kikeresését, hozzákapcsolva az alapértelmezett tartománynevet – és ezúttal sikerrel jár.

Milyen nagyszerű! – gondolhatnánk. De abban a pillanatban, amikor kikerülünk a Matematika tanszék tartományából, ismét a teljes minősített tartománynevekkel találjuk szembe magunkat. Természetesen azt szeretnénk, ha a Fizika tanszék tartományában lévő gazdagépeknek is volna rövid nevük, például **quark.physics**.

Most segíthet a *keresési lista*. A keresési listát a `search` opcióval adhatjuk meg, amely a `domain` utasítás általánosabb célú megfelelője. Míg az utóbbi egyetlen alapértelmezett útvonalat ír elő, az előbbi egy teljes listát, melynek elemeit sorra kipróbálhatjuk, amíg a keresés sikeres nem lesz. A lista elemeit szóközökkel vagy tabulátorokkal választjuk el.

A `search` és `domain` utasítások kölcsönösen kizárják egymást, és csak egyszer fordulhatnak elő. Ha egyik opciót sem adjuk meg, a feloldó az alapértelmezett tartományt a helyi gazdagépnévből próbálja meghatározni a *getdomainname(2)* rendszerhívással. Ha a helyi gazdagépnév nem tartalmazza a tartományt, az alapértelmezett tartomány a gyökértartomány lesz.

Tegyük fel, hogy a Virtuális Sörgyárban ülve szeretnénk bejelentkezni a **foot.groucho.edu** gazdagépre, de elvétjük a nevet, és a **foot** helyett a nem létező **foo** gépre hivatkozunk. A GME névkiszolgálója ekkor megüzeni, hogy ilyen gazdagépről nem tud. A régebbi keresési eljárások esetén most a feloldó megpróbálná kikeresni a **vbrew.com** és a **com** résszel kiegészített neveket. Ez utóbbival azonban gond van, mert a **groucho.edu.com** akár egy érvényes tartománynév is lehet, ahol a névkiszolgáló talán még a **foo** gépet is megtalálja, odairányítva bennünket, holott nem azt keressük.

Bizonyos alkalmazásoknál az ilyen jellegű hamis keresések biztonsági réseket nyithatnak. Ezért a keresési lista tartományát rendszerint érdemes leszűkítenünk a helyi szervezetre vagy más ésszerű tartományra. A Groucho Marx Egyetem Matematika tanszékének keresési listáját például a **maths.groucho.edu** és a **groucho.edu** tartományra állíthatnánk be.

Ha az alapértelmezett tartományok fogalma még nem lenne teljesen világos, vegyük a következő példát, amely a Virtuális Sörgyár *resolv.conf* állománya:

```
# /etc/resolv.conf
# A tartományunk
domain          vbrew.com
```

```
#
# Legyen a vlager a központi névkiszolgáló:
nameserver      172.16.1.1
```

A feloldó a **vale** név feloldásakor először a **vale** nevet, majd sikertelenség esetén a **vale.vbrew.com** nevet keresné.

A feloldó hibatűrése

Ha egy nagyobb hálózaton belül lokális hálózatot üzemeltetünk, feltétlenül központi névkiszolgálókat alkalmazzunk, ha ez lehetséges. A névkiszolgálók hatalmas gyorsítótárakat hoznak létre, és mivel minden lekérdezés hozzájuk fut be, az ismétlődő keresésekhez szükséges idő lerövidül. A rendszernek van azonban egy bökkenője: amikor a könyv egyik szerzőjének egyetemén a gerincvezetéket tűz pusztította el, lehetetlenné vált a munka a tanszék helyi hálózatán, mert a feloldó képtelen volt elérni a névkiszolgálókat. A kiesés a legtöbb hálózati szolgáltatásban is gondot okozott, például az X terminálokra történő bejelentkezésekben és a nyomtatásban.

Bár nem gyakori, hogy az egyetemi gerincvezeték füstté váljon, az ilyen esetekre is fel kell készülnünk.

Az egyik lehetőség, hogy létrehozunk egy helyi névkiszolgálót a helyi tartomány gazdagépneveinek feloldásához, a többi gazdagépnév kikeresését pedig a fő kiszolgálókhoz továbbítjuk. Természetesen ez csak akkor lehetséges, ha saját tartománnyal rendelkezünk.

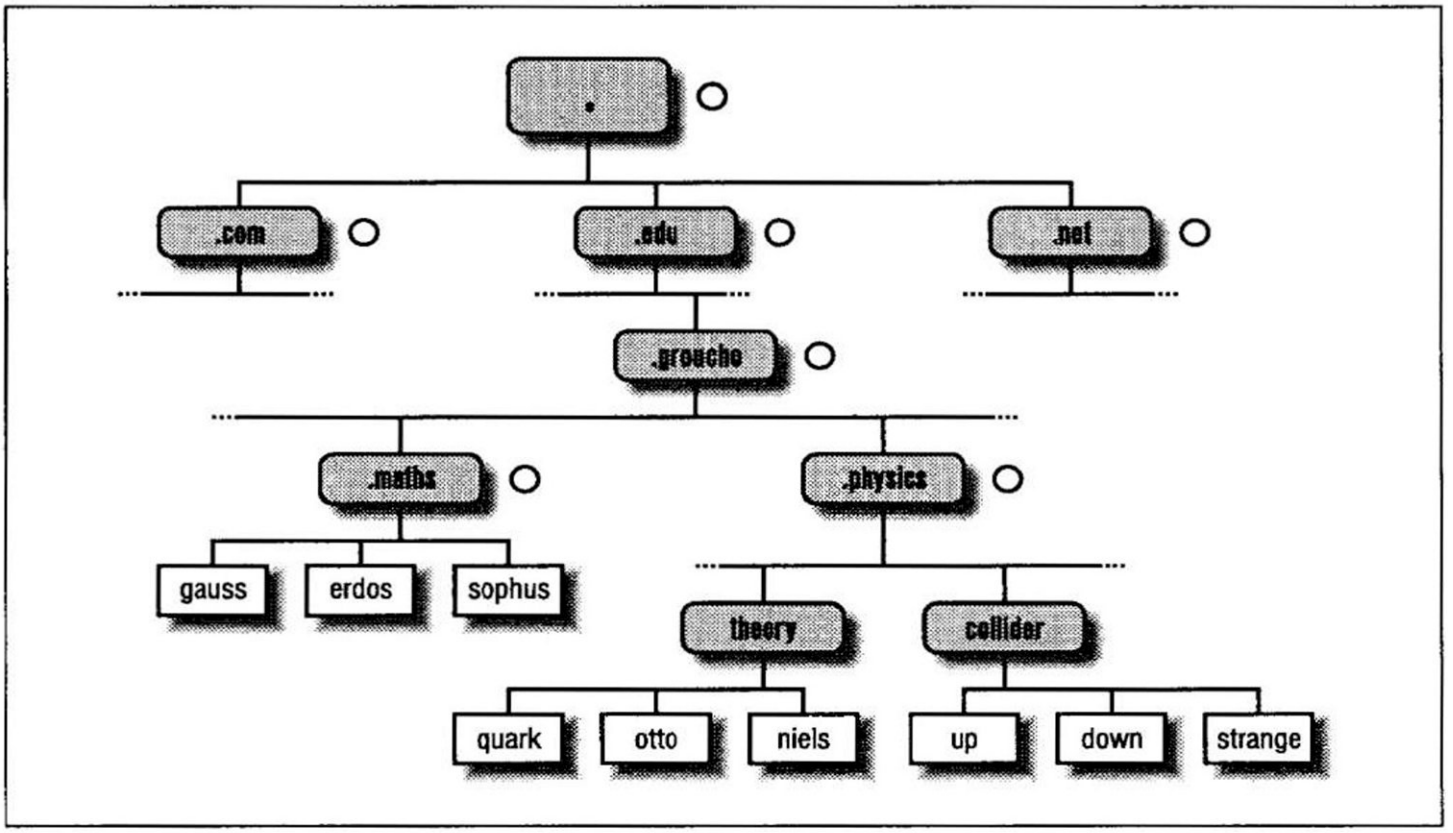
A másik lehetőség, hogy az */etc/hosts* állományban elhelyezünk egy biztonsági másolatot a tartomány vagy LAN gazdagépeinek táblájáról. Csupán arra ügyeljünk, hogy a feloldó könyvtár először a DNS-t kérdezze le, és csak utána forduljon a *hosts* állományhoz. Az */etc/nsswitch.conf* állományban elhelyezett `hosts: dns files` utasítással vehetjük rá a feloldót, hogy csak akkor használja a *hosts* állományt, ha a központi névkiszolgáló elérhetetlen.

A DNS működése

A DNS a gazdagépneveket tartományhierarchiába szervezi. A *tartomány* (domain) olyan hálózati helyek gyűjteménye, amelyek bizonyos szempontból összetartoznak – az összetartozás alapja lehet, hogy együtt alkotnak hálózatot (például az egyetem számítógépei), vagy egyetlen szervezethez tartoznak (például a kormányhoz), vagy pedig egész egyszerűen földrajzilag egymás közelében helyezkednek el. Az egyetemeket például gyakran csoportosítják az **edu** tartományba, de az egyes egyetemek és főiskolák önálló *altartományt* (subdomain) használnak, ezek foglalják magukban gazdagépeiket. A Groucho Marx Egyetem tartománya a **groucho.edu**, a Matematika tanszék lokális hálózatáé pedig a **maths.groucho.edu**. A tanszék hálózatának gazdagépei ezt a tartománynevet „öröklik", így az **erdos** neve például **erdos.maths.groucho.edu** lenne, vagyis ez volna az FQDN (lásd a 4. fejezet *A gazdagép nevének beállítása* című részét).

Az 5.1. ábra a névtartomány-szerkezet egy részletét mutatja be. A bejegyzés a fa gyökerénél, melyet egyetlen pont jelöl, érthető módon a *gyökértartomány* nevet viseli, és

minden más tartományt magában foglal. Annak jelzésére, hogy egy gazdagépnév FQDN, és nem valamely (implicit) helyi tartományhoz viszonyított, esetenként pontot tesznek a végére. A pont arra utal, hogy a név utolsó komponense a gyökértartomány. A névhierarchiában elfoglalt helyétől függően egy tartomány lehet felső szintű, második vagy harmadik szintű. Az alosztályoknak további szintjei is lehetnek, de ez ritkán fordul elő. Az 5.1. táblázat a gyakoribb felső szintű tartományokat foglalja össze.



5.1. ábra. A tartomány névtérrészlete

5.1. táblázat. Gyakori felső szintű tartományok

Tartomány	*Leírás*
edu	(Leginkább észak-amerikai) oktatási intézmények, például egyetemek.
com	Gazdasági szervezetek és vállalatok.
org	Nem gazdasági szervezetek.
net	Eredetileg átjárók és más adminisztratív entitások, ma már kereskedelmi szervezetek és vállalatok is.
mil	Az Egyesült Államok katonai intézményei.
gov	Az Egyesült Államok kormányának intézményei.
biz	Vállalatok és gazdasági entitások.
name	Egyének számára, személyes webhelyekhez.
info	A tájékoztatást szolgáló hálózati helyek részére.

Valamikor az első négy tartományt az Egyesült Államok részére tartották fenn, de a közfelfogás változásának eredményeképp ma már ezeket az úgynevezett globális felső szintű tartományokat (global Top-Level Domains, gTLD) valóban globális jellegűnek fo-

gadjuk el. A közelmúltban a gTLD tartományok bővítéséről tartott egyeztetéseknek köszönhető az utolsó három tartomány felvétele, melyek azonban mindeddig meglehetősen népszerűtlennek bizonyultak.

Az országoknak általában saját felső szintű tartományuk van. Nevük az ISO-3166 specifikációban meghatározott kétbetűs országkódokat követi. Finnország tartománya például **fi**; **fr** Franciaországot jelöli, **de** Németországot, **aq** az Antarktiszt. A felső szintű tartományon belül az egyes országok NIC központjai szabadon, tetszésük szerint alakíthatják ki a gazdagépneveket. Ausztráliában a nemzetközi felső szintű tartományokhoz hasonló második szintű tartományokat találunk, például **com.au** és **edu.au** néven. Más országokban, például Németországban nem alkalmaznak ilyen extra szintet, hanem olyan, némileg hosszabb neveket, melyek közvetlenül hivatkoznak egy adott tartományt fenntartó szervezetre. Gyakran látni ilyen és hasonló gazdagépneveket: *ftp://ftp.informatik.uni-erlangen.de*. Írjuk ezt a német alaposság számlájára.

Természetesen az országos tartományok nem feltétlenül jelentik azt, hogy a tartomány valamely gazdagépe valóban az adott ország területén található; csupán azt jelzik, hogy a gazdagépet az adott ország NIC központjában regisztrálták. Egy svéd vállalkozás Ausztriában megnyitott fiókvállalatának gazdagépeit például regisztrálhatják az **se** felső szintű tartományban.

Ez a gyakorlat az elmúlt években egyre népszerűbbé vált olyan országokban, mint például Tuvalu (**tv**) és Togo (**to**), ahol a felső szintű tartományokat ügynökségeknek adják el, amelyek azután olyan tartományneveket árusítanak, mint a **go.to** vagy **watch.tv**.

A névtér felosztása hierarchikus tartománynevekre megoldja az egyedi nevek problémáját is; a DNS esetében egy gazdagépnévnek csak a saját tartományán belül kell egyedinek lennie, hogy a világon létező minden más gazdagéptől eltérő nevet kaphasson. Mi több, a teljes minősített neveket könnyű megjegyezni. Ezek az okok már önmagukban is elegendőek, hogy egy nagy tartományt több altartományra osszunk fel.

De a DNS még ennél is többet kínál: lehetővé teszi, hogy egy altartomány kezelési jogát az alhálózat rendszergazdáira ruházzuk át. Például a Groucho Számítási Központ fenntartói minden tanszék részére létrehoznak egy alhálózatot; a Matematika és a Fizika tanszékek alhálózataival már találkoztunk. Amikor a fenntartók úgy látják, hogy a Fizika tanszék hálózata kinőtte magát, és kívülről már nem lehet megbirkózni a kaotikus állapotokkal (végül is a fizikusok köztudottan fegyelmezetlen banda), a **physics.groucho.edu** tartomány irányítását egyszerűen átadhatják a hálózat rendszergazdáinak. A rendszergazdák maguk dönthetik el, milyen gazdagépneveket választanak, és hogyan osztják ki az IP címeket a hálózatukon - a külső hálózat vonatkozásában ez nem okozhat zavart.

Mindezt úgy érik el, hogy a névtartományt zónákra osztják, melyek mindegyike egy-egy tartományban „gyökerezik". Vegyük észre a zóna és a tartomány közötti finom különbséget: a **groucho.edu** tartomány a Groucho Marx Egyetem összes gazdagépét felöleli, míg a **groucho.edu** zóna csupán a Számítási Központ közvetlen kezelésében álló gazdagépeket tartalmazza - például a Matematika tanszék gépeit. A Fizika tanszék gazdagépei más zónához tartoznak: ez a **physics.groucho.edu**. Az 5.1. ábrán a zónák kezdetét a tartománynevek után álló kis körök jelzik.

Névkikeresés a DNS rendszerrel

Első látásra úgy tűnhet, hogy a sok hűhó a tartományok és zónák körül az egész névfeloldást igencsak megbonyolítja. Végül is, ha nincs olyan központi hatóság, amely a gazdagépek nevének kiosztását szabályozná, hogyan is várhatjuk el, hogy egy szerény alkalmazás képes legyen rá?

Most következik azonban a DNS valóban leleményes része! Ha például szeretnénk megtudni az **erdos** IP címét, a DNS így válaszol: „Menj, és kérdezd meg azoktól, akik kezelik, ők majd megmondják."

A DNS tulajdonképpen egy hatalmas elosztott adatbázis. Alapját az úgynevezett névkiszolgálók képezik, melyek adott tartományról vagy tartományokról közölnek információt. Minden zónához legalább két, esetleg még néhány névkiszolgáló tartozik, ezek tárolják a létező összes hiteles információt a zóna gazdagépeiről. Az **erdos** IP címének megszerzéséhez csupán fel kell vennünk a kapcsolatot a **groucho.edu** zóna névkiszolgálójával, amely megadja a kért adatokat.

Könnyű azt mondani, gondolhattuk volna. De honnan tudjuk meg, hogyan érhetjük el a névkiszolgálót a Groucho Marx Egyetemen? Arra az esetre, ha nem lakna a számítógépünkben egy mindentudó címfeloldó orákulum, a DNS erre is kínál megoldást. Amikor alkalmazásunk információt szeretne az **erdos** gazdagépről, kapcsolatba lép a helyi névkiszolgálóval, amely úgynevezett iteratív, ismételt kérdezésbe kezd. Mindenekelőtt megkérdezi az **erdos.maths.groucho.edu** címét a gyökértartomány névkiszolgálójától. A gyökér névkiszolgálója felismeri, hogy a név nem a felügyelete alá eső zónához tartozik, hanem egy másikhoz az **edu** tartományban. Így azt javasolja, hogy további információért forduljunk egy **edu** zóna névkiszolgálójához, és segítségképpen megadja az összes edu névkiszolgáló nevét a címmel együtt. A helyi névkiszolgálónk ezután közülük kérdez meg egyet, legyen mondjuk az **a.isi.edu**. A gyökér névkiszolgálójához hasonlóan az **a.isi.edu** is tudja, hogy a **groucho.edu** fiúk saját zónát futtatnak, így az ő kiszolgálóikra irányít bennünket. A helyi névkiszolgáló ekkor az **erdos**ra vonatkozó kérdésével e kiszolgálók egyikéhez fordul, amely végül a saját zónájához tartozónak ismeri fel a nevet, és megküldi a hozzá tartozó IP címet.

Ekkora forgalom egy vacak IP cím miatt? Figyelembe véve azonban a mai hálózatok sebességét, ez szinte jelentéktelen, noha az eljáráson még javíthatunk.

A jövőbeni lekérdezések válaszidejének lerövidítésére a névkiszolgáló a kapott információt a helyi gyorsítótárában tárolja. Így amikor legközelebb valaki a helyi hálózatunkról a **groucho.edu** tartomány egy gazdagépének címét keresi, a névkiszolgáló közvetlenül a **groucho.edu** névkiszolgálóhoz fordul.*

Természetesen a névkiszolgáló az információt nem őrzi a végtelenségig, előbb-utóbb megszabadul tőle. Az információ érvényességi idejét *élettartamnak* (time to live, ttl) nevezzük. Az adott zóna rendszergazdái a DNS adatbázis minden adatához hozzárendelnek ilyen ttl időtartamot.

* Ha nem tárolnánk az információt gyorsítótárban, a DNS hatékonysága ugyanolyan rossz volna, mint bármely más eljárásé, mert minden lekérdezéshez be kellene vonni a gyökérkiszolgálókat is.

A névkiszolgálók típusai

Azokat a névkiszolgálókat, amelyek egy zóna gazdagépeiről az összes információt tárolják, a zóna *meghatalmazott* (authoritative) névkiszolgálóinak hívjuk. A zóna valamely gazdagépére vonatkozó kérés végül e névkiszolgálók egyikéhez jut.

Nagyon fontos a meghatalmazott kiszolgálók szinkronizálása. Ebből a célból a zóna rendszergazdája kijelöl egy *elsődleges* kiszolgálót, amely a zónainformációt adatbázisokból tölti be, az összes többit pedig *másodlagos* kiszolgálónak minősíti - ezek meghatározott időközönként átveszik a zónaadatokat az elsődleges kiszolgálóról.

Több névkiszolgálóval eloszthatjuk a terhelést, ezen felül biztonsági másolatot is jelentenek. Amikor egy névkiszolgálón jóindulatú hiba történik, például összeomlik vagy elveszíti hálózati kapcsolatát, minden lekérdezés a többi kiszolgálóra hárul. Természetesen ez még nem véd meg bennünket a kiszolgáló hibás működésétől, például magának a kiszolgálóprogramnak a hibájától, amely minden DNS lekérdezéskor rossz választ eredményez.

Olyan névkiszolgálót is üzemeltethetünk, amely egyetlen tartományra nézve sem meghatalmazott.* Ez is hasznos lehet, mert a névkiszolgáló így is képes a helyi hálózaton futó alkalmazások DNS kéréseit intézni és az eredményeket gyorsítótárban tárolni. Épp ezért ezeket *csak tároló* kiszolgálóknak nevezzük.

A DNS adatbázis

Láttuk, hogy a DNS nem csak a gazdagépek IP címeit kezeli, de a névkiszolgálókon információt is cserél. Ami azt illeti, a DNS adatbázisok számos különféle típusú bejegyzést tartalmazhatnak.

A DNS adatbázisban tárolt információ legkisebb önálló egysége a *forrásrekord* (resource record, RR). Minden rekordnak van egy típusa, amely a rekord által leírt adat fajtáját azonosítja, valamint egy osztálya, amely megmutatja, hogy milyen típusú hálózatra vonatkozik. Ez utóbbi segít összeegyeztetni a különböző címzési sémák kívánalmait, beleértve az IP címeket (az IN osztályt), a Hesiod címeket (ezt használja a MIT Kerberos rendszere) és még egypárat. A forrásrekord prototípusa az A rekord, amely egy minősített teljes tartománynévhez társít egy IP címet.

Egy gazdagépnek több neve is ismert lehet. Tegyük fel például, hogy kiszolgálónk egyidejűleg kínál FTP és World Wide Web szolgáltatást, ezért két nevet adunk neki: *ftp.machine.org* és *www.machine.org*. Azonban e neveknek csak az egyike a hivatalos, vagy *kanonikus* gazdagépnév, a többi egyszerűen a hivatalos gazdagépnévre utaló alias név. A különbség annyi, hogy a kanonikus gazdagépnévhez egy A rekord tartozik, a többihez pedig csak a kanonikus gazdagépnévre mutató CNAME típusú rekord.

Nem foglalkozunk most az összes rekordtípussal, de bemutatunk egy rövid példát. Az 5.3. példa egy tartomány adatbázisának részletét szemlélteti, ezt töltjük a **physics.groucho.edu** zóna névkiszolgálóiba.

* Nos, ez így nem igaz, mert minden névkiszolgálónak legalábbis a localhosthoz és a **127.0.0.1** fordított kikeresésekhez nyújtania kell névszolgáltatást.

5.3. példa. A Fizika tanszék egy BIND zónaállományának részlete

```
; Megbízható információ a physics.groucho.edu tartományról.
$TTL 3D
@  IN  SOA niels.physics.groucho.edu. janet.niels.physics.groucho.edu. (
                2004010200       ; serial no
                8H               ; refresh
                2H               ; retry
                4W               ; expire
                1D               ; default ttl
              )
;
; Névkiszolgálók
             IN    NS       niels
             IN    NS       gauss.maths.groucho.edu.
gauss.maths.groucho.edu. IN A 149.76.4.23
;
; Elméleti Fizika tanszék (12-es alhálózat)
niels        IN    A        149.76.12.1
             IN    A        149.76.1.12
nameserver    IN    CNAME    niels
otto         IN    A        149.76.12.2
quark        IN    A        149.76.12.4
down         IN    A        149.76.12.5
iowa         IN    AAAA     2001:30fa::3
strange      IN    A        149.76.12.6
...
; Részecskevizsgáló laboratórium (14-es alhálózat)
boson        IN    A        149.76.14.1
muon         IN    A        149.76.14.7
bogon        IN    A        149.76.14.12
...
```

Eltekintve az A és a CNAME rekordoktól, az állomány elején egy többsoros különleges rekordot is találunk. Ez az SOA forrásrekord, és a *meghatalmazás kezdetét* (Start of Authority) jelzi. Ebben általános információt találunk a zónáról, amelyre a kiszolgálónak meghatalmazása van. Az SOA rekord foglalja magában például az összes rekord alapértelmezett élettartamát.

A mintaállományban szereplő összes nevet, amely nem pontra végződik, a **physics.groucho.edu** tartományhoz viszonyítva értelmezzük. Az SOA rekordban használt különleges név (@) magára a tartománynévre utal.

Korábban láttuk, hogy a **groucho.edu** tartomány névkiszolgálóinak valamilyen módon tudniuk kell a **physics** zónáról, különben a lekérdezéseket nem irányíthatnák a zóna névkiszolgálóihoz. Az információkért rendszerint két rekord felel: egy NS rekord, amely a kiszolgáló FQDN nevét adja meg, és egy A rekord, amely egy címet rendel a névhez. Mivel ezek a rekordok tartják egyben a névtartományt, gyakran *ragasztó (glue) rekordnak* is nevezik őket. Ezeken kívül nincs más rekord, amelyben egy szülő zóna ténylegesen információt tárolna az alzónák gazdagépeiről. A **physics.groucho.edu** névkiszolgálóira mutató ragasztó rekordokat láthatjuk az 5.4. példában.

5.4. példa. Részlet a GME egy zónaállományából

```
; Zónaadat a groucho.edu zónához.
...
 ;
 ; Ragasztó rekordok a physics.groucho.edu zónához
 physics          IN     NS        niels.physics.groucho.edu.
                  IN     NS        gauss.maths.groucho.edu.
 niels.physics    IN      A         149.76.12.1
 gauss.maths      IN      A         149.76.4.23
...
```

A BIND named.conf állomány

A BIND elsődleges konfigurációs állománya az */etc/named.conf*. A BIND azóta használ ilyen típusú konfigurációs állományt, amióta a 8-as verzió leváltotta a régi *named.boot* állományt.

A szintaxis meglehetősen összetett és a tevékenységek széles skáláját támogatja, de ha csak az alapvető funkcionalitást kívánjuk beállítani, könnyen boldogulhatunk vele. Az 5.5. példa a **vbrew** tartományhoz mutat be egy egyszerű konfigurációs állományt.

5.5. példa. A vlager gazdagép BIND named.conf állománya

```
// A named BIND DNS kiszolgáló elsődleges konfigurációs állománya.
//

options {
            directory "/var/cache/bind";
        allow-query { any; };
        recursion no;

zone "localhost" {
        type master;
        file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
        type master;
        file "/etc/bind/db.127";
};

zone "vbrew.com" {
        type master;
    allow-transfer { 10.10.0.5;
                172.16.90.4;
                1.2.3.4;
        };
        file "/etc/bind/db.vbrew.com";
};

zone "0.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168.0";
};
```

Közelebbről megnézve láthatjuk, hogy a *named.conf* állomány utasításait a C utasításokhoz hasonlóan írjuk, az attribútumokat { } karakterek közé helyezve.

A megjegyzéseket, melyeket Linuxban kettős kereszt (#) karakterrel jelzünk, itt előre dőlő perjelek (//) vezetik be.

A konfigurációs állomány egyik legfontosabb opciója a `zone`. Ennek segítségével adjuk meg azokat az információkat, amelyeket a BIND-nak ismernie kell. A `zone` alatt egy másik nagyon fontos opció, az `allow-transfer` látható. Ezzel az opcióval állíthatjuk be azokat az IP címeket, ahol engedélyezzük a zónatranszfert. Nagyon fontos, hogy ezt a jogot kizárólag a meghatalmazott entitásokra korlátozzuk, mert rengeteg, a potenciális támadók számára hasznos információt tartalmaz. A minta konfigurációs állományban a zónatranszfert a listában megadott három IP címhez engedélyeztük.

Az `option` utasításban kikapcsoltuk a DNS rekurziót, így elválaszthatjuk egymástól a DNS kiszolgáló funkcionalitást a DNS gyorsítótár funkcionalitástól. Ezt biztonsági szempontból érdemes elvégezni. A témáról számos értekezést írtak, az egyik legjobb magyarázatot a *http://cr.yp.to/djbdns/separation.html* címen olvashatjuk. Ha szükségünk van a rekurzív funkcionalitásra, a legjobb megoldás, ha egy csak tároló kiszolgálót állítunk be a DNS kiszolgálónktól független IP címen. Nemsokára szólunk arról is, hogyan készíthetünk csak tároló kiszolgálót a BIND használatával.

Ha úgy döntünk, hogy a kiszolgálón rekurzió nélkül nem élet az élet, a *named.config* lehetőséget ad arra, hogy a rekurziót adott tartományokra korlátozzuk. Az `allow-recursion` opciót írva a `recursion no` helyett, megadhatjuk azoknak az IP címeknek a tartományát, melyek számára engedélyezzük a rekurzív kéréseket.

Valószínűleg mindenki észrevette, hogy az állományban a **vbrew.com** tartományon kívül is elhelyeztünk néhány bejegyzést. Jelenlétük a kompatibilitás miatt szükséges az RFC 1912 jelölésű szabvánnyal (Common DNS Operational and Configurational Errors). Ez a szabvány ugyanis előírja, hogy a hálózati helyek a localhost forward és reverse zónákhoz, illetve a broadcast zónákhoz legyenek meghatalmazottak.

A BIND konfiguráció jóval több lehetőséget támogat, mint amennyivel idáig foglalkoztunk. Ezekhez a legjobb forrás a BIND Version 8 és 9 forráscsomagjaiban található dokumentáció.

A DNS adatbázis állományok

A *named* program törzsállományaihoz, a zónaállományokhoz mindig társul egy tartomány, melyet *kezdőpontnak* (origin) hívunk. Ez a `cache` és `primary` parancsokkal megadott tartománynév. A törzsállományokban a tartománynevet és a gazdagépek nevét e tartományhoz viszonyítva is megadhatjuk. A konfigurációs állományban megadott név *abszolútnak* számít, ha egyetlen pontra végződik; minden más esetben a kezdőponthoz viszonyított. Magára a kezdőpontra a (@) jelöléssel utalhatunk.

A törzsállományokban foglalt adatokat a DNS-en keresztül elérhető legkisebb információs egységre, forrásrekordokra (RR) osztjuk. Minden forrásrekordnak van egy típusa. Az A típusú rekordok például a gazdagépneveket képzik le IP címekre, míg a CNAME rekordok egy gazdagép alias nevét társítják a hivatalos gazdagépnévvel. Erre példát is láthatunk, amikor a fejezet későbbi részében bemutatunk egy teljes konfigurációs és zónaállomány-garnitúrát.

A forrásrekordok ábrázolása a törzsállományokban azonos mintát követ:

```
[tartomány] [ttl] [osztály] típus rekordadat
```

A mezőket szóközök vagy tabulátorok választják el. Egy bejegyzés akár több soron keresztül is folytatódhat, amennyiben a nyitó zárójel az első újsor karakter előtt áll, és az utolsó mezőt záró zárójel követi. A pontosvessző és az újsor karakter között minden figyelmen kívül marad. Lássuk a kifejezések jelentését:

tartomány
A bejegyzés erre a tartománynévre vonatkozik. Ha nem adjuk meg, akkor feltételezzük a forrásrekordról, hogy az előző forrásrekord tartományára vonatkozik.

ttl
Azért, hogy a feloldók egy bizonyos idő után elvessék az információkat, minden forrásrekordnak megadjuk az élettartamát (*ttl*). A másodpercben megadott ttl mező határozza meg, hogy az információ meddig marad érvényben a kiszolgálóról való lekérést követően. A mező egy legfeljebb nyolc számjegyű decimális szám.
Ha nem adjuk meg, a mező értéke az alapértelmezés szerint azonos a megelőző SOA rekord *minimum* mezőjének értékével.

osztály
Olyan címosztály, amilyen az IN az IP címek esetén, vagy a HS a Hesiod osztály objektumai esetén. TCP/IP hálózatkezeléskor az értéke IN.
Ha nem adjuk meg, az előző forrásrekord osztályát feltételezzük.

típus
A forrásrekord típusát írja le. A leggyakoribb típusok az A, SOA, PTR és NS. A következő rész a forrásrekordok különböző típusaival foglalkozik.

rekordadat
A forrásrekordhoz tartozó adatokat tárolja. A mező formátuma a forrásrekord típusától függ. A következőkben különvesszük az egyes forrásrekordoknál.

Most a DNS törzsállományokban használható forrásrekordok egy részével ismerkedünk meg. Továbbiak is léteznek, de ezekkel nem foglalkozunk, mert kísérleti jellegűek, és nem sok hasznukat vehetjük.

SOA
A forrásrekord egy meghatalmazási zónát ír le, és jelzi, hogy az utána következő rekordok a tartományra vonatkozó meghatalmazási információkat tárolják. A `primary` utasítással megadott minden törzsállománynak tartalmaznia kell egy SOA rekordot a zónához. A forrásadatokban a következő mezőket találjuk:

kezdőpont
A mező a tartomány elsődleges névkiszolgálójának kanonikus neve. Általában abszolút névként adjuk meg.

kapcsolat

A mező a tartomány fenntartásáért felelős személy e-mail címe, a „@" helyett ponttal írva. Például ha a Virtuális Sörgyár felelőse **janet**, a mező tartalma **janet.vbrew.com**.

sorozatszám

A mező a zónaállomány verziószáma, egyetlen decimális számmal kifejezve. Valahányszor a zónaállomány adatai megváltoznak, a számot növelni kell. A szám hagyományosan az utolsó frissítés dátumát tartalmazza, amelyet a verziószám követ arra az esetre, ha egy nap több frissítés is lezajlana. A 2000012600 például a 2000. január 26-án történt 00 verziószámú frissítésre utal.

A sorozatszámot a másodlagos névkiszolgálók használják a zónainformáció változásának figyelésére. Azért, hogy mindig napra készek maradjanak, a másodlagos kiszolgálók bizonyos időközönként lekérik az elsődleges kiszolgáló SOA rekordját, és a sorozatszámot összevetik az általuk tárolt SOA rekord sorozatszámával. Amennyiben a szám megváltozott, a másodlagos kiszolgáló a teljes zónaadatbázist átveszi az elsődleges kiszolgálótól.

frissítés

A mező azt szabályozza, hogy milyen időnként ellenőrizzék a másodlagos kiszolgálók az elsődleges kiszolgáló SOA rekordját. Az értékét másodpercben adjuk meg. Szintén decimális szám, és legfeljebb nyolc számjegyből állhat.

A hálózati topológia általában csak ritkán változik, ezért nagyobb hálózatokon érdemes körülbelül egynapos időtartamot megadni, míg kisebb hálózatokon ez hosszabb is lehet.

ismételt próbálkozás

Ez a szám határozza meg, hogy mennyit várakozik a másodlagos kiszolgáló, mielőtt ismételten megpróbálja elérni az elsődleges kiszolgálót, ha egy kérés vagy egy zónafrissítés kudarcot vallott. Az érték ne legyen túl alacsony, különben a kiszolgáló átmeneti meghibásodása vagy egy hálózati hiba miatt a másodlagos kiszolgáló értékes hálózati erőforrásokat veszteget el. Egy óra, esetleg fél óra jó választás lehet.

lejárat

A mező másodpercben megadott értéke szabja meg, hogy a másodlagos kiszolgáló mennyi idő múltán veti el véglegesen a zónaadatokat, ha nem volt képes felvenni a kapcsolatot az elsődleges kiszolgálóval. Értékét általában legalább egy hétre (604 800 másodperc) állítjuk, de ésszerű lehet akár egy hónapra vagy hosszabb időre növelni.

minimum

A mező a forrásrekordok alapértelmezett élettartamát határozza meg arra az esetre, ha egy forrásrekordban nincs külön megadva. A ttl érték a többi névkiszolgáló számára szabja meg, mennyi ideig tárolhatják legfeljebb a forrásrekordot a gyorsítótárukban. Az időtartam csak a normál kikeresésekre vonatkozik, semmi

köze ahhoz az időtartamhoz, ameddig a másodlagos kiszolgáló vár a zónainformáció frissítésével.

Ha hálózatunk topológiája csak ritkán változik, egy hét vagy ennél több lehet jó választás. Ha az egyes forrásrekordok gyakrabban változnak, ttl idejüket egyenként adhatjuk meg. Ha a hálózat változik gyakrabban, a `minimum` értékét beállíthatjuk egy napra (86 400 másodperc).

A

A rekord egy IP címet társít egy gazdagépnévvel. A forrásadatmező a pontozott négyes jelöléssel megadott címet tartalmazza.

Minden gazdagépnévhez tartozik egy A rekord. Az ebben megadott gazdagépnév a *kanonikus* gazdagépnév. Minden más gazdagépnév hivatkozási név (alias), amelyet a kanonikus gazdagépnévre képzünk le egy CNAME rekorddal. Ha gazdagépünk kanonikus gazdagépneve **vlager**, akkor szükségünk van egy A rekordra, amely ezt a nevet a hozzá tartozó IP címhez kapcsolja. De szeretnénk, hogy a címhez egy további név is tartozzon, mondjuk a **news**, ezért készítünk egy CNAME rekordot, amely az alternatív nevet a kanonikus névhez fűzi. A CNAME rekordokkal rövidesen közelebbről is megismerkedünk.

AAAA

Ez a rekord teljesen azonos az A rekorddal, de kizárólag IPv6 címekhez használjuk.

NS

Az NS rekordok feladata a zóna elsődleges és valamennyi másodlagos kiszolgálójának kijelölése. Egy NS rekord az adott zóna valamely fő névkiszolgálójára mutat; a forrásadatmező a névkiszolgáló gazdagépnevét tartalmazza.

NS rekordokat jellemzően két esetben használunk: amikor a meghatalmazást egy alárendelt zónának adjuk át, és amikor magában az alárendelt zóna fő zónaadatbázisában adjuk meg. A szülő és a delegált zónában megadott kiszolgálóknak egyezniük kell.

Az NS rekord a zóna elsődleges és másodlagos névkiszolgálóinak nevét határozza meg. Ahhoz, hogy ezeket használhassuk, át kell őket alakítanunk címekre. Mivel esetenként a kiszolgálók a kiszolgált tartományhoz tartoznak, felmerül a „tyúk vagy a tojás" kérdése; a nevet mindaddig nem oldhatjuk fel, amíg a névkiszolgáló nem elérhető, ugyanakkor amíg a címet fel nem oldjuk, nem érhetjük el a névkiszolgálót. A megoldáshoz különleges A rekordokat állítunk be, közvetlenül a szülő zóna névkiszolgálójában. Az A rekordok lehetővé teszik a szülő tartomány névkiszolgálói számára, hogy feloldják a delegált zóna névkiszolgálóinak IP címét. Ezeket a rekordokat gyakran ragasztó rekordoknak is nevezik, mert „összetartják" a delegált zónát és a szülő zónát; működésüket a korábbi, *DNS adatbázis* című részből ismerhetjük meg.

CNAME

A rekord egy gazdagép kanonikus gazdagépnevéhez társít egy alias nevet. A felhasználók a hivatkozási névvel is utalhatnak arra a gazdagépre, melynek kanonikus nevét paraméterként megadtuk. A kanonikus gazdagépnév az a név, amelyhez a törzs-

állományban tartozik egy A rekord; az alias neveket CNAME rekordok kapcsolják e névhez, de más saját rekorddal nem rendelkeznek.

PTR

Ez a fajta rekord az **in-addr.arpa** tartományban társít neveket a gazdagépnevekhez. Fordított feloldásra használjuk, IP címek átalakítására gazdagépnevekké. A megadott névnek kanonikus gazdagépnévnek kell lennie.

MX

A forrásrekord *levélváltót* (mail exchanger, MX) jelöl ki egy tartomány részére. A levélváltókról a 11. fejezet *Levelezési útválasztás az interneten* című részében lesz szó. Az MX rekord szintaxisa:

```
[tartomány] [ttl] [osztály] MX preferencia gazdagép
```

A *gazdagép* a tartományhoz kijelölt levélváltó. Minden levélváltóhoz tartozik egy egész szám, a *preferencia*. A levélközvetítő program, amikor egy levelet szeretne a *tartomány*ba kézbesíteni, az adott tartomány MX bejegyzéssel rendelkező összes gazdagépét végigpróbálja, amíg sikerrel nem jár. Először a legkisebb preferenciaértékkel próbálkozik, majd sorban a többit is megvizsgálja növekvő preferenciaérték szerint.

HINFO

A rekord a rendszer hardveréről és szoftveréről nyújt információt. Szintaxisa:

```
[tartomány] [ttl] [osztály] HINFO hardver szoftver
```

A *hardver* mező a gazdagép hardverét azonosítja. A mező formátuma különleges szabályokat követ: ha a mezőben szóköz fordul elő, idézőjelek között kell megadnunk. A *szoftver* mező a rendszer operációs rendszerét nevezi meg. Biztonsági okokból azonban jobb, ha ezt az felvilágosítást nem tesszük nyilvánossá, mert a támadóknak értékes információt jelenthet. Jelenleg csak nagy ritkán adják meg, noha vannak rendszergazdák, akik szívesen használják, és valótlan vagy humoros információt helyeznek el benne.

Alább a számítógépek leírására szolgáló minta HINFO rekordot láthatunk:

```
tao   36500  IN  HINFO  SEGA-DREAMCAST  LINUX2.6
cevad 36500 IN  HINFO  ATARI-104ST LINUX2.0
jedd  36500 IN  HINFO  TIMEX-SINCLAIR  LINUX2.6
```

A named csak tároló konfigurációja

Mielőtt egy teljes névkiszolgáló-konfigurációt összeállítanánk, beszéljünk még a *named* konfiguráció egy különleges fajtájáról, melynek neve *csak tároló* konfiguráció. Valójában nem egy tartomány kiszolgálására használjuk, hanem gazdagépünk DNS kéréseinek kezelésére. Az eljárás előnye, hogy átmeneti tárolót hoz létre, így egy adott gazdagépre vonatkozó lekérdezést csak az első alkalommal szükséges az internetes névkiszolgálókhoz küldenünk. Az ismétlődő lekérdezésekre a válasz már közvetlenül a helyi név-

kiszolgáló gyorsítótárából érkezik. Még egy fontos figyelmeztetés: a tároló kiszolgálót lehetőség szerint ne futtassuk ugyanarról az IP címről, mint a DNS kiszolgálót!

Egy csak tároló kiszolgáló *named.conf* állománya így néz ki:

```
// A lekérdezésre jogosult hálózatok megadása.

acl allowednets { 192.168.99.0/24; 192.168.44.0/24; };
options {
     directory "/etc/bind";           // Working directory
     allow-query { allowednets; };
};

// A gyökér kiszolgálókat tartalmazó lista
zone "." { type hint;
        file "root.hint"; };

// A 127.0.0.1 visszacsatoló cím fordított leképzésének biztosítása
zone "0.0.127.in-addr.arpa" {
     type master;
     file "localhost.rev";
     notify no;
};
```

A *named.conf* állomány mellett be kell állítanunk a *root.hint* állományt is, amely a gyökér névkiszolgálók érvényes listáját tartalmazza. Erre a célra kimásolhatjuk a 6.10. példa állományát, vagy ami még jobb, beszerezhetjük az állomány legújabb verzióját a *dig* nevű BIND eszközzel. A csak tároló kiszolgálók beállításához mindössze ez a két állomány szükséges. A *dig* programról részletesen szólunk még a fejezetben, egyelőre legyen elég annyi, hogy a következő paranccsal kérhetjük le a *root.hint* állomány elkészítéséhez szükséges információt:

```
vbrew# dig at e.root-servers.net . ns
```

A törzsállományok elkészítése

Az 5.6., 5.7., 5.8. és 5.9. példák a **vlager** gazdagép minta konfigurációs és zónaállományait tartalmazzák a sörgyár egy névkiszolgálója számára. A tárgyalt hálózat (egyetlen LAN) természetéből adódóan a példa könnyen érthető.

Az 5.6. példa *root.hint* átmeneti tároló állományában minta útbaigazító (hint) rekordokat láthatunk egy gyökér névkiszolgáló számára. Egy tipikus átmeneti tároló állomány rendszerint egy tucat névkiszolgáló leírását tartalmazza. A gyökértartomány névkiszolgálóinak aktuális listáját a *dig* programmal, a fent bemutatott módon szerezhetjük be.

5.6. példa. A root.hint állomány

```
; <<>> DiG 9.2.2 <<>> at e.root-servers.net . ns
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21972
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
```

5.6. példa. A root.hint állomány (folytatás)

```
;; QUESTION SECTION:
;at.                            IN      A

;; AUTHORITY SECTION:

;; Query time: 54 msec
;; SERVER: 206.13.28.12#53(206.13.28.12)
;; WHEN: Sat Jan 31 11:28:44 2004
;; MSG SIZE  rcvd: 83

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8039
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
;e.root-servers.net.             IN      A

;; ANSWER SECTION:
e.root-servers.net.     566162  IN      A       192.203.230.10

;; AUTHORITY SECTION:
ROOT-SERVERS.net.       134198  IN      NS      a.ROOT-SERVERS.net.
ROOT-SERVERS.net.       134198  IN      NS      f.ROOT-SERVERS.net.
ROOT-SERVERS.net.       134198  IN      NS      j.ROOT-SERVERS.net.
ROOT-SERVERS.net.       134198  IN      NS      k.ROOT-SERVERS.net.

;; ADDITIONAL SECTION:
a.ROOT-SERVERS.net.     566162  IN      A       198.41.0.4
f.ROOT-SERVERS.net.     566162  IN      A       192.5.5.241
j.ROOT-SERVERS.net.     566162  IN      A       192.58.128.30
k.ROOT-SERVERS.net.     566162  IN      A       193.0.14.129

;; Query time: 12 msec
;; SERVER: 206.13.28.12#53(206.13.28.12)
;; WHEN: Sat Jan 31 11:28:44 2004
;; MSG SIZE  rcvd: 196

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61551
;; flags: qr rd ra; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 13

;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       479762  IN      NS      F.ROOT-SERVERS.NET.
.                       479762  IN      NS      B.ROOT-SERVERS.NET.
.                       479762  IN      NS      J.ROOT-SERVERS.NET.
.                       479762  IN      NS      K.ROOT-SERVERS.NET.
.                       479762  IN      NS      L.ROOT-SERVERS.NET.
.                       479762  IN      NS      M.ROOT-SERVERS.NET.
.                       479762  IN      NS      I.ROOT-SERVERS.NET.
.                       479762  IN      NS      E.ROOT-SERVERS.NET.
```

5.6. példa. A root.hint állomány (folytatás)

```
.                       479762  IN      NS      D.ROOT-SERVERS.NET.
.                       479762  IN      NS      A.ROOT-SERVERS.NET.
.                       479762  IN      NS      H.ROOT-SERVERS.NET.
.                       479762  IN      NS      C.ROOT-SERVERS.NET.
.                       479762  IN      NS      G.ROOT-SERVERS.NET.

;; ADDITIONAL SECTION:
F.ROOT-SERVERS.NET.     566162  IN      A       192.5.5.241
B.ROOT-SERVERS.NET.     566162  IN      A       192.228.79.201
J.ROOT-SERVERS.NET.     566162  IN      A       192.58.128.30
K.ROOT-SERVERS.NET.     566162  IN      A       193.0.14.129
L.ROOT-SERVERS.NET.     566162  IN      A       198.32.64.12
M.ROOT-SERVERS.NET.     566162  IN      A       202.12.27.33
I.ROOT-SERVERS.NET.     566162  IN      A       192.36.148.17
E.ROOT-SERVERS.NET.     566162  IN      A       192.203.230.10
D.ROOT-SERVERS.NET.     566162  IN      A       128.8.10.90
A.ROOT-SERVERS.NET.     566162  IN      A       198.41.0.4
H.ROOT-SERVERS.NET.     566162  IN      A       128.63.2.53
C.ROOT-SERVERS.NET.     566162  IN      A       192.33.4.12
G.ROOT-SERVERS.NET.     566162  IN      A       192.112.36.4

;; Query time: 17 msec
;; SERVER: 206.13.28.12#53(206.13.28.12)
;; WHEN: Sat Jan 31 11:28:44 2004
;; MSG SIZE  rcvd: 436
```

5.7. példa. A vbrew.com zónaállomány

```
;
;                    A sörgyár gazdagépei
;                 /etc/bind/zone/vbrew.com
;                           A kezdőpont: vbrew.com
;
$TTL 3D
@               IN  SOA   vlager.vbrew.com. janet.vbrew.com. (
                          200401206       ; sorozatszám, dátum + a napi sorszám
                          8H              ; frissítés, másodperc
                          2H              ; ismételt próbálkozás, másodperc
                          4W              ; lejárat, másodperc
                          1D )            ; minimum, másodperc

                IN  NS    vlager.vbrew.com.
;
; a helyi levelek szétosztása a vlager gépen
                IN  MX    10 vlager
;
; a visszacsatolási cím
localhost.      IN  A     127.0.0.1
;
; Virtuális Sörgyár Ethernet
vlager          IN  A     172.16.1.1
vlager-if1      IN  CNAME vlager
```

5.7. példa. A vbrew.com zónaállomány (folytatás)

```
; A vlager egyben hírkiszolgáló is
news            IN  CNAME vlager
vstout          IN  A     172.16.1.2
vale            IN  A     172.16.1.3
;
; Virtuális Borászat Ethernet
vlager-if2      IN  A     172.16.2.1
vbardolino      IN  A     172.16.2.2
vchianti        IN  A     172.16.2.3
vbeaujolais     IN  A     172.16.2.4
;
; Virtuális Szeszfőzde (alhálózat) Ethernet
vbourbon        IN  A     172.16.3.1
vbourbon-if1    IN  CNAME vbourbon
```

5.8. példa. A visszacsatoló zónaállomány

```
;
; /etc/bind/zone/127.0.0       a 127.0.0 fordított leképzése
;                             Kezdőpont: 0.0.127.in-addr.arpa.
;
$TTL 3D
@           IN  SOA   vlager.vbrew.com. joe.vbrew.com. (
                      1          ; sorozatszám
                      360000     ; frissítés: 100 óra
                      3600       ; ismételt próbálkozás: egy óra
                      3600000    ; lejárat: 42 nap
                      360000     ; minimum: 100 óra
                      )
            IN  NS    vlager.vbrew.com.
1           IN  PTR   localhost.
```

5.9. példa. A vbrew fordított kikeresési állomány

```
;
;                   /etc/bind/zones/16.172.in-addr.arpa
;                            Kezdőpont: 16.172.in-addr.arpa.
;
$TTL 3D
@           IN  SOA   vlager.vbrew.com. joe.vbrew.com. (
                       16         ; sorozatszám
                       86400      ; frissítés: naponta egyszer
                       3600       ; ismételt próbálkozás: egy óra
                       3600000    ; lejárat:  42 nap
                       604800     ; minimum: 1 hét
                       )
            IN  NS    vlager.vbrew.com.
; sörgyár
1.1         IN  PTR   vlager.vbrew.com.
2.1         IN  PTR   vstout.vbrew.com.
```

5.9. példa. A vbrew fordított kikeresési állomány (folytatás)

```
3.1             IN  PTR   vale.vbrew.com.
; borászat
1.2             IN  PTR   vlager-if2.vbrew.com.
2.2             IN  PTR   vbardolino.vbrew.com.
3.2             IN  PTR   vchianti.vbrew.com.
4.2             IN  PTR   vbeaujolais.vbrew.com.
```

A névkiszolgáló beállításának ellenőrzése

A névkiszolgálók lekérdezésére jelenleg előszeretettel használják a dig programot, mely a régebbi, jól ismert nslookup helyére lépett. A dig rugalmas, gyors és szinte bármit lekérdezhetünk vele egy DNS kiszolgálóról. Szintaxisa egyszerű:

```
vbrew# dig névkiszolgáló név típus
```

A parancssori paraméterek definíciója:

névkiszolgáló
: Annak a kiszolgálónak a neve, amelyhez a lekérdezést intézzük. Megadhatjuk névvel vagy IP címmel. Ha a mezőt üresen hagyjuk, a dig a *resolv.conf* állományban megadott DNS kiszolgálóhoz fordul.

név
: A keresendő DNS rekord neve.

típus
: A végrehajtani kívánt lekérdezés típusa. Gyakori típusok az ANY, az MX és a TXT. Ha a mezőt üresen hagyjuk, a *dig* alapértelmezés szerint egy A rekordot keres.

Következésképpen ha szeretnénk megtudni, mely kiszolgálók kezelik a Virtuális Sörgyár levelezését, a következő lekérdezést futtatjuk:

```
vbrew# dig vlager.vbrew.com MX
; <<>> DiG 9.2.2 <<>> vlager.vbrew.com MX
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40590
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0

;; QUESTION SECTION:
;vlager.vbrew.com.      IN      MX

;; AUTHORITY SECTION:
vbrew.com.          10794   IN      SOA     vlager.vbrew.com. vlager.vbrew.-
com. 2003080803 10800 3600 604800 86400

;; Query time: 14 msec
```

```
;; SERVER: 192.168.28.12#53(192.168.28.12)
;; WHEN: Sun Feb  1 12:19:06 2004
;; MSG SIZE  rcvd: 104
```

Szintén hasznos és érdekes lekérdezési típus a BIND verzió lekérdezése. A következő szintaxissal a *dig* számára ez sem jelent gondot:

```
vlager# dig @vlager.vbrew.com version.bind. CHAOS TXT
.
.
; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;version.bind.                  CH      TXT

;; ANSWER SECTION:
VERSION.BIND.           0       CH      TXT     "BIND 8.2.2-P5"
.
.
```

Az nslookup alkalmazása

Az `nslookup`, noha alkalmazását ma már nem javasoljuk, megfelelő segédeszköz lehet a névkiszolgáló beállításának ellenőrzésére. Futtathatjuk interaktív üzemmódban, amikor futását a promptból irányítjuk, de önálló parancsként is, ekkor azonnal megkapjuk az eredményt. Ez utóbbi esetben így hívjuk meg:

```
$ nslookup gazdagépnév
```

Az `nslookup` a *gazdagépnév* után kutatva lekérdezi a *resolv.conf* állományban megadott névkiszolgálót. (Ha az állomány több kiszolgálót is megjelöl, az `nslookup` véletlenszerűen választ ki egyet.)

Az interaktív üzemmód azonban jóval izgalmasabb. Nem csak önálló gazdagépeket kereshetünk ki, de bármilyen típusú DNS rekordra rákérdezhetünk, és egy tartomány teljes zónainformációját átmásolhatjuk.

Az *nslookup* argumentumok nélküli meghívása megjeleníti az általa használt névkiszolgálót, és interaktív üzemmódba lép. A parancskérő jelnél beírhatunk egy tetszőleges tartománynevet, melyet le szeretnénk kérdezni. Alapértelmezés szerint a program A osztályú rekordokat keres - olyanokat, melyek a tartománynévhez kapcsolódó IP címeket tartalmaznak.

A keresett rekord típusát így adhatjuk meg:

```
> set type=típus
```

ahol a *típus* a korábban ismertetett forrásrekord nevek valamelyike, vagy `ANY`. Elképzelhető például a következő *nslookup* művelet:

```
$ nslookup
Default Server:  tao.linux.org.au
Address:  203.41.101.121

> metalab.unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

Name:    metalab.unc.edu
Address:  152.2.254.81

>
```

A kimenetben először a lekérdezett DNS kiszolgálót láthatjuk, ezt követi a lekérdezés eredménye.

Ha olyan nevet kérdezünk le, amelyhez nem tartozik IP cím, de más rekordokat találunk a DNS adatbázisban, az `nslookup` hibaüzenettel tér vissza: „No type A records found" (A típusú rekord nem található). A *set type* utasítással azonban rávehetjük, hogy ne csak A típusú rekordokat keressen. Az **unc.edu** tartomány SOA rekordjait például így kérhetjük le:

```
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

*** No address (A) records available for unc.edu
> set type=SOA
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121

unc.edu
        origin = ns.unc.edu
        mail addr = host-reg.ns.unc.edu
        serial = 1998111011
        refresh = 14400 (4H)
        retry   = 3600 (1H)
        expire  = 1209600 (2W)
        minimum ttl = 86400 (1D)
unc.edu name server = ns2.unc.edu
unc.edu name server = ncnoc.ncren.net
unc.edu name server = ns.unc.edu
ns2.unc.edu     internet address = 152.2.253.100
ncnoc.ncren.net internet address = 192.101.21.1
ncnoc.ncren.net internet address = 128.109.193.1
ns.unc.edu      internet address = 152.2.21.1
```

Hasonlóképpen kérhetjük le az MX rekordokat:

```
> set type=MX
> unc.edu
Server:  tao.linux.org.au
Address:  203.41.101.121
```

```
unc.edu preference = 0, mail exchanger = conga.oit.unc.edu
unc.edu preference = 10, mail exchanger = imsety.oit.unc.edu
unc.edu name server = ns.unc.edu
unc.edu name server = ns2.unc.edu
unc.edu name server = ncnoc.ncren.net
conga.oit.unc.edu       internet address = 152.2.22.21
imsety.oit.unc.edu      internet address = 152.2.21.99
ns.unc.edu      internet address = 152.2.21.1
ns2.unc.edu     internet address = 152.2.253.100
ncnoc.ncren.net internet address = 192.101.21.1
ncnoc.ncren.net internet address = 128.109.193.1
```

Az ANY típust beállítva minden, a megadott névhez tartozó forrásrekordot megkapunk. Az **nslookup** feladata a gyakorlatban - a hibakeresés mellett - a gyökér névkiszolgálók aktuális listájának beszerzése lehet. A listát a gyökértartományhoz kapcsolódó összes NS rekord lekérdezésével kaphatjuk meg:

```
> set type=NS
> .
Server:  tao.linux.org.au
Address:  203.41.101.121

Non-authoritative answer:
(root)  name server = A.ROOT-SERVERS.NET
(root)  name server = H.ROOT-SERVERS.NET
(root)  name server = B.ROOT-SERVERS.NET
(root)  name server = C.ROOT-SERVERS.NET
(root)  name server = D.ROOT-SERVERS.NET
(root)  name server = E.ROOT-SERVERS.NET
(root)  name server = I.ROOT-SERVERS.NET
(root)  name server = F.ROOT-SERVERS.NET
(root)  name server = G.ROOT-SERVERS.NET
(root)  name server = J.ROOT-SERVERS.NET
(root)  name server = K.ROOT-SERVERS.NET
(root)  name server = L.ROOT-SERVERS.NET
(root)  name server = M.ROOT-SERVERS.NET

Authoritative answers can be found from:
A.ROOT-SERVERS.NET      internet address = 198.41.0.4
H.ROOT-SERVERS.NET      internet address = 128.63.2.53
B.ROOT-SERVERS.NET      internet address = 128.9.0.107
C.ROOT-SERVERS.NET      internet address = 192.33.4.12
D.ROOT-SERVERS.NET      internet address = 128.8.10.90
E.ROOT-SERVERS.NET      internet address = 192.203.230.10
I.ROOT-SERVERS.NET      internet address = 192.36.148.17
F.ROOT-SERVERS.NET      internet address = 192.5.5.241
G.ROOT-SERVERS.NET      internet address = 192.112.36.4
J.ROOT-SERVERS.NET      internet address = 198.41.0.10
K.ROOT-SERVERS.NET      internet address = 193.0.14.129
L.ROOT-SERVERS.NET      internet address = 198.32.64.12
M.ROOT-SERVERS.NET      internet address = 202.12.27.33
```

Az elérhető parancsok teljes listájának megtekintéséhez adjuk ki a *help* parancsot az **nslookup** programban.

Egyéb hasznos eszközök

BIND adminisztrátorként további hasznos eszközök is rendelkezésünkre állnak. Ezek közül kettővel röviden megismerkedünk. Alkalmazásukról további információt a hozzájuk mellékelt dokumentációban találhatunk.

A *hostcvt* állomány a BIND kezdeti beállításában segíthet: */etc/hosts* állományunkat a *named* számára törzsállományokká alakítja. Az előre (A) és a visszafelé (PTR) történő leképzések bejegyzéseit is létrehozza, és az alias nevekről is gondoskodik. Természetesen nem végezhet el helyettünk mindent, hiszen előfordulhat például, hogy meg szeretnénk változtatni az SOA rekord időtúllépési értékeit, vagy MX rekordokat szeretnénk felvenni. Mindazonáltal megspórolhat nekünk néhány aszpirint. A *hostcvt* a BIND forrásának része, de önálló csomagként is létezik.

A névkiszolgáló beállítása után tesztelhetjük a konfigurációt. Lássunk egypár eszközt munkánk megkönnyítésére; az első neve *dnswalk,* egy Perl alapú csomag. A második neve *nslint.* Mindkettő a DNS adatbázist vizsgálja át gyakran előforduló hibákat keresve, de azt is ellenőrzi, hogy az információ konzisztens-e. További hasznos segédeszköz a *host*, amely egy általános célú DNS adatbázis-lekérdező eszköz. Az eszközzel manuálisan vizsgálhatjuk és diagnosztizálhatjuk a DNS adatbázis bejegyzéseit.

Az eszköz valószínűleg elérhető előre csomagolt formában. A *dnswalk* és *nslint* forráskódja letölthető a *http://www.visi.com/~barr/dnswalk/* és az *ftp://ftp.ee.lbl.gov/nslint.tar.Z* címről. A *host* forráskódja az *ftp://ftp.nikhef.nl/pub/network/* és az *ftp://ftp.is.co.za/networking/ip/dns/dig/* címekről.

A BIND alternatívái

Azok, akik aggódnak a BIND kiszolgáló évek során felfedezett számos biztonsági hiányossága miatt vagy egyszerűbb DNS megoldásra vágynak, vethetnek egy pillantást az egyik alternatívára, a djbdns programra. A szoftver, melyet a semmiből hozott létre D. J. Bernstein, megbízhatóbb, egyszerűbb és biztonságosabb DNS szerkezetet kínál. A djbdns telepítése és beállítása könnyű, és noha funkcionalitásában alapvetően megegyezik a BIND-dal, kevésbé összetett. A következő részben megnézzük, hogyan telepíthetünk és állíthatunk be egy DNS kiszolgálót a djbdns segítségével. Fontos megjegyeznünk, hogy a djbdns DNS kiszolgálót annak tervezték, ami, vagyis DNS kiszolgálónak, ezért alapértelmezés szerint a meghatalmazásunkon kívül eső gépekre vonatkozó lekérdezések feloldására nem alkalmas. Ehhez egy külön tároló kiszolgálót kell építenünk, egy önálló gépen vagy önálló IP címmel. Szó volt már róla, hogy a tárolókat és DNS kiszolgálókat biztonsági okokból érdemes elkülöníteni. Erről többet is megtudhatunk a djbdns webhelyén, a *http://cr.yp.to/djbdns.html* címen.

A djbdns telepítése

A djbdns futtatásához először telepítenünk kell egy további DJB programot, melynek neve daemontools. Ez lényegében eszközök gyűjteménye a különféle Unix daemonok

kezelésére. A daemontools dokumentációját és forráskódját megtekinthetjük a weboldalán, a *http://cr.yp.to/daemontools.html* címen. Sikeres letöltése után bontsuk ki a szoftvert egy könyvtárba és fordítsuk le. A daemontools mellett találunk egy szkriptet, mely automatikusan lefordítja és telepíti a programot. A következőképpen indíthatjuk el:

```
vlager# mkdir software
vlager# cd software
vlager# tar xzpf daemontools-0.76.tar.gz
vlager# cd admin/daemontools-0.76
vlager# package/install
```

A szkript futását követően eltávolíthatjuk a telepítési könyvtárakat, és hozzákezdhetünk a soron következő összetevő telepítéséhez. Ez a ucspi-tcp, a DJB saját TCP ügyfél/kiszolgáló programja, melyet ugyancsak egyszerűen telepíthetünk:

```
vlager# mkdir software
vlager# cd software
vlager# tar xzpf ucspi-tcp-0.88.tar.gz
vlager# cd ucspi-tcp-0.88
vlager# make
vlager# make setup check
```

A kód a programot gépünk */usr/local* könyvtárába telepíti. A program működtetéséhez és beállításához egyelőre mást nem kell tennünk.

Most már készen állunk a djbdns üzembe állítására. A djbdns telepítését a `ucspi-tcp` telepítésének fent leírt lépéseivel végezhetjük el. A művelet a djbdns programot a */usr/local* könyvtárba helyezi. A djbdns beállítása előtt győződjünk meg arról, hogy az `svscan` folyamat fut. Az `svscan` a `daemontools` csomag része, és futása elengedhetetlen a djbdns működéséhez.

Miután ellenőriztük, hogy az *svscan* fut, megkezdhetjük a DNS kiszolgáló beállítását. Mindenekelőtt hozzunk létre két felhasználói fiókot (account) *tinydns* és *dnslog* néven. A djbdns e két fiókot használja a működéséhez, és nem rootként fut, mint sok esetben a BIND installáció.

Hozzunk most létre egy könyvtárat a DNS kiszolgáló konfigurációs és naplóállományai számára, majd állítsuk be a következők szerint:

```
vlager# mkdir /etc/tinydns
vlager# tinydns-conf tinydns dnslog /etc/tinydns 172.16.0.2
```

A példa 172.16.0.2 IP címe helyére írjuk saját DNS kiszolgálónk külső IP címét. Ezt követően tájékoztatnunk kell az *svscan* programot az új szolgáltatásról:

```
vlager# ln -s /etc/tinydns /service
vlager# svstat /service/tinydns
```

Ezzel befejeztük djbdns kiszolgálónk telepítését; csak az maradt hátra, hogy beállítsuk a gazdagépeinket. A BIND esetén még csak most jönne a bonyolult és zavarba ejtő feladatok többsége; a djbdns azonban leegyszerűsíti az új DNS rekordok létrehozását.

Gazdagépek felvétele

Azért, hogy a DNS kiszolgáló a szolgáltatást biztosítani tudja, be kell állítanunk a gazdagépek adatait. Először meg kell határoznunk azt a tartományt, amelyre meghatalmazásunk érvényes. A Virtuális Sörgyár esetén például úgy szeretnénk beállítani a DNS kiszolgálót, hogy a **vbrew.com** tartományra vonatkozó lekérdezéseket kezelje. A hosszú zónaállományokkal való birkózás helyett elegendő néhány rövid lépés.

```
vlager# cd /service/tinydns/root
vlager# ./add-ns vlager.com 172.16.1.1
vlager# ./add-ns 1.16.172.in-addr.arpa 172.16.1.1
vlager# make
```

Miután beállítottuk kiszolgálónkat a **vbrew** tartományra vonatkozó lekérdezések kezelésére, felhasználhatjuk hálózatunk gazdagépeinek beállítására is. Szerencsénkre ez sem bonyolultabb, mint az előző lépés. Ahhoz, hogy egy címet társítsunk kedvenc **vlager** kiszolgálónkhoz, valamint webkiszolgálónkhoz, a következő utasításokat adjuk ki:

```
vlager# cd /service/tinydns/root
vlager# ./add-host vlager.vbrew.com 172.16.1.10
vlager# ./add-host www.vlager.com 172.16.1.11
vlager# ./add-alias mail.vbrew.com 172.16.1.10
vlager# make
```

A DNS rekord létrehozásához az *add-host* parancsot használjuk, ahol megadjuk az FQDN nevet és az IP címet. Feltűnhetett a példa másik parancsa is, az *add-alias*. Az utasítás egy már kiosztott IP címhez hoz létre hivatkozási nevet. A példában a **vlager** gazdagép a **mail** névre is válaszol. Ez többcélú kiszolgálók esetén lehet hasznos. Vegyük észre a *make* parancsot az utasítások végén! Ha elfelejtjük kiadni, a kiszolgáló nem fog működni, mert ez a parancs felelős a nyers konfigurációs állomány fordításáért a kiszolgáló számára érthető formában. Ha hibát tapasztalunk telepítés közben, először ezt ellenőrizzük!

Az *add-host*, *add-ns* és *add-alias* utasítások csupán a djbdns fő konfigurációs állományát módosítják. Az állomány neve *data*, és a */service/tinydns/root* könyvtárban található. Az előbbi műveleteket manuálisan is elvégezhetjük: nyissuk meg az adatállományt a böngészőnkben és írjuk be a következő sorokat:

```
=vlager.vbrew.com:172.16.1.10
=www.vlager.com:172.16.1.11
+mail.vbrew.com:172.16.1.10
```

Vegyük észre, hogy a gazdagép bejegyzésének sora = jellel kezdődik, az alias név sora pedig + karakterrel. Noha az adatokat kézzel is beírhatjuk, a helyzetet bonyolítja, hogy nekünk kell ellenőriznünk az adatállományban, nem létezik-e két azonos bejegyzés. Az egyszerűség kedvéért általában az automatizált eszközökhöz ragaszkodunk.

Külső DNS gyorsítótár telepítése

A DNS kiszolgáló sikeres beállítását és üzembe helyezését követően létrehozhatunk egy külső DNS tárat, hogy a hálózat gazdagépei a külső számítógépek IP címeit is feloldhassák. Ehhez egy DNS gyorsítótár telepítése szükséges, ami a djbdns segítségével ismét csak gyerekjáték. Feltéve, hogy az *svscan* fut, először létrehozunk két rendszerfiókot (illetve megnézzük, hogy már léteznek-e), az egyiket a tárolásról gondoskodó program, a másikat a naplózó rendszer számára. Noha nem kötelező, ajánlatos jelentéssel bíró nevet adnunk nekik, például *dnscache* és *dnslog*.

Ezt követően meghatározzuk azt az IP címet, melyen a DNS gyorsítótárat futtatni kívánjuk. Ne feledjük, érdemes a DNS kiszolgáló címétől eltérő címet választanunk. Root felhasználóként készítsünk egy könyvtárt a DNS szolgáltatás részére, és végezzük el a beállítását a következő utasításokkal:

```
vstout# mkdir /etc/dnscache
vstout# dnscache-conf dnscache dnslog /etc/dnscache <cache.ip.address>
```

Ismét root felhasználóként tájékoztassuk az svscan programot, hogy új szolgáltatást hoztunk létre, melyet futtatnia kell:

```
vstout# ln -s /etc/dnscache /service
```

Győződjünk meg róla, hogy az új szolgáltatás fut: várjunk néhány másodpercet, majd adjuk ki a következő utasítást:

```
vstout# svstat /service/dnscache
/service/dnscache: up (pid 1139) 149 seconds
```

Miután láttuk, hogy a szolgáltatás rendben fut, megadhatjuk, mely IP címek jogosultak a gyorsítótár elérésére. A Virtuális Sörgyár esetén a jogosultságot a teljes **172.16** hálózat számára szeretnénk megadni, ezért az utasítás így fest:

```
vstout# touch /etc/dnscache/root/ip/172.16
```

Persze szeretnénk biztosan tudni, hogy az */etc/resolv.conf* állomány értesült az új DNS gyorsítótárról. A gyorsítótár működését megvizsgálhatjuk az `nslookup`, a *dig* vagy a mellékelten kapott djbdns eszköz, a *dnsip* segítségével:

```
vlager# dnsip www.google.com
216.239.57.104 216.239.57.99
vlager#
```

6

A Point-to-Point protokoll

A Point-to-Point protokoll (PPP) alkalmas datagramok közvetítésére soros kapcsolaton keresztül. Ebben a fejezetben röviden átnézzük a protokoll alapvető építőelemeit. Foglalkozunk még a PPP Ethernet hálózaton történő alkalmazásával (PPP over Ethernet, PPPoE), amit a távközlési vállalatok ma már széles körben használnak DSL kapcsolatok létesítésére. A témával Andrew Sun *Using & Managing PPP* című kötete részletesen foglalkozik (kiadója az O'Reilly).

A PPP alapja a *High-Level Data Link Control* (HDLC) protokoll, amely az egyes PPP keretek határait definiálja, és egy 16 bites ellenőrző összeget biztosít.* A PPP keretek az IP protokollon kívül más, például a Novell IPX vagy az AppleTalk protokollok csomagjait is képesek tárolni. Ehhez a PPP egy protokollmezőt fűz a kiindulási HDLC kerethez, amely megadja a keretben szállított csomag típusát.

A *Link Control Protocol* (LCP) a HDLC fölött helyezkedik el, és feladata az adatkapcsolatra vonatkozó beállítások egyeztetése. A *Maximum Receive Unit* (MRU) például a datagramnak azt a maximális méretét adja meg, amelyet a kapcsolat valamely oldala hajlandó fogadni.

A PPP kapcsolat konfigurációs fázisának fontos lépése az ügyfél hitelesítése. Noha ez nem kötelező, a betárcsázó vonalak esetén elengedhetetlen ahhoz, hogy a behatolókat távol tarthassuk. Rendszerint a hívott gazdagép (a kiszolgáló) kéri fel az ügyfelet az azonosítására, amit azzal tehet meg, hogy bizonyítja, ismeri a megfelelő titkos kulcsot. Ha a hívó nem képes a helyes kulcs megadására, a kapcsolat véget ér. A PPP esetén a hitelesítés két irányban történik: a hívó szintén kérheti a kiszolgálót, hogy azonosítsa magát. A két hitelesítési folyamat egymástól teljesen független. A különböző típusú hitelesítésekhez két protokoll áll rendelkezésre: a *Password Authentication Protocol* (PAP) és a *Challenge Handshake Authentication Protocol* (CHAP). Ebben a fejezetben még külön szó lesz róluk.

Az adatkapcsolaton keresztül irányított minden hálózati protokoll (például IP vagy AppleTalk) beállítása dinamikusan történik, egy megfelelő *Network Control Protocol* (NCP) segítségével. Ahhoz például, hogy IP datagramokat küldjünk át a kapcsolaton, a PPP-t futtató két oldalnak mindenekelőtt egyeztetnie kell, hogy milyen IP címeket használnak majd. Az egyeztetéshez szükséges vezérlő protokoll az *Internet Protocol Control Protocol* (IPCP).

A szabványos IP datagramok küldése mellett a PPP az IP datagramok Van Jacobson-féle fejléctömörítését is támogatja. Az eljárás a TCP fejlécek csomagjait egészen kicsire,

* A HDLC ennél valójában jóval általánosabb protokoll; a HDLC-t a Nemzetközi Szabványügyi Szervezet (ISO) tervezte, és az X.25 specifikációnak is alapvető része.

3 bájt méretűre csökkenti. Ezt gyakran VJ fejléctömörítésnek is nevezik. A tömörítés alkalmazásának egyeztetése szintén az IPCP-n keresztül, indulásnál történik.

PPP a Linux rendszereken

A Linux a PPP funkcionalitást két részre osztva támogatja: a rendszermag komponens az alacsony szintű protokollokat (HDLC, IPCP, IPXCP stb.) kezeli, míg a felhasználói térben futó *pppd* daemon a különböző felső szintű protokollokkal foglalkozik; ilyenek a PAP és a CHAP. A Linux jelenlegi PPP szoftvere tartalmazza a *pppd* PPP daemont, valamint egy *chat* nevű programot, amely a távoli rendszer hívását automatizálja.

A PPP rendszermag illesztőprogramját Michael Callahan írta és Paul Mackerras dolgozta át. A *pppd* a Sun és a 386BSD számítógépekhez készült ingyenes PPP implementációból származik, ezt Drew Perkins és mások készítették, és Paul Mackerras tartotta karban. Linuxra Al Longyear dolgozta át. A *chat* szerzője Karl Fox.

A PPP megvalósításához egy különleges vonali üzemmód (line discipline) szükséges. Azért, hogy egy soros vonalat PPP kapcsolatként használhassunk, mindenekelőtt hagyományos módon létrehozzuk a kapcsolatot modemünkön keresztül, majd a vonalat PPP üzemmódba állítjuk. Ebben az üzemmódban a bejövő adatok a PPP illesztőprogramhoz kerülnek, amely ellenőrzi a bejövő HDLC keretek érvényességét (minden egyes HDLC keret egy 16 bites ellenőrző összeget rejt magában), kicsomagolja és célba juttatja azokat. Jelenleg a PPP képes a - szükség szerint Van Jacobson fejléctömörítéssel ellátott - IP protokoll és az IPX protokoll átvitelére.

A *pppd* a rendszermag illesztőprogramját segíti, elvégezve azokat az inicializálási és hitelesítési műveleteket, melyek a tulajdonképpeni hálózati forgalom megindításához szükségesek a kapcsolaton keresztül. A *pppd* viselkedését számos opcióval hangolhatjuk. Minthogy a PPP meglehetősen bonyolult, az opciók leírására kevés volna egyetlen fejezet. Következésképpen nem foglalkozunk a *pppd* valamennyi lehetőségével, csupán bevezető ismeretekre szorítkozunk. További információt találhatunk a *Using & Managing PPP* című kötetben, vagy a *pppd* forrás disztribúciójának *pppd* súgóoldalán és *README* állományaiban. Ezek valószínűleg elegendő segítséget nyújtanak majd ahhoz, hogy megoldhassuk a felmerülő problémákat, melyekkel a fejezet nem tud foglalkozni. A *PPP HOWTO* szintén hasznos forrás lehet.

A PPP beállításával kapcsolatban a legtöbb segítséget talán azoktól a Linux-felhasználóktól kaphatjuk, akik azonos Linux terjesztést futtatnak. A PPP beállításával kapcsolatban gyakran merülnek fel kérdések, ezért érdemes a helyi felhasználói csoport levelezőlistájával, az IRC Linux csatornával próbálkoznunk. Ha a hibát így sem tudtuk elhárítani, körülnézhetünk a *comp.protocols.ppp* hírcsoporton. Itt a *pppd* fejlesztésében részt vevők többségét megtalálhatjuk.

A pppd futtatása

Amikor PPP kapcsolattal szeretnénk az internetre kapcsolódni, be kell állítanunk az alapvető hálózatkezelési lehetőségeket, például a visszacsatoló eszközt és a feloldó könyvtárt. Ezekről a 4. és 5. fejezetben esett szó. Az */etc/resolv.conf* állományban egyszerűen

beállíthatjuk az internetszolgáltatónk névkiszolgálóját is, ez azonban azt jelenti, hogy minden DNS lekérdezés a soros vonalunkon keresztül történik. Ennél elképzelhető jobb is, ugyanis minél közelebb vagyunk a névkiszolgálóhoz (hálózati értelemben), annál gyorsabb a névkikeresés. Alternatív megoldást jelenthet, ha csak tárolásra szolgáló névkiszolgálót állítunk be hálózatunk valamely gazdagépén. Így amikor egy adott gazdagépről először kérünk DNS információt, a soros vonalon halad majd keresztül, a további lekérdezésekre azonban közvetlenül a helyi névkiszolgáló válaszol, ami jóval gyorsabb. Ezzel az elrendezéssel az 5. fejezetben foglalkoztunk.

Lássunk most egy bevezető példát a PPP kapcsolat kialakítására a *pppd* segítségével! Tegyük fel, hogy ismét a **vlager** gépnél vagyunk. Először tárcsázzuk fel a **c3po** PPP kiszolgálót, és jelentkezzünk be **ppp** felhasználóként. A **c3po** ekkor elindítja PPP illesztőprogramját. Amikor a betárcsázáshoz használt kommunikációs programból kilépünk, futtassuk a következő utasítást, az általunk használt soros eszköz nevét adva meg a `ttyS3` helyett:

```
# pppd /dev/ttyS3 38400 crtscts defaultroute
```

Az utasítás a soros vonali *ttyS3* üzemmódját PPP vonali üzemmódra állítja, és IP kapcsolatot hoz létre a **c3po** kiszolgálóval. A soros porton alkalmazott sebesség 38 400 bps. A `crtscts` opció bekapcsolja a porton a hardver kézfogást (handshake), ami 9600 bps sebesség fölött feltétlenül szükséges.

Indulása után a *pppd* először a kapcsolat néhány jellemzőjét egyezteti a távoli végggel, az LCP protokollon keresztül. Rendszerint elegendő az alapértelmezés szerint beállított opciók egyeztetése, ezért itt most nem állunk meg, csupán megjegyezzük, hogy az egyeztetés magában foglalja az IP címek kérését és kijelölését a kapcsolat mindkét végén.

Egyelőre tételezzük fel azt is, hogy a **c3po** nem vár tőlünk hitelesítést, így a konfigurációs fázis sikeresen befejeződik.

A *pppd* most az IPCP-n, vagyis az IP vezérlő protokollon keresztül egyezteti az IP paramétereket a társgéppel. Mivel korábban nem adtunk meg IP címet a *pppd* számára, a daemon a feloldótól kéri a helyi gazdagépnév kikeresését, és a kapott címet használja fel. Ezután mindkét vég közli a másikkal a saját címét.

Általában az alapértelmezett beállítások megfelelők. Még ha számítógépünk Ethernet hálózatra kapcsolódik is, ugyanazt az IP címet alkalmazhatjuk az Ethernet és a PPP interfész számára. Mindazonáltal a *pppd* lehetővé teszi az eltérő címek megadását, de akár azt is, hogy a társgépet felkérjük egy bizonyos cím használatára. Ezekről a lehetőségekről a fejezet *Az IP címek kiválasztása* című részében lesz szó.

Az IPCP beállítási fázison túljutva a *pppd* előkészíti gazdagépünk hálózatkezelő rétegét a PPP kapcsolatra. Először a PPP hálózati interfészt állítja pont-pont kapcsolatra; az első aktív PPP kapcsolathoz a *ppp0*-t, a másodikhoz a *ppp1*-et stb. használja. Ezután az útvonalválasztó táblázatban létrehoz egy bejegyzést, amely a kapcsolat másik végén lévő gazdagépre mutat. A korábbi példánkat folytatva, a *pppd* az alapértelmezett hálózati útvonalat a **c3po** kiszolgálóra irányította, hiszen megadtuk a `defaultroute` opciót.* Az alapértelmezett útvonal leegyszerűsíti az útválasztást, mert minden IP datagram, ame-

* Az alapértelmezett hálózati útvonal telepítése csak akkor történik meg, ha még nem létezik.

lyet egy nem helyi gépre szánunk, a c3po gépre jut; és valóban ez az egyetlen útvonal, melyen keresztül más gépeket elérhetünk. A *pppd* több különféle útválasztási sémát támogat, ezekkel a fejezetben később még foglalkozunk.

Az opcióállományok használata

Mielőtt a *pppd* feldolgozná a parancssori argumentumokat, átvizsgál néhány állományt, az úgynevezett opcióállományokat, melyekben az alapértelmezett beállításokat keresi. Az állományok bármilyen érvényes parancssori argumentumot tartalmazhatnak, tetszőleges számú sorban elhelyezve. A kettős kereszt karakter a megjegyzéseket vezeti be.

Az első opcióállomány, melyet a *pppd* minden indulásakor ellenőriz, az */etc/ppp/options*. Érdemes ezt használni a globális érvényű alapértelmezett értékek megadásához, mert így elkerülhetjük, hogy felhasználóink olyasmit tegyenek, ami biztonsági kockázatot hordoz. Ahhoz például, hogy a *pppd* hitelesítést igényeljen a társgéptől (PAP vagy CHAP hitelesítést), adjuk meg az `auth` opciót az állományban. Ezt az opciót a felhasználók nem írhatják felül, így lehetetlen PPP kapcsolatot létrehozni olyan rendszerrel, amely nem szerepel a hitelesítési adatbázisban. Fontos tudnunk azonban, hogy bizonyos opciók felülírhatók; jó példa rá a `connect`.

Az */etc/ppp/options* állomány értelmezése után következik a másik opcióállomány, a felhasználó saját (home) könyvtárában található .ppprc állomány feldolgozása. Ebben a felhasználó adhatja meg egyéni alapértelmezett opcióit.

Lássunk egy minta */etc/ppp/options* állományt:

```
# Globális opciók a vlager.vbrew.com gépen futó pppd daemonhoz
lock    # UUCP mintájú eszközlezárás használata
auth    # hitelesítés kérése
usehostname  # a helyi gazdagépnév használata a CHAP hitelesítéshez
domain vbrew.com  # tartományunk neve
```

A `lock` kulcsszó hatására a *pppd* a szabványos UUCP eszközlezárást alkalmazza. Ennek értelmében minden folyamat, amelynek egy soros eszközhöz, például a */dev/ttyS3* eszközhöz van hozzáférése, létrehoz egy zárolási állományt mondjuk *LCK..ttyS3* néven, és elhelyezi egy különleges, a zárolási állományokat tartalmazó könyvtárban, így jelezve, hogy az eszköz használatban van. Erre azért van szükség, hogy más programok, például a *minicom* vagy az *uucico* ne nyithassa meg a soros eszközt, amíg azt a PPP használja.

A következő három opció a hitelesítésre vonatkozik, és ezért a rendszerbiztonságot is érinti. A hitelesítési opciókat érdemes a globális konfigurációs állományban elhelyezni, mert itt különleges „előjogokat" élveznek, és a felhasználók a *~/.ppprc* opcióállományokkal nem írhatják felül.

A betárcsázás automatizálása a chat programmal

Az előző példa egyik hiányossága, hogy a *pppd* indítása előtt a kapcsolatot manuálisan kell felépítenünk. A *pppd* a távoli rendszerre történő bejelentkezéskor és kapcsolódás-

kor egy külső programra vagy héjszkriptre támaszkodik. A végrehajtandó parancsot a `connect` parancssori argumentummal adhatjuk meg. A *pppd* a parancs szabványos bemenetét és kimenetét a soros vonalra irányítja.

A *pppd* szoftvercsomagban találunk egy nagyon egyszerű programot, amelynek neve *chat*. Ezt pontosan a fent leírt módon használhatjuk az egyszerű bejelentkezési folyamatok automatizálására. A *chat* utasítást rövidesen egy kicsit közelebbről is megvizsgáljuk.

Összetett bejelentkezési folyamatok esetén azonban a *chat* programnál hatékonyabb eszközre van szükségünk. Megfelelő alternatíva lehet Don Libes programja, az *expect*. Rendkívül hatékony nyelve Tcl alapú, és kimondottan ilyen jellegű felhasználásra tervezték. Azok, akiknek a bejelentkezési folyamata felszólítás/válasz (challenge/response) hitelesítést kíván és a kalkulátorokra emlékeztető kulcsgenerátort igényel, az *expect* programot megfelelően felkészültnek találják majd a feladat elvégzésére. Mivel rengeteg variációs lehetőség létezik, a könyvben nem foglalkozunk az *expect* szkript kidolgozásával. Elég annyi, hogy az *expect* szkript meghívásához elhelyezzük nevét a *pppd* `connect` opciójában. Fontos megjegyeznünk, hogy amikor a szkript fut, a szabványos be- és kimenet a modemhez kerül, nem a *pppd* daemont meghívó terminálhoz. Amennyiben a hitelesítéshez a felhasználó aktív közreműködésére is szükség van, a feladatot egy tartalék virtuális terminál megnyitásával vagy egyéb módszerrel oldhatjuk meg.

A *chat* utasításban egy *chat* szkriptet adhatunk meg. A *chat* szkript alapvetően azokat a kifejezéseket tartalmazza váltakozó sorrendben, amelyeket a távoli rendszertől várunk, illetve amelyeket válaszként el kívánunk küldeni. Ezeket *elvárt* és *küldendő* kifejezéseknek fogjuk nevezni. Íme egy jellemző részlet a *chat* szkriptből:

```
ogin: b1ff ssword: s3|<r1t
```

A szkript utasítja a *chat* programot, hogy várja meg, amíg a távoli rendszer elküldi a bejelentkezési promptot, majd válaszul küldje el a **b1ff** bejelentkezési nevet. Mivel csak az `ogin:` kifejezést várjuk, nem számít, hogy a bejelentkezési (login) prompt kis vagy nagy L betűvel kezdődik, esetleg nem is tartalmaz L betűt. A következő ismét egy elvárt kifejezés. Hatására a *chat* a jelszópromptra vár, majd elküldi jelszavunkat.

Gyakorlatilag ez a *chat* szkriptek feladata. Egy PPP kiszolgálót tárcsázó teljes szkript természetesen tartalmazná a megfelelő modemparancsokat is. Tételezzük fel, hogy modemünk érti a Hayes utasításkészletet, és a kiszolgáló telefonszáma 318714. A **c3po** kiszolgálóval kapcsolatot létesítő *chat* szkriptet ekkor így hívhatnánk meg:

```
$ chat -v '' ATZ OK ATDT318714 CONNECT '' ogin: ppp word: GaGariN
```

Definíció szerint az első kifejezésnek egy elvárt kifejezésnek kell lennie, de mert a modem egy szót sem szól addig, amíg nem noszogatjuk, rávesszük a *chat* szkriptet, hogy ugorja át az első elvárt kifejezést – ezért itt üres karakterláncot adunk meg. Majd elküldjük az `ATZ` utasítást, amely nem más, mint a Hayes kompatibilis modemeket alaphelyzetbe hozó utasítás (reset), és várjuk a választ (`OK`). A következő kifejezés a betárcsázó parancsot és a telefonszámot küldi el a *chat* részére, és cserébe várja a `CONNECT` üzenetet. Ezt ismét egy üres karakterlánc követi, mert jelenleg semmit nem szeretnénk küldeni, csupán a bejelentkezési promptra várunk. A *chat* szkript fennmaradó része a korábban

leírtaknak megfelelően működik. Mindez talán lehetne egyszerűbb is, és pillanatokon belül valóban látni fogjuk, hogy jóval érthetőbb *chat* szkripteket is készíthetünk.

A `-v` opció a *chat* programmal minden tevékenységről bejegyzést készített a syslog daemon `local2` kategóriában.*

A *chat* szkript megadása a parancssorban bizonyos kockázatot hordoz, hiszen a felhasználók megtekinthetik egy folyamat parancssorát a *ps* utasítással. A kockázatot elkerülhetjük, ha a *chat* szkriptet egy állományban helyezzük el, például *dial-c3po* néven. A `-f` opció és az állomány nevének megadásával utasíthatjuk a *chat* programot, hogy a szkriptet a parancssor helyett az állományból olvassa ki. Az eljárás további előnye, hogy az elvárt kifejezések is jobban átláthatók. Példánkat átalakítva nézzük most meg a *dial-c3po* állomány tartalmát:

```
''      ATZ
OK      ATDT318714
CONNECT ''
ogin:   ppp
word:   GaGariN
```

Az ily módon felépített *chat* szkriptállományban az elvárt karakterlánc a bal oldalon, a küldendő válasz a jobb oldalon áll, ami megkönnyíti a szkript olvasását és értelmezését.

A teljes *pppd* hívás most így fest:

```
# pppd connect "chat -f dial-c3po" /dev/ttyS3 38400 -detach \
        crtscts modem defaultroute
```

A betárcsázó szkriptet kijelölő `connect` opción kívül két további opciót fűzünk a parancssorhoz: a `-detach` utasítja a *chat* programot, hogy ne váljon le a konzolról és ne alakuljon háttérben futó folyamattá, míg a `modem` kulcsszó hatására a program modemspecifikus műveleteket végez a soros eszközön, például megszakítja a vonalat a hívás előtt és után. E kulcsszó nélkül a *pppd* nem figyeli a port DCD vonalát, és ezért nem érzékeli, ha a távoli vég váratlanul megszakítja a kapcsolatot.

A bemutatott példák elég egyszerűek, de a *chat* ennél összetettebb szkriptekhez is alkalmas. Megadhatunk például olyan kifejezést, melynek hatására a *chat* hibaüzenetet küldve felfüggeszti működését. Jellemző megszakító üzenet például a `BUSY` és a `NO CARRIER`, melyet a modem általában akkor állít elő, ha a hívott szám foglalt vagy nem válaszol. Azért, hogy a *chat* ezeket az üzeneteket azonnal felismerje és ne az időkorlátra várjon, az `ABORT` kulcsszóval megadhatjuk őket a szkript elején:

```
$ chat -v ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ...
```

Hasonló módon a *chat* szkript egyes részeire vonatkozó időkorlátot is szabályozhatjuk, amihez a `TIMEOUT` opciót használjuk.

* Ha a *syslog.conf* állomány módosításával a bejegyzéseket más állományba irányítjuk át, győződjünk meg róla, hogy az állomány olvasási joga korlátozott, mert a *chat* az alapértelmezés szerint bejegyzi a teljes *chat* szkriptet is – a jelszavakkal együtt.

Esetenként a *chat* szkript egyes részeinek futását bizonyos feltételhez szeretnénk kötni: ha például nem kapunk a távoli végtől bejelentkezési promptot, szeretnénk egy BREAK vagy kocsivissza jelet küldeni. Ehhez egy alszkriptet fűzünk az elvárt kifejezéshez. Az alszkript a küldendő és elvárt kifejezések sorozatát tartalmazza, éppúgy, mint a főszkript; a kifejezéseket kötőjellel választjuk el. Az alszkript akkor hajtódik végre, ha a várt kifejezés, melyhez hozzáfűztük, nem érkezik meg idejében. Az előbbi példát folytatva így módosíthatnánk a *chat* szkriptet:

```
ogin:-BREAK-ogin: ppp ssword: GaGariN
```

Amikor a *chat* látja, hogy a távoli rendszer nem küldi a bejelentkezési promptot, végrehajtja az alszkriptet, elküldve a BREAK üzenetet, majd tovább várakozik a bejelentkezési promptra. Amennyiben a prompt most megjelenik, a szkript a szokásos módon fut tovább; egyébként hibaüzenettel leáll.

Az IP beállítási lehetőségei

Az IPCP feladata a különböző IP paraméterek egyeztetése a kapcsolat konfigurációs fázisában. Rendszerint mindkét társgép elküldi a maga IPCP beállítást kérő csomagját, megjelölve, hogy az alapértelmezett értékektől hol kíván eltérni, és megadva az új értékeket. Fogadáskor a távoli vég egyesével megvizsgálja az opciókat, és vagy elfogadja, vagy visszautasítja azokat.

Az egyeztetni kívánt IPCP opciók tekintetében a *pppd* nagyon jól szabályozható. A beállítást szolgáló különféle parancssori kapcsolókat most részletesen is megvizsgáljuk.

Az IP címek kiválasztása

Minden IP interfészhez tartoznia kell egy IP címnek, és a PPP eszközöknek mindig van IP címe. A PPP protokollcsomag biztosít egy eljárást, amely lehetővé teszi az IP címek automatikus kiosztását a PPP interfészek számára. A pont-pont kapcsolat egyik végén lévő PPP program képes a távoli vég számára IP címet kijelölni, de mindketten használhatják saját címüket is.

A sok ügyfelet kezelő PPP kiszolgálók a címeket dinamikusan osztják ki; a rendszerek csak a bejelentkezéskor kapják meg a címüket, kijelentkezéskor pedig elveszítik azt. Így a szükséges IP címek számát a telefonvonalak számára lehet redukálni. Noha a korlátozás kényelmes a PPP telefonos kiszolgáló üzemeltetői számára, gyakran kényelmetlen a bejelentkező felhasználóknak. Az 5. fejezetben láttuk, hogy a gazdagépnevek leképzése IP címekre egy adatbázis segítségével történik. Ahhoz, hogy valaki kapcsolatot hozzon létre gazdagépünkhöz, ismernie kell az IP címünket vagy az ahhoz tartozó gazdagépnevet. Ha olyan PPP szolgáltatást veszünk igénybe, amely az IP címeket dinamikusan osztja ki, ez az információ csak úgy biztosítható, ha létezik valamilyen lehetőség a DNS adatbázis frissítésére, miután az IP címet megkaptuk. Noha ilyen rendszerek valóban léteznek, részletesen nem foglalkozunk velük; ehelyett szemügyre veszünk egy

kívánatosabb megközelítést, ahol egy hálózati kapcsolat létesítésekor mindig ugyanazt az IP címet kapjuk.*

Korábbi példánkban a *pppd* program a **c3po** gazdagépet betárcsázva hozott létre IP kapcsolatot. A rendszertől nem vártuk el, hogy a végeken adott IP címeket használjon, hanem a *pppd*-re bíztuk, hogy elvégezze alapértelmezett feladatát. Ennek megfelelően a *pppd* megkísérelte a helyi gazdagépnév, példánkban a **vlager** feloldását, vagyis IP címének kiderítését, és ezt használta a helyi végponton, miközben hagyta, hogy a távoli gép, a **c3po** saját maga válassza meg a címét. A PPP azonban emellett más megoldásokat is kínál.

Ha meghatározott címeket szeretnénk használni, általában a következő opciót adjuk meg a *pppd* számára:

```
helyi_cím:távoli_cím
```

A *helyi_cím* és a *távoli_cím* megadható pontozott négyes jelöléssel vagy a gazdagép nevével.** Az opció hatására a *pppd* megkísérli az első címet saját IP címeként, a második címet pedig a távoli gép címeként felhasználni. Ha a távoli gép az IPCP egyeztetés során ezek bármelyikét elutasítja, nem jön létre IP kapcsolat.***

Amikor feltárcsázunk egy kiszolgálót, és attól várjuk az IP címünk kiosztását, biztosítanunk kell, hogy a *pppd* nem próbál maga is IP címet kérni. Ehhez adjuk meg a `noipdefault` opciót, és hagyjuk a *helyi_cím* helyét üresen. A `noipdefault` opció megakadályozza, hogy a *pppd* a gazdagépnévhez tartozó IP címet használja helyi címként.

Ha csupán a helyi címet szeretnénk beállítani, de elfogadjuk a társgép által használt címet, egyszerűen hagyjuk el a *távoli_cím* részt. Ha azt szeretnénk, hogy a **vlager** a 130.83.4.27 IP címet használja a sajátja helyett, adjuk meg ezt a címet a parancssorban. Hasonló módon, amikor kizárólag a távoli címet szeretnénk beállítani, hagyjuk üresen a *helyi_cím* mezőt. Alapértelmezésben a *pppd* ekkor a gazdagépünkhöz tartozó címet alkalmazza.

Útválasztás a PPP kapcsolaton

A hálózati interfész beállítása után a *pppd* általában csak a társgéphez alakít ki gazdagépútvonalat. Ha a távoli gazdagép egy LAN része, természetesen a társgép „mögött" található gazdagépekhez is szeretnénk kapcsolatot; ez esetben egy hálózati útvonalat kell létrehoznunk.

Láttuk, hogy a *pppd* alapértelmezett útvonala beállítható a `defaultroute` opcióval. Az opció különösen jól jön, ha a betárcsázott PPP kiszolgáló szolgál számunkra internetes átjáróként.

Fordított esetben, amikor a mi rendszerünk lép fel átjáróként egyetlen gazdagép számára, szintén egyszerűen megoldható a feladat. Vegyük például a Virtuális Sörgyár egyik

* Két dinamikus hosztkijelölési eljárásról olvashatunk a *http://www.dynip.com/címen*.

** A gazdagépnevek megadása ebben az opcióban hatással van a CHAP hitelesítésre. Ezzel kapcsolatban lásd a fejezet A *CHAP jelszóállománya* című részét.

*** Az ipcp-accept-local és *ipcp-accept-remote* opciók arra utasítják a *pppd* programot, hogy fogadja el a távoli PPP által javasolt helyi, illetve távoli címet még akkor is, ha a konfigurációban más címet adtunk meg. Ha az opciókat nem állítjuk be, a *pppd* elutasít minden kísérletet az IP címek egyeztetésére.

alkalmazottját, akinek otthoni számítógépe a **oneshot**. Tegyük fel azt is, hogy a **vlager** gépet betárcsázó PPP kiszolgálónak állítottuk be. Ha a **vlager** a beállítása szerint dinamikusan osztja ki a Sörgyár alhálózatához tartozó IP címeket, megadhatjuk a `proxyarp` opciót - ekkor a *pppd* létrehoz egy proxy ARP bejegyzést a **oneshot** részére, ami automatikusan elérhetővé teszi a **oneshot** gépet a Sörgyár és a Borászat bármely gazdagépéről.

A helyzet azonban nem mindig ilyen egyszerű. Két helyi hálózat összekapcsolása rendszerint egy specifikus hálózati útvonal megadását igényli, mivel e hálózatok saját alapértelmezett útvonallal rendelkezhetnek. Emellett, ha két társgép a PPP kapcsolatot használja alapértelmezett útvonalként, hurok jöhet létre, amelyen az ismeretlen rendeltetésű csomagok ide-oda vándorolnak a társgépek között, míg életidejük végül lejár.

Képzeljük el, hogy a Virtuális Sörgyár új leányvállalatot nyit egy másik városban. A leányvállalat saját Ethernet hálózatot tart fenn, ahol az IP hálózati szám a **172.16.3.0**, amely a Sörgyár B osztályú hálózatának 3-as alhálózata. A leányvállalat PPP kapcsolaton keresztül szeretne a Sörgyár hálózatára kapcsolódni, hogy ügyfeleinek adatbázisát frissíthesse. A **vlager** ismét a Sörgyár hálózatának átjárójaként lép fel, és támogatja a PPP kapcsolatot; a leányvállalatnál a társgép neve **vbourbon**, IP címe pedig a **172.16.3.1**.

Amikor a **vbourbon** a **vlager** géphez kapcsolódik, az alapértelmezett útvonalat szokás szerint a **vlager** gépre állítja. A **vlager** gépen azonban csak a pont-pont útvonal létezik a **vbourbon** géphez, míg a **vbourbon** gépet átjáróként használó 3-as alhálózathoz külön be kell állítanunk egy hálózati útvonalat. Ezt a *route* paranccsal manuálisan is megtehetjük a PPP kapcsolat létrejötte után, ám ez egy kényelmetlen megoldás. Szerencsére az útvonalat automatikus beállításra is átválthatjuk a *pppd* egyik szolgáltatásával, amelyről mindeddig nem esett szó - az *ip-up* utasítással. A parancs egy héjszkript vagy program az */etc/ppp* könyvtárban, melyet a *pppd* a PPP interfész konfigurálása után futtat le. Ha létezik, a következő paraméterekkel hívhatjuk meg:

```
ip-up interfész eszköz sebesség helyi_cím távoli_cím
```

A 6.1. táblázat az egyes argumentumok jelentését foglalja össze (az első oszlopban megadott számokat a héjszkript használja az egyes argumentumokra történő hivatkozáskor).

6.1. táblázat. Az ip-up argumentumai

Név	*Feladat*
`interfész`	A használt hálózati interfész, például `ppp0`.
`eszköz`	A használt soros eszköz állományának útvonal a (`/dev/tty`, az stdin/stdout esetén).
`sebesség`	A soros eszköz sebessége bit/másodpercben megadva.
`helyi_cím`	A kapcsolat helyi végének IP címe pontozott négyes jelöléssel.
`távoli_cím>`	A kapcsolat távoli végének IP címe pontozott négyes jelöléssel.

Esetünkben az *ip-up* szkript a következő kódrészletet tartalmazhatja:*

* Ha azt szeretnénk, hogy más webhelyekhez is készüljön útvonal, amikor bejelentkeznek, case utasításokat helyeznénk a példa kipontozott (...) részébe.

```
#!/bin/sh
case $5 in
172.16.3.1)             # ez a vbourbon
        route add -net 172.16.3.0 gw 172.16.3.1;;
...
esac
exit 0
```

Az */etc/ppp/ip-down* az ip-up műveletek visszavonására alkalmas a PPP kapcsolat bontása után, így például az */etc/ppp/ip-down* szkriptünk tartalmazhat egy útvonalparancsot, mellyel eltávolítjuk az */etc/ppp/ip-up* szkriptben létrehozott útvonalat.

Az útválasztó séma azonban még nem teljes. Mindkét PPP gazdagépen be kell állítanunk az útvonalválasztó táblázat bejegyzéseit, de jelenleg még egyik hálózaton lévő gazdagépek sem tudnak semmit a PPP kapcsolatról. Ez nem gond, ha az alhálózatok összes gazdagépének alapértelmezett útvonala a **vbourbon** gépre mutat, és a Sörgyár gazdagépeinek útvonala alapértelmezés szerint a **vlager**.

Kapcsolatvezérlési lehetőségek

Láttuk, hogy az LCP protokoll a kapcsolat jellemzőinek egyeztetésére és a kapcsolat ellenőrzésére szolgál.

Az LCP által egyeztetett két legfontosabb opció az *Asynchronous Control Character Map* és a *Maximum Receive Unit*. Léteznek egyéb LCP beállítási lehetőségek is, ezek azonban túl specifikusak ahhoz, hogy itt tárgyaljuk.

Az Asynchronous Control Character Map, elterjedt nevén az *async map* vagy *térkép* aszinkron kapcsolatokon (például telefonvonalakon) használható a vezérlő karakterek kiváltására: ha nem szeretnénk, hogy a kapcsolatot létrehozó berendezések értelmezzék a vezérlő karaktereket, kétkarakteres különleges sorozattal helyettesítjük és így kiváltjuk azokat. Érdemes lehet például elkerülni a szoftveres kézfogáshoz használt XON és XOFF karaktereket, mert egy roszszul beállított modemet összezavarhat a XOFF jel. Egy további vezérlő karakter, amelyet szintén nem árt kiváltani, a Ctrl-l (a *telnet* escape karaktere). A PPP lehetővé teszi, hogy a 0-tól 31-ig terjedő ASCII kódtartomány karaktereit kiváltsuk, ha megadjuk azokat az async mapben.

Az async map egy 32 bit széles bittérkép hexadecimális formában. A legkisebb helyi értékű bit a decimális ASCII NULL karakternek felel meg, míg a legnagyobb helyi értékű bit a decimális ASCII 31-nek. Ezt a harminckét ASCII karaktert nevezzük vezérlő karakternek. Ha a bittérkép egy bitje beállított állapotban van, akkor a neki megfelelő karaktert ki kell váltani, mielőtt a kapcsolaton áthaladhat.

Ha közölni szeretnénk a társgéppel, hogy nem szükséges minden vezérlő karaktert kiváltania, csupán egyeseket, az `asyncmap` kulcsszóval adhatunk meg egy async térképet a *pppd* számára. Így például ha csak az `^S` és `^Q` (ASCII 17 és 19, általában a XON és XOFF vezérlő karakterek) kiváltása szükséges, adjuk meg a következő opciót:

```
asyncmap 0x000A0000
```

Az értékek konvertálása nagyon egyszerű, feltéve hogy tudjuk, hogyan válthatunk át bináris számokat hexadecimálissá. Terítsünk ki képzeletben 32 bitet magunk elé. A jobb

szélső bit a decimális ASCII 00 (NULL) karakternek felel meg, míg a bal szélső a decimális ASCII 32-nek. Állítsuk a kiváltani kívánt karaktereknek megfelelő bitek értékét egyre, minden más bit értékét nullára. A *pppd* hexadecimális értéket vár, ezért 4 bitenként sorban alakítsuk át a sorozatot hexadecimális számmá, így végül nyolc hexadecimális értéket kapunk. Ezeket egymás után írva és eléjük a hexadecimális értékre utaló „0x" kifejezést fűzve el is készültünk az átalakítással.

Az asynx térkép kezdeti értéke `0xffffffff`, vagyis minden vezérlő karaktert kiváltunk. Alapértelmezett értéknek ez biztonságos, ám rendszerint ennél kevesebb is elegendő. Mivel az async térképen megjelenő minden egyes karakter a kapcsolati átvitelben két karaktert jelent, a kiváltás ára a kapcsolat nagyobb mértékű terhelése, ami teljesítménycsökkenést okoz.

A legtöbb esetben a `0x0` asynx térkép megfelelő választás. Ekkor nem történik kiváltás.

A Maximum Receive Unit (MRU) a fogadni kívánt HDLC keretek maximális méretét jelzi a társgép számára. Ne keverjük össze a Maximum Transfer Unit (MTU) értékkel, a kettőnek kevés köze van egymáshoz. Az MTU a rendszermag hálózatkezelő eszközének paramétere, és az interfész által forgalmazható legnagyobb keretméretet írja le. Az MRU inkább javaslat a távoli vég számára, hogy az MRU értékénél ne generáljon nagyobb kereteket; az interfész persze ettől még akár 1500 bájtos keretek fogadására is képes.

Az MRU kiválasztása ezért nem annyira azzal függ össze, hogy milyen a kapcsolat átviteli képessége, hanem azzal, hogy mi biztosítja a legjobb átvitelt. Ha interaktív alkalmazásokat fogunk futtatni a kapcsolaton, érdemes az MRU értékét kicsire, akár 296-ra állítani, hogy egy esetleges nagyobb csomag (mondjuk egy FTP munkafolyamatból) ne okozzon „kurzorugrást". Ahhoz, hogy a *pppd* 296 értékű MRU-t kérjen, az `mru 296` opciót adnánk meg. A kis MRU értékeknek azonban csak VJ fejléctömörítés esetén van értelme (alapértelmezésben a tömörítés működik), mert egyébként a sávszélesség jelentős részét az egyes datagramok IP fejlécének átvitele foglalná le.

A *pppd* emellett néhány olyan LCP opciót is értelmezni tud, amelyek az egyeztetési folyamat általános viselkedési jellemzőit szabályozzák; ilyen például a beállítással kapcsolatos kérések maximális száma, amikor több kérés esetén bontjuk a kapcsolatot. Ha nem tudjuk biztosan, minek mi a szerepe, jobb, ha nem nyúlunk ezekhez az opciókhoz.

Végül ejtsünk szót két további lehetőségről, amelyek az LCP visszhangüzeneteket szabályozzák. A PPP két üzenetet definiál, ezek az *Echo Request* és az *Echo Response*. A *pppd* arra használja őket, hogy a kapcsolat működőképességét ellenőrizze. Az ellenőrzést bekapcsolhatjuk az `lcp-echo-interval` opcióval, amely után egy másodpercben kifejezett időtartamot adunk meg. Ha ezen az időn belül nem érkezik keret a távoli gazdagéptől, a *pppd* Echo Request üzenetet küld és várja, hogy a társgép erre egy Echo Response üzenettel válaszoljon. Amennyiben a társ nem válaszol, adott számú kérés küldése után megszakítja a kapcsolatot. Ezt a számot az `lcp-echo-failure` opcióval adhatjuk meg. Alapértelmezés szerint ez a szolgáltatás nem működik.

A biztonság általános kérdései

Egy rosszul beállított PPP daemon veszedelmes biztonsági rést hagyhat. Épp olyan veszedelmeset, mintha akárki rákapcsolódhatna Ethernet hálózatunkra (ez pedig valóban

nagyon veszedelmes). Ebben a részben néhány olyan intézkedést ismertetünk, melyekkel biztonságossá tehetjük PPP rendszerünket.

A hálózati eszköz és az útvonalválasztó táblázat beállításához root jogosultság szükséges. Ezt általában úgy oldjuk meg, hogy a *pppd* programot a rendszergazda jogosultságával futtatjuk (setuid **root**). A pppd azonban különféle biztonsági vonatkozású opciókat is kínál a felhasználóknak.

Azért, hogy megakadályozzuk azokat a támadásokat, amelyeket egy felhasználó a **pppd** opciók manipulálásával indíthat, be kell állítanunk néhány alapértelmezett értéket a globális */etc/ppp/options* állományban - ilyenek például a fejezet *Az opcióállományok használata* című részének mintaállományában bemutatott értékek. Ezek némelyikét, például a hitelesítéssel kapcsolatos beállításokat a felhasználók nem írhatják felül, így észszerű védelmet nyújtanak a mesterkedésekkel szemben. A védelmet igénylő opciók egyike a `connect`. Amennyiben szándékunkban áll megengedni a nem root felhasználóknak, hogy a *pppd* program meghívásával kapcsolódjanak az internetre, minden esetben helyezzük el a `connect` és `noauth` opciókat az */etc/ppp/options* globális opcióállományban. Ezt elmulasztva a felhasználók tetszőleges utasításokat futtathatnak root jogosultsággal, ha az utasításokat a *pppd* sorban vagy a személyes opcióállományukban connect parancsként adják meg.

Ajánlatos lehet a *pppd* futtatására jogosult felhasználókat is korlátoznunk egy csoport létrehozásával az/etc/group állományban, ahová csak azokat a felhasználókat vesszük fel, akiknek engedélyezni akarjuk a PPP daemon futtatását. Ezután állítsuk a *pppd* daemon csoportos tulajdonjogát erre a csoportra, és távolítsuk el azokat a jogosultságokat, melyek mindenki számára futtatási jogot biztosítanak. Feltéve hogy csoportunknak a **dialout** nevet adtuk, ezt valahogy így oldhatnánk meg:

```
# chown root /usr/sbin/pppd
# chgrp dialout /usr/sbin/pppd
# chmod 4750 /usr/sbin/pppd
```

Természetesen védenünk kell magunkat azoktól a rendszerektől is, melyekkel PPP kapcsolatot tartunk fenn. A magukat másnak kiadó gazdagépek kivédéséhez mindig kérjünk hitelesítést a társgéptől. Emellett ne engedjük, hogy az idegen gazdagépek tetszőleges IP címeket használjanak - korlátozzuk őket csupán néhány címre. A következő részben ezekkel a témakörökkel foglalkozunk.

A PPP és a hitelesítés

A PPP két hitelesítési protokollal biztosítja, hogy egy rendszer hitelesítést kérhessen egy társrendszertől: az első a *Password Authentication Protocol* (PAP), a második a *Challenge Handshake Authentication Protocol* (CHAP). A kapcsolat létrejötte után bármelyik végpont kérheti a másiktól, hogy hitelesítse magát, függetlenül attól, ki volt a hívó és ki a hívott. A következő leírásban némileg pongyolán az „ügyfél" és „kiszolgáló" elnevezéseket alkalmazzuk a hitelesítést kérő és a kérésre válaszoló rendszerek megkülönböz-

tetésére. A PPP daemon ugyancsak egy LCP beállítási kéréssel kérheti társától a hitelesítést, megadva a kívánt hitelesítési protokollt.

PAP vagy CHAP?

A sok internetszolgáltató által kínált PAP alapvetően a rendes bejelentkezési eljáráshoz hasonlóan működik. Az ügyfél úgy azonosítja magát, hogy elküld egy felhasználói nevet és egy (esetleg titkosított) jelszót a kiszolgálóra, melyet ez utóbbi összevet titkos adatbázisával. Az eljárás sebezhető, mert ha valaki „lehallgatja" a soros vonalat, megszerezheti a jelszót, de sebezhető az ismétlődő, próba-szerencse támadásokkal szemben is.

A CHAP kiküszöböli ezeket a hiányosságokat. A CHAP rendszeren a kiszolgáló a gazdagépneve mellett egy véletlenszerűen előállított „felszólító" karakterláncot is küld az ügyfélnek Az ügyfél a gazdagépnév alapján kikeresi a megfelelő kulcsot, kombinálja a kettőt, és egy egyirányú hasító függvénnyel (hash) titkosítja azt. Az eredményt saját gazdagépnevével egyetemben elküldi a kiszolgálónak. A kiszolgáló szintén végrehajtja e műveleteket, és ha az eredmény egyezik, hitelesnek fogadja el az ügyfelet.

A CHAP ugyanakkor nem elégszik meg azzal, hogy az ügyfelet csak a kapcsolat létesítésekor azonosítsa, hanem szabályos időközönként küld egy felszólító kifejezést. Így ellenőrizheti, hogy az ügyfél helyére nem lépett egy behatoló, például a telefonvonalak megcserélésével vagy a modem beállítási hibája miatt – ilyenkor ugyanis a PPP daemon nem észleli, hogy az eredeti hívás már befejeződött, és valaki más jelentkezett be a hívó helyett.

A *pppd* a PAP és a CHAP titkos kulcsait két önálló állományban, az */etc/ppp/pap-secrets* és az */etc/ppp/chap-secrets* állományokban tárolja. Ahhoz, hogy megadjuk, hogy egy távoli gazdagépen PAP vagy CHAP hitelesítést szeretnénk-e alkalmazni, illetve hogy a távoli vég melyik hitelesítést alkalmazza velünk kapcsolatban, elegendő, ha a megfelelő állományban elhelyezünk egy bejegyzést a távoli gazdagépről.

Alapértelmezés szerint a *pppd* nem kíván hitelesítést a távoli gazdagéptől, de hajlandó hitelesíteni magát, ha a távoli gép erre felkéri. Mivel a CHAP jóval megbízhatóbb a PAP hitelesítésnél, a *pppd* lehetőség szerint ezt szeretné alkalmazni. Ha a társ nem támogatja, vagy ha a *pppd* nem találja a távoli rendszerhez tartozó titkos CHAP kulcsot a *chap-secrets* állományban, akkor hagyatkozik a PAP hitelesítésre. Ha a társ titkos PAP kulcsát sem találja, elutasítja a hitelesítést és megszakítja a kapcsolatot.

A *pppd* viselkedését több módon is befolyásolhatjuk. Az `auth` kulcsszót megadva a *pppd* megköveteli, hogy a társ hitelesítse magát, és hajlandó akár CHAP, akár PAP hitelesítést alkalmazni, feltéve hogy CHAP, illetve PAP adatbázisa tartalmazza a társhoz tartozó titkos kulcsot. Más lehetőségek is léteznek az adott hitelesítési protokoll ki- és bekapcsolására, ezekkel azonban itt nem foglalkozunk.

Amennyiben a PPP protokollon keresztül elért valamennyi rendszer hajlandó hitelesíteni magát, helyezzük az `auth` opciót a globális */etc/ppp/options* állományba, az egyes rendszerek jelszavát pedig a *chap-secrets* állományba. Ily módon biztosíthatjuk, hogy csak hitelesített rendszer kapcsolódhat a gazdagépünkre.

A következő két részben a *pap-secrets* és *chap-secrets* titkos állományokkal ismerkedünk meg. Ezek az */etc/ppp* könyvtárban helyezkednek el és hármas bejegyzéseket tartalmaznak (ügyfél, kiszolgáló, jelszó); végül készenlétben állhat még egy IP címlista. Az ügyfél- és a kiszolgálómezők értelmezése különbözik CHAP és PAP esetén, és attól is

függ, hogy mi kívánjuk azonosítani magunkat a társ számára, vagy mi kérjük, hogy a kiszolgáló hitelesítse magát.

A CHAP titkos állománya

Amikor a *pppd* a CHAP segítségével hitelesíti magát egy kiszolgáló számára, olyan bejegyzést keres a *chap-secrets* állományban, amelynek ügyfélmezője megegyezik a helyi gazdagépnévvel, és a kiszolgálómezője megegyezik a CHAP felszólító üzenetben elküldött távoli kiszolgáló nevével. Amikor a *pppd* a társ azonosítását kéri, a szerepek megcserélődnek: most olyan bejegyzést keres, ahol az ügyfélmező megegyezik a távoli gazdagép nevével (melyet az ügyfél a CHAP válaszban küld el), és a kiszolgálómező megegyezik a helyi gazdagépnévvel.

Lássuk most a **vlager** minta *chap-secrets* állományát:*

```
# A vlager.vbrew.com CHAP secrets állománya
#
# kliens          kiszolgáló       titkos kulcs          cím
#———————————————————————————————————————————————————————
vlager.vbrew.com  c3po.lucas.com   "Use The Source Luke" vlager.vbrew.com
c3po.lucas.com    vlager.vbrew.com "arttoo! arttoo!"     c3po.lucas.com
*                 vlager.vbrew.com "TuXdrinksVicBitter"  pub.vbrew.com
```

Amikor a **vlager** PPP kapcsolatot hoz létre a **c3po** géppel, a **c3po** CHAP felszólító üzenetet küldve felkéri a **vlager** gépet, hogy hitelesítse magát. A **vlager** gépen futó *pppd* ekkor átnézi a *chap-secrets* állományt: olyan bejegyzést keres, amelyben az ügyfélmező a **vlager.vbrew.com**, míg a kiszolgálómező a **c3po.lucas.com**. A feltételnek a példa első sora felel meg.** Ezután a felszólító elkészíti karakterláncból és a titkos kulcsból (`Use The Source Luke`) a CHAP választ, és elküldi a **c3po** gépre.

A *pppd* egyúttal egy CHAP felszólítást is készít a **c3po** számára, amely egy egyedi felszólító karakterláncot, valamint saját teljes minősített gazdagépnevét (**vlager.vbrew.com**) tartalmazza. A **c3po** azonos módon készíti el a CHAP választ, majd elküldi a **vlager** gépnek. A *pppd* a válaszból kiemeli az ügyfél gazdagépnevét (**c3po.vbrew.com**), és a *chap-secrets* állományban megpróbál találni egy olyan sort, ahol a **c3po** az ügyfél és a **vlager** a kiszolgáló. A második sor egyezik, így a *pppd* egyesíti a CHAP felszólítást és az `arttoo! arttoo!` titkos kulcsot***, titkosítja azokat, és az eredményt összeveti a **c3po** CHAP válaszával.

Az opcionális negyedik mező az első mezőben megadott ügyfél részéről elfogadható IP címek listáját tartalmazza. A címek megadhatók pontozott négyes jelöléssel vagy a gazdagép neveként, amelyet a feloldó alakít IP címmé. Ha például a **c3po** az IPCP egyeztetéskor olyan IP cím használatát ajánlja, amely nem szerepel a listában, a kérést eluta-

* Az idézőjelek nem részei a titkos kulcsnak; céljuk csupán a whitespace karakterek megőrzése a kulcson belül.

** A gazdagép neve a CHAP felszólító karakterláncból, a titkos kulcs elnevezése (Use The Source Luke) pedig az open source világából származik. Az utóbbi a Csillagok háborúja híres mondata, a „Use the Force Luke" ferdítése a force-source szójátékkal – *a lektor.*

*** Ez a „Use The Source Luke"-hoz hasonló újabb szójáték a Csillagok háborúja másik robot főszereplője, az R2D2 robot nevével – *a lektor.*

sítjuk és az IPCP-t lezárjuk. Az előbbi mintaállomány alapján a **c3po** csak saját IP címét használhatja. Ha a cím mező üres, bármely cím elfogadható; a „-" érték teljesen kizárja az IP használatát az adott ügyféllel.

A minta *chap-secrets* állomány harmadik sora lehetővé teszi, hogy bármely gazdagép PPP kapcsolatot hozzon létre a **vlager** géppel, mert a * karaktert tartalmazó ügyfél- vagy kiszolgálómező bármely gazdagépnek megfelel. Az egyetlen feltétel, hogy a kapcsolódó gazdagép ismerje a titkos kulcsot, és a **pub.vbrew.com** tartományhoz tartozó IP címet használja. A helyettesítő karaktert tartalmazó bejegyzések a titkos állományban bárhol lehetnek, mivel a *pppd* mindig a kiszolgáló/ügyfél párnak leginkább megfelelő bejegyzést választja.

A gazdagépnevek kialakításakor a *pppd* segítséget igényelhet. Láttuk, hogy a távoli gazdagép nevét mindig a társ adja meg a CHAP felszólítás- vagy válaszcsomagban. A helyi gazdagépnevet alapértelmezés szerint a *gethostname(2)* függvény hívásával kapjuk. Ha a rendszernevet a minősítetlen gazdagépnevünkre állítjuk, a `domain` opcióval kell biztosítanunk a *pppd* számára a tartománynevet:

```
# pppd domain vbrew.com
```

Az utasítással a Sörgyár tartománynevét fűzzük a **vlager** névhez minden hitelesítéssel kapcsolatos tevékenység esetére. A *pppd* program helyi gazdagépnévről alkotott képét ezen kívül a `usehostname` és `name` opciókkal módosíthatjuk. Amikor a helyi IP címet pontozott négyes jelölés helyett a *`helyi:távoli`* argumentumokkal adjuk meg a parancssorban, a *pppd* a *`helyi`* nevet használja lokális névként.

A PAP titkos állománya

A PAP titkos állománya nagyon hasonló a CHAP állományához. Az első két mező mindig egy felhasználó és egy kiszolgáló nevét tartalmazza; a harmadik tárolja a PAP titkos kulcsot. Amikor a távoli gazdagép elküldi a hitelesítési információt, a *pppd* azt a bejegyzést használja fel, amelyben a kiszolgálómező megegyezik a helyi gazdagépnévvel, és a felhasználói fiók megegyezik a kérésben elküldött felhasználónévvel. Amikor nekünk kell magunkat hitelesítenünk, a *pppd* azt a titkos kulcsot alkalmazza, melyben a felhasználómező megegyezik a helyi felhasználónévvel, míg a kiszolgálómező megegyezik a távoli gazdagép nevével.

Lássunk egy példát a PAP titkos állományára!

```
# /etc/ppp/pap-secrets
#
# felhasználó   kiszolgáló      titkos kulcs     cím
vlager-pap      c3po            cresspahl        vlager.vbrew.com
c3po            vlager          DonaldGNUth      c3po.lucas.com
```

Az első sor feladata, hogy azonosítson bennünket a **c3po** géppel folytatott kommunikációban. A második sor azt írja le, hogyan igazolhatja magát előttünk a **c3po** nevű gép.

Az első oszlopban a `vlager-pap` név az a felhasználói név, amelyet a **c3po** számára elküldünk. Alapértelmezés szerint a *pppd* a helyi gazdagépnevet alkalmazza felhasz-

nálói névként, azonban ettől eltérő nevet is megadhatunk a `user` opció, majd a név beírásával.

Amikor a *pppd* kiválaszt egy bejegyzést a *pap-secrets* állományból, hogy azzal azonosítson bennünket a távoli gazdagépnél, ismernie kell a távoli gazdagép nevét. Mivel ezt kitalálni semmiképp nem tudja, nekünk kell megadnunk a parancssorban a `remotname` kulcsszó, majd a társgép nevének beírásával. Az előbbi bejegyzéssel tehát úgy azonosíthatjuk magunkat a **c3po** számára, hogy a következő opciót fűzzük a *pppd* parancssorához

```
# pppd ... remotename c3po user vlager-pap
```

A PAP titkos állományának negyedik mezőjében (és minden további mezőben) az adott gazdagép számára engedélyezett IP címeket adhatjuk meg csakúgy, mint a CHAP titkos állományában. A társ csak ebből a listából kérelmezhet címet. A mintaállomány azon bejegyzése, melyet a **c3po** használ majd a híváskor – az a sor, ahol a **c3po** az ügyfél –, csupán saját, valódi IP címének használatát engedélyezi, semmi mást.

A PAP meglehetősen gyenge hitelesítési eljárás, ezért ha lehet, válasszuk a CHAP hitelesítést. Ezért most nem tárgyaljuk a PAP hitelesítést részletesebben; az érdeklődők a PAP lehetőségeiről bővebben a *pppd(8)* súgóoldalakon olvashatnak.

A PPP beállítási hibáinak kezelése

Alapértelmezés szerint a *pppd* a figyelmeztető és hibaüzeneteket a *syslog* `daemon` kategóriába jegyzi. Az üzeneteket átirányíthatjuk egy állományba vagy akár a konzolra, ha egy bejegyzést helyezünk el a *syslog.conf* állományban; egyébként a *syslog* elveti azokat. A következő bejegyzés minden üzenetet a */var/log/ppp-log* állományba irányít át: *

```
daemon.*                /var/log/ppp-log
```

Amennyiben a PPP konfigurációnk nem üzemel megfelelően, itt kereshetjük a hiba okát. Ha a napló üzenetei sem nyújtanak támpontot, a `debug` opcióval további adatokat kaphatunk a hibakereséshez. Az opció hatására a *pppd* valamennyi elküldött és fogadott vezérlőcsomag tartalmát a syslog naplóba jegyzi. Az üzenetek ezután a `daemon` kategóriába kerülnek.

Egy hiba felderítésének végső és legdrasztikusabb eszköze a rendszermagszintű hibakeresés bekapcsolása, amit a *pppd* `kdebug` opciójának megadásával tehetünk meg. Az opció után egy numerikus argumentum áll, amely a következő értékek összege: 1 – az általános hibakeresési üzenetek, 2 – az összes bejövő HDLC keret tartalmának kiíratása, és 4 – az összes kimenő HDLC keret kiíratása. A rendszermag hibakeresési üzeneteihez úgy férhetünk hozzá, ha futtatjuk a *syslogd* daemont, amely a */proc/kmsg* állományt olvassa, vagy a *klogd* daemont. Mindkét daemon a syslog rendszermag szolgáltatáshoz irányítja a rendszermag hibakeresési üzeneteit.

* Az itt következő sor minden olyan kiszolgáló üzeneteit a ppp-log fájlba küldi, amely a daemon kategóriában kíván naplózni – *a lektor.*

Haladó PPP beállítások

Noha a PPP-t általában olyan hálózat betárcsázására használjuk, mint például az internet, egyes felhasználóknak ennél magasabb elvárásai lehetnek. Ebben a részben a Linuxon elérhető haladó PPP konfiguráció néhány példájával ismerkedünk meg.

A PPP kiszolgáló

A *pppd* kiszolgálóként is üzemelhet, és ehhez mindössze egy soros tty eszközt kell úgy beállítanunk, hogy a bejövő adathívás fogadásakor a megfelelő opciókkal elindítsa a *pppd* daemont. Ennek egyik módja, hogy létrehozunk egy különleges felhasználót (special account), például **ppp** néven, és bejelentkezési héjként olyan szkriptet vagy programot adunk meg, amely a megfelelő opciókkal elindítja a *pppd* programot. Ha szeretnénk PAP vagy CHAP hitelesítést alkalmazni, a modem támogatására használhatjuk a *mgetty* programot és „/AutoPPP/" szolgáltatását.

A bejelentkezési eljárással úgy hozhatunk létre kiszolgálót, hogy a következő sort elhelyezzük */etc/passwd* állományunkban:*

```
ppp:x:500:200:Public PPP Account:/tmp:/etc/ppp/ppplogin
```

Amennyiben rendszerünk az árnyékjelszavakat is kezeli, az */etc/shadow* állományhoz is fűzünk egy bejegyzést:

```
ppp:!:10913:0:99999:7:::
```

Természetesen a megadott UID és GID attól függ, melyik felhasználót szeretnénk a kapcsolat tulajdonosának megtenni, és hogyan hoztuk létre. Az említett felhasználóhoz a jelszót is be kell állítanunk a *passwd* utasítással.

A *ppplogin* szkript így nézhet ki:

```
#!/bin/sh
# ppplogin - szkript a pppd indításához bejelentkezéskor
mesg n
stty -echo
exec pppd -detach silent modem crtscts
```

A *mesg* parancs megakadályozza, hogy más felhasználók a tty eszközre írjanak, például a *write* utasítással. Az *stty* parancs a karaktervisszhangozást kapcsolja ki, és igen fontos szerepe van: ha nem adjuk meg, akkor mindent, amit a társgép küld számunkra, visszhang módjára visszaküldünk neki. A megadott opciók közül a legfontosabb *pppd* opció a `-detach`, hiszen ezzel akadályozzuk meg, hogy a *pppd* leváljon a vezérlő tty eszközről. Ha elhagynánk, a *pppd* futása a háttérben folytatódna, és a héjszkript kilépne. Ennek következtében a soros vonal megszakadna, és a kapcsolat megszűnne. A `silent` opció

* A művelet egyszerűbben elvégezhető a *useradd* vagy *adduser* segédprogramokkal, ha rendelkezünk velük.

hatására a *pppd* addig nem kezd küldeni, amíg csomagot nem kap a hívó rendszertől. Az opcióval elkerülhető az átviteli időkorlát túllépése olyan esetekben, amikor a hívó rendszeren a PPP ügyfél csak lassan indul el. A `modem` opció hatására a *pppd* kezeli a soros port modemvezérlő vonalait. Amikor a *pppd* programot modemmel használjuk, mindig kapcsoljuk be ezt az opciót. A `crtscts` opció a hardver kézfogást kapcsolja be.

Mindemellett szükség lehet valamiféle hitelesítés alkalmazására is – például úgy, hogy a globális opcióállományban vagy a *pppd* parancssorában megadjuk az `auth` kulcsszót. A súgóoldalakon az egyes hitelesítési protokollok ki- és bekapcsolásáról részletesen olvashatunk.

Az *mgetty* esetén csupán azt kell beállítanunk, hogy a program támogassa a soros eszközt, amelyre modemünk kapcsolódik (a részleteket megtaláljuk a 3. fejezetben), majd be kell állítanunk a *pppd* daemont a PAP vagy CHAP hitelesítéshez – elhelyezve a megfelelő opciókat az *options* állományban –, és végül a következő kódrészletet kell */etc/mgetty/login.conf* állományunkhoz fűznünk:

```
# Az mgetty beállítása a bejövő PPP hívások automatikus érzékelésére és
# a pppd daemon indítására a kapcsolat kezeléséhez.
#
/AutoPPP/ -     ppp   /usr/sbin/pppd auth -chap +pap login
```

Az első mező egy ritka varázsszó, melyet arra használunk, hogy észleljük, ha egy bejövő hívás PPP hívás. Ügyeljünk a kis- és nagybetűkre, mert a kifejezés érzékeny rá. A harmadik oszlop a bejelentkezéskor a *who* listában megjelenő felhasználónév. A sor maradék része a futtatni kívánt parancs. Példánkban PAP hitelesítést írtunk elő, kikapcsoltuk a CHAP hitelesítést, és megadtuk, hogy a felhasználók hitelesítése a rendszer *passwd* állománya alapján történjék. Ez a legtöbb igényt kielégíti. Ne feledjük továbbá, hogy az opciókat az *options* állományban vagy a parancssorban is megadhatjuk.

Lássunk most egy rövid ellenőrző listát, melynek lépéseit követve működésre bírhatjuk a PPP behívó programot rendszerünkön. Győződjünk meg arról, hogy az egyes lépések hibátlanul működnek, mielőtt továbblépnénk a következőre:

1. Állítsuk a modemet automata válasz üzemmódra. Hayes kompatibilis modemeken ehhez például az `ATS0=3` utasítás szükséges. Amennyiben az *mgetty* daemont használjuk, erre nincs szükség.
2. Állítsuk be a soros eszközt egy *getty* típusú utasítással úgy, hogy válaszoljon a bejövő hívásokra. A *getty* széles körben elterjedt változata az *mgetty*.
3. Vegyük fontolóra a hitelesítés kérdését. A hívók a PAP, a CHAP vagy a rendszer bejelentkezési prompt segítségével hitelesítik magukat?
4. Állítsuk be kiszolgálónak a *pppd*-t, ahogy ebben a részben láttuk.
5. Vegyük fontolóra az útválasztást. Nekünk kell hálózati útvonalat biztosítanunk a hívókhoz? Az útválasztást az *ip-up* szkripttel valósíthatjuk meg.

Betárcsázás igény szerint

Amikor IP forgalmat szeretnénk lebonyolítani a kapcsolaton keresztül, *igény szerinti tárcsázással* utasíthatjuk a telefonos modemünket betárcsázásra és a kapcsolat létrehozására a távoli gazdagéphez. Az igény szerinti tárcsázás leginkább akkor hasznos, ha tele-

fonvonalunkat nem hagyhatjuk állandóan az internetszolgáltatóra kapcsolva, mert például a helyi hívások időtartama szerint fizetünk. Ilyenkor jobban járunk, ha a telefonvonalat csak akkor kapcsoljuk be, ha valóban szükségünk van a kapcsolatra, és kikapcsoljuk, ha nem használjuk az internetet.

Régebben a Linux megoldások a *diald* parancsot használták, és ez jól is működött, ellenben a beállítása meglehetős hozzáértést követelt. A PPP daemon a 2.3.0 verziótól kezdve támogatja az igény szerinti tárcsázást, és a beállítása is roppant egyszerű.

A *pppd* igény szerinti tárcsázásra történő beállításához csupán bizonyos opciókat szükséges megadnunk az *options* állományban, illetve a *pppd* parancssorában. A 6.2. táblázat az igény szerinti tárcsázással kapcsolatos opciókat foglalja össze.

6.2. táblázat. Az igény szerinti tárcsázással kapcsolatos opciók

Opció	*Leírás*
`demand`	Az opció előírja, hogy a PPP kapcsolat igény szerinti tárcsázás üzemmódba kerül. A PPP hálózati eszköz létrejön, de a *connect* parancs mindaddig nem hajtódik végre, amíg a helyi gazdagép egy datagramot nem küld. Az igény szerinti tárcsázáshoz az opció megadása kötelező.
`active-filter` *kifejezés*	Az opció előírja, hogy milyen adatcsomagokat tekintünk aktív forgalomnak. Minden, a megadott szabálynak megfelelő forgalom újraindítja az igény szerinti tárcsázás időkorlátját mérő számlálót, így biztosítva, hogy a *pppd* ismét várakozik, mielőtt lezárná a kapcsolatot. A szűrő (filter) szintaxisa a *tcpdump* parancsét követi. Az alapértelmezett szűrő minden datagramnak megfelel.
`holdoff` *n*	Az opció előírja, hogy legkevesebb hány másodpercig kívánunk várakozni az újrakapcsolódás előtt, ha a kapcsolat megszakadt. Ha a kapcsolatban hiba adódik, miközben a *pppd* szerint a kapcsolat aktív volt, az újrakapcsolódásra ennyi idő után kerülhet sor. Az időzítés nem vonatkozik arra az esetre, ha a kapcsolat az időkorlát túllépése miatt szakadt meg.
`idle` *n*	Ha az opciót megadjuk, a *pppd* megszakítja a kapcsolatot, valahányszor az időzítő lejár. Az időkorlátot másodpercben adjuk meg. Minden új aktív adatcsomag újraindítja az időzítőt.

Ennek alapján egy egyszerű igény szerinti tárcsázás beállítása valahogy így festene:

```
demand
holdoff 60
idle 180
```

A kód bekapcsolja az igény szerinti tárcsázás üzemmódot, a kapcsolat megszakadását követően 60 másodpercet várakozik az újrakapcsolódás előtt, és megszakítja a kapcsolatot 180 másodperc után, ha nincs aktív adat a kapcsolaton.

Állandó tárcsázás

Az *állandó tárcsázás* azok számára hasznos, akiknek folyamatos betárcsázó kapcsolatuk van egy hálózattal. Az igény szerinti és az állandó tárcsázás között hajszálnyi a különbség. Állandó tárcsázás esetén a kapcsolat a PPP daemon indulásakor automatikusan létrejön. Az „állandóság" akkor nyilvánul meg, ha a kapcsolatot biztosító telefonhívás összeomlik: az állandó tárcsázással a kapcsolat mindig elérhető, mert hiba esetén a kapcsolat automatikusan újra felépül.

Szerencsés esetben nem kell telefonhívásainkért fizetnünk, mert például helyi ingyenes hívásnak számítanak, vagy a cégünk fizeti a számlát. Az állandó tárcsázás opció ilyen esetben rendkívül hasznos. Ha viszont fizetünk a telefonhívásainkért, legyünk óvatosak! Ha a számlát a hívásidő alapján kapjuk, nem biztos, hogy jól járunk az állandó tárcsázással, kivéve, ha a nap huszonnégy órájában használjuk a kapcsolatot. Amennyiben fizetünk ugyan a hívásokért, de nem az időtartam alapján, óvakodjunk az olyan helyzetektől, amikor a modem újra és újra tárcsáz a végtelenségig. A *pppd* daemon biztosít egy lehetőséget, amellyel az ilyen hibák következményeit mérsékelhetjük.

Az állandó tárcsázás elindításához helyezzük el a `persist` opciót az egyik *pppd* opcióállományban. Ez önmagában elegendő ahhoz, hogy a *pppd* automatikusan futtassa a `connect` opcióban megadott parancsot a kapcsolat ismételt felépítéséhez, ha az megszakadt. Ha tartunk tőle, hogy a modem majd túl gyakran próbálkozik az újrahívással (a kapcsolat másik végén lévő modem vagy kiszolgáló hibája miatt), a `holdoff` opcióval megadhatjuk, hogy mennyi ideig várakozzon a *pppd*, mielőtt ismételten kísérletet tesz a kapcsolódásra. Továbbra is fizetnünk kell ugyan a hibákból fakadó elvesztegetett hívások után, de legalább az egyes hibákból fakadó „költséget" csökkenthetjük.

Lássunk egy jellemző kialakítást, amely tartalmazza az állandó tárcsázás opcióit:

```
persist
holdoff 300
```

A `holdoff` időt másodpercben adjuk meg. Példánkban a *pppd* teljes öt percet várakozik, mielőtt a vonal megszakadását követően ismételten tárcsázna.

Az állandó tárcsázást az igény szerinti tárcsázással kombinálhatjuk is, ha az `idle` segítségével megszakítjuk a kapcsolatot, amikor egy adott ideig nincs rajta forgalom. Kétséges, hogy sokan vállalkoznának erre, de ha valakit mégis érdekel, a *pppd* súgóoldalon megtalálhatja az elv leírását.

PPPoE lehetőségek a Linuxon

A PPPoE szerepe a közelmúltban megnőtt, minthogy sok DSL szolgáltató ezt a kapcsolati eljárást választja. A Linux-felhasználóknak szerencsére ezen a téren számos lehetőségük van, és legtöbbjüket igen egyszerű beállítani. A PPPoE eljárásban egyébként semmi új nincs: ugyanaz, mint a betárcsázó kapcsolaton alkalmazott PPP, ezúttal Ethernet hálózaton.

Tételezzük fel, hogy DSL modemünk és felszerelésünk beállítása megtörtént, és készen állnak a bevetésre. Arról, hogy ezt hogyan érhetjük el, David Fannin és Hal Burgiss

kiváló Linux DSL HOWTO leírásából tájékozódhatunk (*http://www.tldp.org/HOWTO/DSL-HOWTO*). Feltételezzük továbbá, hogy Ethernet kártyánk a helyén van és üzemképes.

A legtöbb DSL környezetben a DSL modemet hídként konfiguráljuk, vagyis nincsen IP címe. Következésképpen kiszolgálónkat WAN IP címmel állítjuk be. A WAN interfész bekapcsolása előtt győződjünk meg róla, hogy számítógépünk valamennyi figyelő szolgáltatását frissítettük. Emellett vegyük fontolóra az iptables vagy más tűzfal telepítését. A biztonság, amikor közvetlenül kapcsolódunk az internetre, rendkívüli fontos. A beszámolókból kiderül, hogy a nem frissített Linux disztribúciók csupán néhány óra túlélésben reménykedhetnek az interneten, mielőtt a rendszert kompromittálják. Tegyünk meg minden tőlünk telhetőt, hogy velünk ez ne fordulhasson elő!

A PPPoE ügyfelek

A PPPoE beállításának első lépése egy PPPoE ügyfél beszerzése. Különféle ügyfelek állnak rendelkezésre, többek között a Roaring Penguin ügyfele, amely nagy népszerűségnek örvend a felhasználók és szolgáltatók körében. A program letölthető a *http://www.roaringpenguin.com* címről, akár forrásállományként, akár előre fordított RPM állományként. Letöltése és fordítása, illetve telepítése után a szoftver készen áll a beállításra. Az ügyfélprogram egy rendkívül könnyen kezelhető konfigurációs szkripttel érkezik, neve *adsl-setup*. A szkript különböző kérdéseket tesz fel rendszerünkről, hálózatunkról és a PPPoE felhasználói adatokról. Egyes esetekben már a válasszal is készen áll, csupán megerősítést vár tőlünk!

Noha a szkript rendkívül hasznos, azért nem tökéletes, így a manuális beállítást most részletesen is átvesszük. Ez már csak azért is hasznos, különösen a hálózati adminisztrátor szemszögéből, mert érdemes alaposan megismernünk a szoftver beállítási lehetőségeit, hátha a jövőben gondunk lesz vele.

A PPPoE ügyfél kézi beállítása

Az ügyfél beállítása valóban egyszerű, különösen ha korábban már készítettünk standard PPP konfigurációt. Mindenekelőtt az */etc/ppp/pap-secrets* állományt módosítjuk. Az alapértelmezett értékeket saját PPPoE felhasználói nevünkkel és jelszavunkkal helyettesítjük. Az állomány valahogy így néz ki:

```
#Felhasználó            #Kiszolgáló     #jelszó       #IP
groucho@dslcompany.to   *               jelszavam     *
```

Most nyissuk meg az */etc/ppp/ppoe.conf* állományt a szövegszerkesztőnkben. Adjuk meg a WAN interfészünk nevét, valamint a PPPoE felhasználói nevünket. Az állomány releváns része így fest:

```
# ADSL modemhez kapcsolódó Ethernet kártya
ETH=eth0

# ADSL felhasználónév.  Esetleg meg kell adnunk: "@provider.com"
USER=groucho@dslcompany.to
```

Az állomány további beállítási lehetőségeket is tartalmaz. Ha nem vagyunk benne teljesen biztosak, hogy szükség van a módosításukra, jobb, ha nem nyúlunk hozzájuk. Amennyiben mégis a módosítás mellett döntünk, olvassuk el a PPP súgóoldalát.

Végül, ha még nem állítottuk be a DNS kiszolgálónkat az */etc/resolv.conf* állományban, itt az ideje, hogy megtegyük. A DNS beállításáról részletes leírást az 5. fejezetben olvashatunk.

A beállításokat elvégezve most már ellenőrizhetjük, működik-e a kapcsolatunk. Az *adsl-start* szkript kimondottan ezt a célt szolgálja. Meghívhatjuk a parancssorból, de érdemes inkább a rendszerindító szkriptek közé helyeznünk. Ennek menete az egyes terjesztéseken eltérő lehet. Az indítószkriptek telepítését illetően olvassuk el a terjesztésünkhöz kapott dokumentációt.

Ha az indítószkript hiba nélkül lefutott, máris az interneten vagyunk! Ennek ellenőrzésére a legegyszerűbb eljárás, ha a ping programot egy biztosan válaszoló célra állítva futtatjuk. Ha a következőhöz hasonló kimentet kapunk, a művelet sikerült:

```
vlager# ping www.google.com
PING www.google.akadns.net (66.102.7.99) 56(84) bytes of data.
64 bytes from 66.102.7.99: icmp_seq=1 ttl=245 time=5.94 ms
64 bytes from 66.102.7.99: icmp_seq=2 ttl=245 time=5.02 ms
64 bytes from 66.102.7.99: icmp_seq=3 ttl=245 time=5.02 ms
ctrl-c
-- www.google.akadns.net ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 5.028/5.333/5.945/0.440 ms
vlager#
```

Az *ifconfig* segítségével a beállításokat is megvizsgálhatjuk:

```
vlager# ifconfig -a
eth0    Link encap:Ethernet  HWaddr 00:08:02:F0:BB:0E
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:8701578 errors:6090 dropped:0 overruns:0 frame:5916
        TX packets:3888596 errors:0 dropped:0 overruns:0 carrier:0
        collisions:6289 txqueuelen:100
        RX bytes:1941625928 (1851.6 Mb)  TX bytes:1481305134 (1412.6 Mb)
        Interrupt:30

eth1    Link encap:Ethernet  HWaddr 00:90:27:FE:02:A0
        inet addr:10.10.0.254  Bcast:10.10.0.255  Mask:255.255.255.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:48920435 errors:0 dropped:0 overruns:0 frame:0
        TX packets:55211769 errors:0 dropped:0 overruns:2 carrier:9
        collisions:367030 txqueuelen:100
        RX bytes:2018181326 (1924.6 Mb)  TX bytes:1564406617 (1491.9 Mb)
        Interrupt:10 Base address:0x4000
```

```
ppp0  Link encap:Point-to-Point Protocol
      inet addr: 64.168.44.33 P-t-P:64.168.44.1  Mask:255.255.255.255
      UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1492  Metric:1
      RX packets: 8701576 errors:0 dropped:0 overruns:0 frame:0
      TX packets: 3888594 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:10
```

Amennyiben ennél a pontnál hibát észlelünk, ellenőrizzük a kapcsolatainkat és azt is, hogy a DSL beállítása megfelelő-e. Vessünk továbbá egy pillantást a konfigurációs állományban megadott felhasználói névre és jelszóra is – az egyik leggyakoribb beállítási hiba az elgépelt jelszó!

7

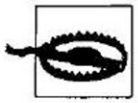
A TCP/IP tűzfal

A biztonság a vállalkozások és az egyének számára is egyre fontosabb. Az internet egyfelől hatékony eszköz, hogy megismertessük magunkat a világgal és megismerjünk másokat, másfelől azonban veszélyeket is hordoz, amilyenekkel korábban még nem találkozhattunk. A számítógépes bűnözés, az információlopás és a rosszindulatú támadások potenciális veszélyt jelentenek valamennyiünk számára.

A fejezetben megnézzük, hogyan állíthatjuk be a Linux rendszer tűzfalát, amelyet a parancsfelület *(iptables)* és a rendszermag alrendszerének nevéről *(netfilter)* egyaránt ismerhetünk. Ez a tűzfal-megvalósítás a 2.4 rendszermagban jelent meg, és alapvetően a 2.6 verzióban is hasonlóan működik.

Amikor egy rosszindulatú támadó hozzáférést szerez egy rendszerhez, megpróbálhatja kitalálni a rendszerjelszavakat, vagy kihasználhatja egyes programok hibáit, sajátos viselkedését, hogy működő felhasználói fiókhoz jusson a gazdagépen. Amint be tud jelentkezni a gazdagépre, hozzáférhet fontos adatokhoz. A vállalkozások világában például a marketingtervek, a fejlesztési tervek vagy az ügyféladatbázis adatainak ellopása, törlése vagy módosítása jelentős veszteséget okozhat a cégeknek.

Az ilyen súlyos károkat a legbiztosabban úgy kerülhetjük el, ha megakadályozzuk, hogy jogosulatlan személyek hálózati hozzáférést szerezzenek a gazdagépekhez. És itt lépnek a képbe a tűzfalak.



A biztonságos tűzfalak kialakítása művészet. Alaposan ismernünk kell hozzá az eljárásokat, de legalább ilyen fontos, hogy tisztában legyünk a tűzfal tervezését inspiráló védelmi és technikai filozófiával. A kötetben nem szólhatunk mindenről, amit tudnunk szükséges, ezért feltétlenül ajánlott, hogy mielőtt egy tűzfalra bíznánk magunkat - az itt bemutatottakat is beleértve -, tájékozódjunk a témáról.

A fejezetben a Linux rendszerekkel kapcsolatos műszaki kérdéseket nézzük át. Később bemutatunk egy minta tűzfal-konfigurációt is, amely megfelelő alap lehet saját tűzfalunk kialakításához. Mindazonáltal, mint minden biztonsági vonatkozású kérdésben, alaposan ismernünk kell a témát ahhoz, hogy az itt bemutatott példát a saját igényeinkhez alakíthassuk és ellenőrizhessük a tűzfal működését. Kétszer is nézzük át, és győződjünk meg róla, hogy a működést pontosan értjük; ezután már módosíthatjuk a céljainknak megfelelően. A biztonságunk alapja a megértés.

A támadások fajtái

Hálózati adminisztrátorként fontos ismernünk a számítógépes biztonságot fenyegető támadások természetét. Most röviden átnézzük a támadások főbb fajtáit, hogy érzékeltessük, mi ellen véd majd bennünket a Linux IP tűzfal. Az is fontos azonban, hogy további ismereteket szerezzünk, mert csak így védhetjük meg hálózatunkat az egyéb típusú támadásokkal szemben. Lássuk tehát a legelterjedtebb támadások fajtáit és a lehetséges védekezési módokat!

Jogosulatlan hozzáférés

A jogosulatlan hozzáférés egyszerűen azt jelenti, hogy olyan személyek kapcsolódnak számítógépünkre és használják szolgáltatásait, akiknek ez tilos; például külső személyek, akik nem tartoznak a vállalathoz, megpróbálják elérni a cég beléptető gazdagépét vagy NFS kiszolgálónkat.

Számos különféle eljárással védekezhetünk az ilyen támadásokkal szemben. Mindössze gondosan meg kell határoznunk, kik juthatnak hozzáféréshez ezen a szolgáltatásokon keresztül. A jogosultakon kívül mindenki más számára tiltsuk le a hálózati hozzáférést.

A programok ismert hibáinak kihasználása

Egyes programok és hálózati szolgáltatások tervezésekor a készítők nem fordítottak elég figyelmet a biztonságra, és ezért ezek könnyen sebezhetők. Példaként említhetjük a BSD távoli szolgáltatásokat (*rlogin, rexec* stb.).

Az ilyen típusú támadások ellen a legjobb védekezés, ha kikapcsoljuk a sebezhető szolgáltatásokat vagy alternatív megoldásokat keresünk helyettük. Kezdetnek csak olyan szolgáltatásokat telepítsünk, futtassunk és tegyünk elérhetővé, amelyekre feltétlenül szükségünk van. A hálózati szolgáltatásokat először ne is kapcsoljuk be, és csak lépésről lépésre indítsuk el azokat. A *netstat* paranccsal ellenőrizzük, mely portokra figyel a gazdagép, és győződjünk meg arról, hogy a lista a lehető legkevesebb portot tartalmazza, és mindegyik portról pontosan tudjuk, mi a célja. A tűzfalként szolgáló gazdagépen ne futtassunk hálózati szolgáltatásokat – talán csak a Secure Shell (SSH) lehet kivétel.

Kövessük nyomon a hibaleírásokat tartalmazó adatbázisokat és a hibajavító programok listáit, így a rendszert napra kész állapotban tarthatjuk. A két legnépszerűbb hibaleíró adatbázis a Bugtraq, melyet a *http://www.securityfocus.com/bid* címen találunk (a *http://www.securityfocus.com/rss* oldalon megtudhatjuk, hogyan érhetjük el a Bugtraq adatbázist RSS értesítőn keresztül), és a Common Vulnerabilities and Exposures (CVE) adatbázis a *http://cve.mitre.org/* címen (az RSS-szel kapcsolatban lásd a *http://www.opensec.org/feeds/cve/latest.xml* oldalt). A legtöbb Linux terjesztés eszközöket nyújt a frissítések letöltéséhez és telepítéséhez. A Red Hat segédprogramja a *yum*, a SuSE eszköze a YaST Online Update (YOU), a Debiané pedig az *apt-get* program.

A szolgáltatásmegtagadás célú támadások

A szolgáltatásmegtagadási támadás célja, hogy egy szolgáltatás vagy egy program futását akadályozza, illetve a felhasználókat meggátolja, hogy a szolgáltatást vagy a programot igénybe vegyék. A támadások megvalósíthatók a hálózat szintjén ponto-

Az SSH és az iptables

Az SSH és az *iptables* két egyszerű módszert kínál, hogy a külvilág számára elérhetővé tegyük gazdagépeinket és szolgáltatásainkat anélkül, hogy közvetlenül felfednénk őket. Az első megoldás, hogy SSH-t futtatunk a tűzfalon, és a belső gazdagépek és szolgáltatások elérését az SSH porttovábbítási funkciójával biztosítjuk anélkül, hogy közvetlenül felfednénk őket a külvilág előtt. Ehhez Bob Toxen könyvében, a *Real World Linux Security* (Prentice Hall) második kiadásának 12.1 fejezetében további ötleteket kaphatunk. A második megoldás, hogy az *iptables* Destination Network Address Translation eljárásával a tűzfal nyilvános IP címének különböző portjain több kiszolgáló számára is elérhetővé tesszük az SSH-t, miközben a kapcsolatokat a hálózaton belüli gazdagépekre továbbítjuk. A hálózati címfordítás a 9. fejezet témája.

san megszerkesztett, rosszindulatú csomagok küldésével, amelyek tönkreteszik a hálózati kapcsolatot. De mindez megtörténhet az alkalmazások szintjén is gondosan összeállított parancsok kiadásával, amelyek teljesen lefoglalják az alkalmazást vagy akadályozzák futását.

A szolgáltatásmegtagadás célú támadások kockázatának csökkentéséhez meg kell akadályoznunk, hogy a gyanús hálózati forgalom vagy a gyanús parancsok és kérések eljussanak a gazdagépeinkhez (ehhez a mögöttes protokollokat értő szoftver szükséges, például proxy kiszolgáló). Nagyon hasznos, ha ismerjük a támadási módszerek részleteit, ezért érdemes az egyes új támadásokról szóló publikációkat olvasgatnunk.

A hamisítás

Az ilyen típusú támadás lényege, hogy egy gazdagép vagy alkalmazás másnak adja ki magát, mint ami. A támadó gazdagépe jellemzően a hálózati csomagok IP címeinek meghamisításával ártatlan számítógépnek adja ki magát. Alaposan dokumentált eljárások, például a BSD `rlogin` szolgáltatásának kihasználásával is vissza lehet élni, egy másik gazdagépről érkező TCP kapcsolatot lehet utánozni a TCP szekvenciaszámok találgatásával.

Az ilyen jellegű támadással szemben úgy védekezhetünk, hogy ellenőrizzük a csomagok és parancsok hitelességét (itt egy szűrőkből és proxy kiszolgálókból álló együttes segíthet). Az érvénytelen forráscímű csomagokat ne engedjük át az útválasztókon. Használjunk olyan operációs rendszereket (például Linux rendszert), amely kiszámíthatatlan kapcsolatvezérlő mechanizmust alkalmaz, például TCP szekvenciaszámokat vagy dinamikus portcímkiosztást.

Ha azokat a gazdagépeket, amelyek operációs rendszerei nem biztonságos szekvenciaszám-algoritmust használnak, elhelyezzük egy hálózati címfordítást alkalmazó Linux tűzfal mögé, nagyobb biztonsággal futtathatjuk őket, mert a tűzfal számítógép saját szekvenciaszámozási algoritmusával kommunikál a külvilággal.

Lehallgatás

A legegyszerűbb támadási módszer. A gazdagépet arra programozzák, hogy figyelje és elfogja a nem neki szánt adatokat (ehhez a hálózati interfészét „lehallgató" - promiscuous - üzemmódba kapcsolják, így a hálózati szegmensen áthaladó összes csomagot figyelhetik). A gondosan megírt lehallgató programokkal a felhasználói bejelentkezést végző hálózati kapcsolatokból kiszűrhetők a felhasználói nevek és a jelszavak. Az üzenetszóró hálózatok, például a nem kapcsolt Ethernet különösen sebezhetők az ilyen típusú támadásokkal szemben, noha a támadáshoz a támadónak fizikai kapcsolatban kell állnia az Ethernet hálózattal. A vezeték nélküli hálózatokkal hasonló gondok vannak, de a veszély még nagyobb, hiszen nincs szükség fizikai kapcsolatra, elegendő, ha a támadó megfelelő közelségben van.

A fenyegetés ellen úgy védekezhetünk, hogy kerüljük az üzenetszóró hálózati eljárásokat és megköveteljük az adatok titkosítását.

A kapcsolt környezetekben már nehezebb, de nem lehetetlen a csomagok kifürkészése. Egyes Ethernet kapcsolókon olyan adminisztratív beállítást vagy akár hibakereső üzemmódot is találhatunk, amelynek aktiválásakor a kapcsoló a csomagokat egy vagy több portjára is átmásolja.

Az IP tűzfalak rendkívül hatásosak a jogosulatlan hozzáféréssel, a hálózati réteg szintjén megvalósított szolgáltatásmegtagadás célú és IP hamisítási támadásokkal szemben. Nem sokat érnek azonban a hallgatózókkal és a hálózati szolgáltatások, illetve programok hibáit kihasználó támadókkal szemben.

Mi a tűzfal?

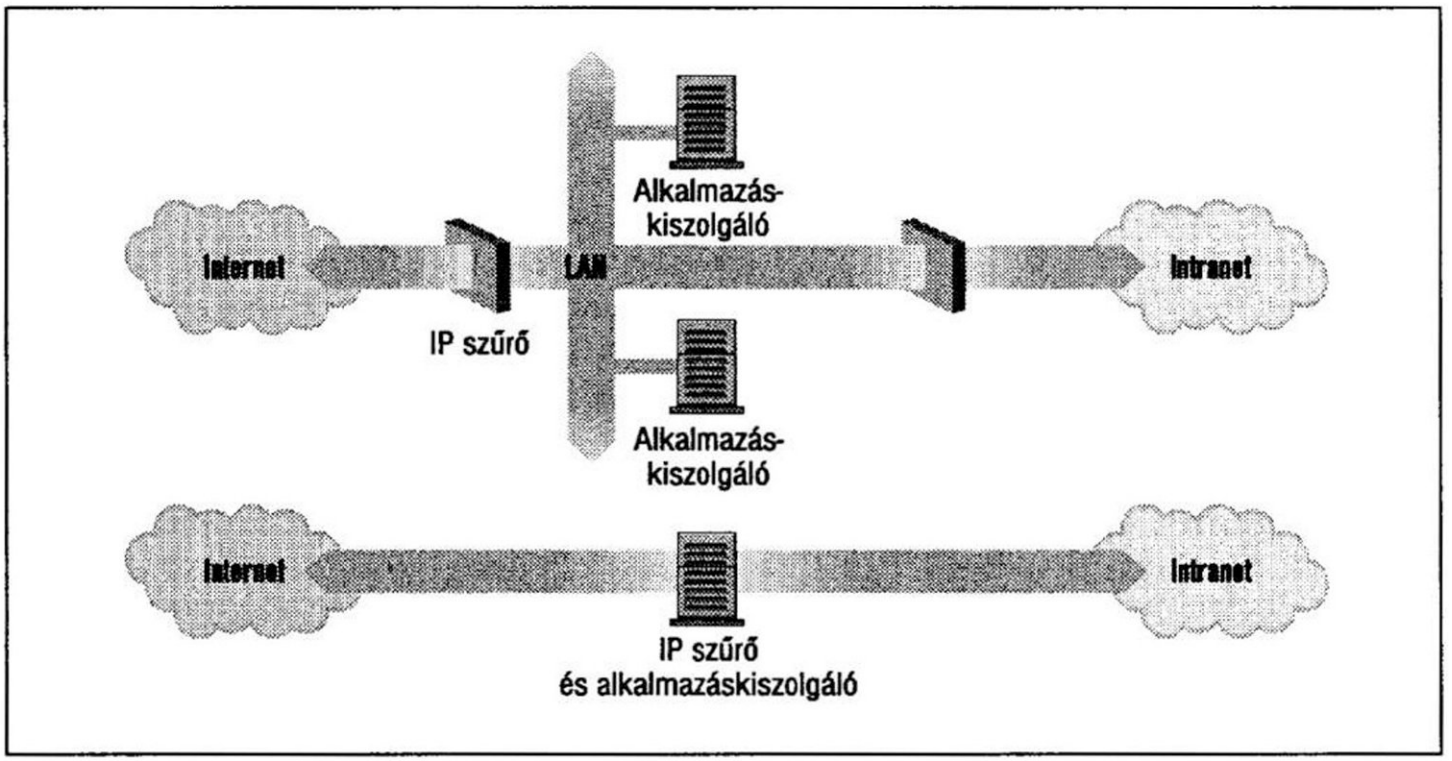
A tűzfal egy megerősített és megbízható gazdagép, amely a több hálózatból (rendszerint egyetlen magánhálózatból és egyetlen nyilvános hálózatból) álló csoport tagjai között a *fojtópont* szerepét játssza.* Az érintett hálózatok között a teljes hálózati forgalom a tűzfalon keresztül bonyolódik. A tűzfal gazdagépen szabályokkal határozzuk meg, milyen hálózati forgalom haladhat át rajta, és milyen forgalmat kell megállítania (válasz nélkül eldobnia) vagy visszautasítania (válasz kíséretében elutasítania). A nagyobb szervezeteknél akár a vállalati hálózaton belül elhelyezett tűzfalakkal is találkozhatunk. Feladatuk, hogy a szervezet egyes érzékeny részeit elkülönítsék más területek felhasználóitól. A számítógépes bűncselekmények nagyon gyakran nem kívülről, hanem a szervezeten belülről indulnak ki.

A tűzfalak létrehozásának számos módja létezik. A legkifinomultabb elrendezés számos önálló gazdagépet foglal magában, és neve *peremhálózat* (perimeter network) vagy *demilitarizált zóna* (DMZ). Két gazdagép „szűrőként" funkcionál (néha *fojtóknak* is hívják őket); szerepük, hogy csak bizonyos típusú hálózati forgalmat engedjenek tovább. A fojtók között helyezkednek el a hálózati kiszolgálók, például e-mail (SMTP) kiszolgálók vagy World Wide Web (HTTP) proxy kiszolgálók. A kialakítás rendkívül biztonságos,

* A tűzfal kifejezés olyan eszköz nevéből származik, amelynek feladata, hogy a tűztől óvjon. A tűzfal tűzálló anyagból készült pajzs, amelyet a tűz lehetséges helye és a védeni kívánt személyek közé helyeznek el.

és segítségével nagyon pontosan szabályozhatjuk a kapcsolódást, mind belülről kifelé, mind kívülről befelé. A nagy szervezetek használhatnak ilyen rendszereket.

A legtöbb esetben azonban a felhasználók az egyéb szolgáltatásokat (például SMTP-t vagy HTTP-t) is biztosító tűzfalakat építenek. Ezek kevésbé biztonságosak, mert a tűzfalgépen futó bármely további szolgáltatás hibájának kihasználása a teljes hálózati biztonságot fenyegetheti. A támadó módosíthatja a tűzfal szabályrendszerét, és ezzel jobb hozzáféréshez juthat, illetve kikapcsolhatja a nyomkövetést, amely egyébként figyelmeztetné a hálózati adminisztrátort a szokatlan tevékenységre. Mindazonáltal ezek a tűzfalak olcsóbbak és könnyebben fenntarthatók, mint az előbb bemutatott kifinomult elrendezés. A 7.1. ábra a két leggyakoribb tűzfal-kialakítást szemlélteti.



7.1. ábra. A tűzfalmodellek két fő osztálya

A Linux rendszermagja olyan beépített tulajdonságkészletet biztosít, amely lehetővé teszi, hogy IP tűzfalként alkalmazzuk. A hálózati implementációhoz tartozó kóddal (a *netfilter* alrendszerrel) az IP csomagokat különféle módszerekkel dolgozhatjuk fel, miközben a felhasználói térben futó mechanizmussal (az iptables paranccsal) megadhatjuk a működést leíró szabályokat. A Linux tűzfal elég rugalmas ahhoz, hogy a 7.1. ábrán bemutatott bármely elrendezésben kiválóan működjön. A Linux tűzfal szoftvere két további hasznos eljárást biztosít, ezeket önálló fejezetekben tárgyaljuk: az IP nyomkövetés a 8. fejezet, míg az IP álcázás és hálózati címfordítás a 9. fejezet témája.

A csomagok feldolgozásának három fő osztálya a szűrés (filtering), a manipulálás (mangling) és a hálózati címfordítás (Network Address Translation, NAT). A szűrés egyszerűen azt jelenti, hogy a csomagfolyam különböző pontjain eldöntjük, hogy átengedjük-e a csomagokat a következő állomásukra. A csomagmanipulálás általános kifejezés és arra utal, hogy a csomagokat módosítjuk, amikor a csomagfolyamban haladnak. A NAT a csomagmódosítás egy különleges alkalmazási formája, amikor a forráscímeket

vagy rendeltetési IP címeket és/vagy portokat módosítjuk, hogy a forgalmat transzparens módon átirányíthassuk.

Mi az IP szűrés?

Az IP szűrés egyszerű mechanizmus, amely eldönti, hogy milyen típusú IP csomagokat dolgozzon fel a rendszer, és milyen csomagokat dobjon el vagy utasítson vissza. Az *eldobás* (drop) alatt azt értjük, hogy a csomagot a rendszer törli és teljesen figyelmen kívül hagyja, mintha soha nem is érkezett volna meg. Az *elutasítás* (reject) azt jelenti, hogy a tűzfal egy ICMP válaszban értesíti a küldőt, hogy miért utasította el a csomagot. Számos különféle feltételhez köthetjük, hogy milyen csomagokat kívánunk kiszűrni. Néhány példa:

- protokoll szerint: TCP, UDP, ICMP stb.;
- portszám szerint (TCP/UDP protokollok esetén);
- a csomag típusa szerint: SYN/ACK, adat, ICMP Echo Request stb.;
- a csomag forrásának címe szerint, vagyis annak alapján, hogy honnan jött a csomag;
- a csomag rendeltetési címe szerint, vagyis annak alapján, hogy hova tart a csomag.

Ezen a ponton fontos felhívnunk a figyelmet, hogy az IP szűrés a hálózati réteg szolgáltatása. Ez azt jelenti, hogy semmit nem tud a hálózati kapcsolatokat használó alkalmazásról, csupán magukat a kapcsolatokat ismeri. Tegyük fel például, hogy az alapértelmezett Telnet porton megtiltjuk, hogy a felhasználók belső hálózatunkhoz férjenek. Ha a feladatot kizárólag az IP szűrésre bízzuk, akkor viszont nem akadályozhatjuk meg, hogy a felhasználók a Telnet programot olyan porton használják, melynek engedélyeztük, hogy áthaladjon a tűzfalon. Az ilyen típusú problémákat úgy oldhatjuk meg, hogy minden szolgáltatáshoz, amely áthaladhat a tűzfalon, proxy kiszolgálót alkalmazunk. A proxy kiszolgálók felismerik a rájuk bízott alkalmazásokat, így megakadályozhatják, hogy azokkal visszaéljenek, és például a Telnet program a World Wide Web porton keresztül kerülje meg a tűzfalat. Ha tűzfalunk támogat egy World Wide Web proxy kiszolgálót, mindig az válaszol a HTTP porton kifelé tartó Telnet kapcsolatokra, és csak a HTTP kéréseket engedi kijutni. Proxy-kiszolgálóprogramok nagy számban léteznek. Egyesek ingyenesek, sok más program viszont kereskedelmi termék. A *Firewall and Proxy Server HOWTO* dokumentáció (amely elérhető a *http://www.tldp.org/HOWTO/Firewall-HOWTO.html* oldalon) a legnépszerűbbekkel foglalkozik; ismertetésük túlmutat könyvünk lehetőségein.

Az IP szűrés szabályrendszere az előbbiekben bemutatott feltételek különféle kombinációjából építkezik. Képzeljük most el, hogy azt szeretnénk, hogy a Virtuális Sörgyáron belül a felhasználók ne érhessék el az internetet korlátlanul, csupán más hálózati helyek webkiszolgálóit használhassák. Ekkor a tűzfalat a következő csomagok továbbítására állítanánk be:

- csomagok, amelyek forráscíme a Virtuális Sörgyár hálózatán van, a rendeltetési címe bármi, a rendeltetési port a 80-as (WWW),
- csomagok, amelyek rendeltetési címe a Virtuális Sörgyár hálózatán van, a rendeltetési port a 80-as (WWW), és a forrás címe bármi.

Vegyük észre, hogy két szabályt alkalmaztunk. Lehetővé kell tennünk, hogy adataink kijussanak, de azt is, hogy a megfelelő válaszok bejussanak. A gyakorlatban, ahogy azt az IP álcázás és hálózati címfordítással foglalkozó fejezetben látni fogjuk (9. fejezet), az *iptables* leegyszerűsíti a folyamatot, mert egyetlen parancsban megadhatjuk mindezt.

A netfilter és az iptables

Paul „Rusty" Russell – miközben a Linux IP tűzfalának korábbi változatán (`ipchains`) dolgozott – úgy döntött, hogy egyszerűbbé teszi az IP tűzfalkezelést. Hozzálátott tehát, hogy a rendszermag tűzfalkezelő kódjának csomagfeldolgozó részét leegyszerűsítse, és végül olyan szűrő keretrendszert hozott létre, amely jobban átlátható és egyben rugalmasabb volt. Az új keretrendszernek a *netfilter* nevet adta.

Míg az *ipchains* az elődjéhez képest *(ipfwadm)* hatalmas előrelépést jelentett a tűzfalszabályok kezelését illetően, a csomagok feldolgozása továbbra is összetett feladat maradt, különösen ha más lehetőségekkel egybevonva használtuk, például az IP álcázással (erről bővebben a 9. fejezetben) vagy a címfordítás egyéb formáival együtt. Az összetettségre részben az ad magyarázatot, hogy az IP álcázást és a NAT-ot az IP tűzfalkezelő kódjától függetlenül fejlesztették és csak később egyesítették őket, tehát a eredetileg nem voltak az IP tűzfal kódjának integráns részei. Ha egy fejlesztő úgy gondolta, hogy ő bizony további lehetőségeket szeretne a csomagfeldolgozó eljáráshoz fűzni, nehezen talált volna olyan helyet, ahová a kódját elhelyezheti, és ehhez még akkor is meg kellett volna változtatnia a rendszermag kódját.

A *netfilter* a korábbi megoldások összetettségére és merevségére is megoldást kínál, mert egy általános keretrendszert valósít meg a rendszermagban, amely gondosan követi a csomagok feldolgozásának menetét, és lehetővé teszi a szűrés szabályrendszerének kiterjesztését anélkül, hogy a rendszermag módosítására szükség volna. A *Linux 2.4 Packet Filtering HOWTO* (elérhető a *http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html* címen) részletes listát közöl a változtatásokról, ezért foglalkozzunk most inkább a gyakorlati kérdésekkel.

Egy Linux IP tűzfal építéséhez szükségünk van egy IP tűzfal *(netfilter)* támogatással fordított rendszermagra és az *iptables* felhasználói konfigurációs segédprogramra. A *netfilter* a Linux csomagkezelő folyamatának alapos átszervezésével jött létre. A *netfilter* visszafelé közvetlenül kompatibilis mindkét korábbi linuxos tűzfalkezelő megoldással *(ipfwadm* és *ipchains),* és egy új parancsot is biztosít: az *iptables* parancsot. A könyvben csak ezzel a paranccsal foglalkozunk, azonban ha szeretnénk megismerni az *ipfwadm* vagy az *ipchains* szabályait is, megtalálhatjuk kötetünk korábbi kiadásaiban.

Minta iptables parancsok

Az *iptables* architektúra a hálózati csomagok feldolgozási szabályait feladatuk (csomagszűrés, hálózati címfordítás és más csomagmanipulációk) alapján *táblázatokba* csoportosítja, amelyek mindegyike feldolgozási *szabályokból* álló *láncokat* (sorozatokat) tartalmaz. A szabályok *illesztéseket* és *célokat* írnak le. Az előbbiek határozzák meg, hogy mely csomagokra illeszkedik a szabály, az utóbbiak pedig azt mondják meg, mi történjen az illeszkedő (a szabálynak megfelelő) csomagokkal.

Az *iptables* az OSI Layer 3 (Network) rétegben működik. Az OSI Layer 2 (Link) réteghez más eljárások használatosak, például az *ebtables* (Ethernet Bridge Tables). Erről bővebben a *http://ebtables.sourceforge.net/* címen olvashatunk.

Most az *iptables* felhasználására látunk néhány, alapos magyarázattal kísért példát. További részleteket találunk a fejezet *Az iptables alapfogalmai* című részében.

Példa a csomagszűrésre

A következő parancs a *Mi az IP szűrés?* című részben megfogalmazott szabályokat ülteti át a gyakorlatba, és a tűzfalban arra használhatjuk, hogy kiszűrjünk minden nem HTTP forgalmat. A parancs feltételezi, hogy az `eth0` a belső Ethernet interfész, míg az `eth1` az internethez kapcsolódó Ethernet interfész.

```
iptables -t filter -P FORWARD DROP
iptables -t filter -A FORWARD -i eth0 -p tcp —dport 80 -j ACCEPT
iptables -t filter -A FORWARD -i eth1 -p tcp —sport 80 -j ACCEPT
```

Az első parancs a `filter` táblázat `FORWARD` láncát úgy állítja be, hogy alapértelmezés szerint minden csomagot eldobjon (`DROP`). A második parancs jelentése „engedélyezz minden kimenő HTTP kérést". A parancs értelmezéséhez a 7.1. táblázat nyújt segítséget. A harmadik parancs hasonló: „engedélyezz minden bejövő HTTP választ".

7.1. táblázat. A minta iptables parancs argumentumainak leírása

Komponens	*Leírás*
`-t filter`	A művelet a `filter` táblázatra vonatkozik (egyébként alapértelmezett)...
`-A FORWARD`	...és a következő szabályt fűzi annak `FORWARD` láncához.
`-i eth0`	Ha az `eth0` belső hálózati interfészre érkező csomagok...
`-p tcp`	...a `tcp` (TCP/IP) protokollt használják...
`--dport 80`	...és a (külső) rendeltetési gazdagép 80-as portjára tartanak, akkor...
`- j ACCEPT`	...fogadd el a csomagokat továbbításra.

Példa az álcázásra

Az előző rész csomagszűrő példája nem használja ki az iptables lehetőségeit. Ha internet interfészünk IP címe dinamikus, jobban járunk, ha álcázást alkalmazunk (az álcázásról bővebben a 9. fejezetben):

```
iptables -t nat -P POSTROUTING DROP
iptables -t nat -A POSTROUTING -o eth1 -p tcp —dport 80 -j MASQUERADE
```

Példa a hálózati címfordításra

A parancsot a tűzfalon használhatjuk a bejövő HTTP forgalom továbbítására a belső hálózat valamely webkiszolgálójához (a hálózati címfordításról bővebben a 9. fejezetben).

```
iptables -t nat -A PREROUTING -i eth1 -p tcp -dport 80 \
  -j DNAT —to-destination 192.168.1.3:8080
```

A 7.2. táblázat az előbbi minta *iptables* parancsot értelmezi.

7.2. táblázat. A minta iptables parancs argumentumainak leírása

Komponens	*Leírás*
`-t nat`	A művelet a `nat` (Network Address Translation) táblázatra vonatkozik...
`-A PREROUTING`	...és a következő szabályt fűzi annak `PREROUTING` láncához.
`-i eth1`	Ha az `eth1` hálózati interfészre érkező csomagok...
`-p tcp`	...a `tcp` (TCP/IP) protokollt használják...
`--dport 80`	...és a helyi `80`-as portra tartanak, akkor...
`- j DNAT`	...ugorj a `DNAT` (Destination Network Address Translation) célhoz...
`--to-destination 192.168.1.3:8080`	...és változtasd a rendeltetési címet 192.168.1.3-ra, a rendeltetési portot pedig `8080`-ra.

Az iptables alapfogalmai

Az *iptables* öt kapcsolódási pontot (hook point) definiál a rendszermag csomagfeldolgozási útvonalában: ezek a `PREROUTING`, az `INPUT`, a `FORWARD`, a `POSTROUTING` és az `OUTPUT`. A kapcsolódási pontokra beépített *láncok* kapcsolódnak; ezek mindegyikéhez meghatározhatunk szabálysorozatokat. Feladatuk, hogy segítségükkel befolyásolhassuk és figyelemmel kísérhessük a csomagfolyamot.

Gyakran hallani ilyesmit: „a *nat* táblázat `PREROUTING` lánca". Ebből azt gondolhatnánk, hogy a láncok a táblázatokhoz tartoznak. A láncok és a táblázatok azonban csak részben felelnek meg egymásnak, és valójában egyik sem „tartozik" a másikhoz. A *láncok* voltaképpen kapcsolódási pontok a csomagfolyamban, a *táblázatok* pedig a lehetséges feldolgozási típusokat adják meg. A 7.2. ábra bemutat néhány érvényes kombinációt, valamint azt a sorrendet, ahogy a rendszeren keresztülhaladó csomagok ezekkel a lehetőségekkel találkoznak.

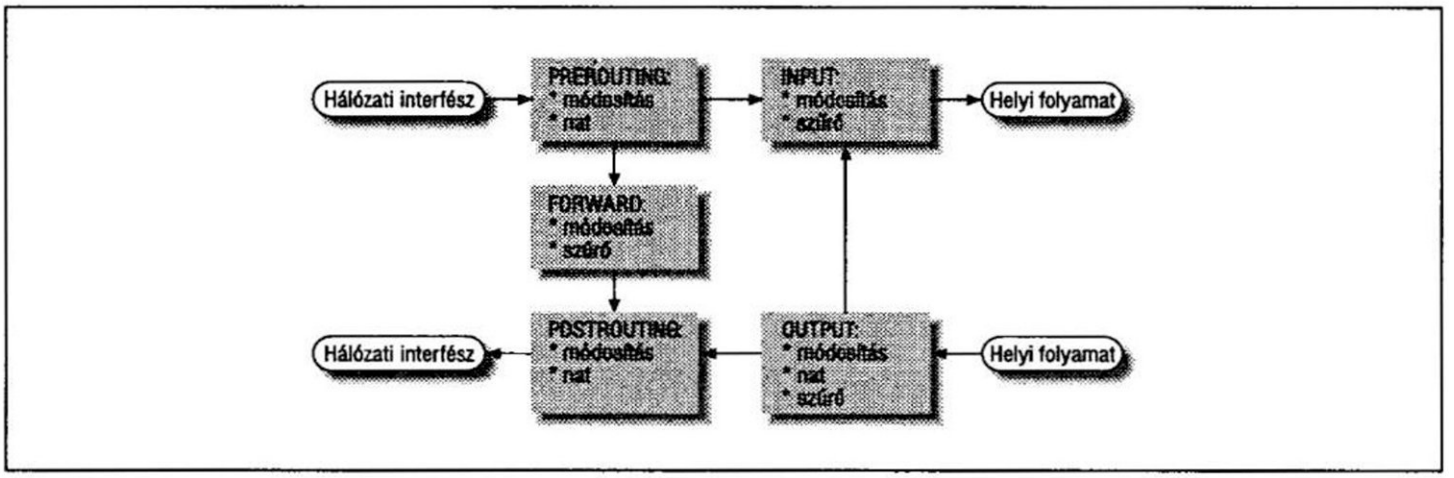
A csomagfolyam

A 7.2. ábra azt szemlélteti, hogyan haladnak keresztül a csomagok a rendszeren. A négyzetek az *iptables* láncokat jelképezik. A négyzetekben láthatjuk azon táblázatokat, amelyek rendelkeznek ilyen lánccal (a meghívásuk sorrendjében). A csomagmódosítás e táblázatlánc-kombinációk mindegyikét magában foglalja.

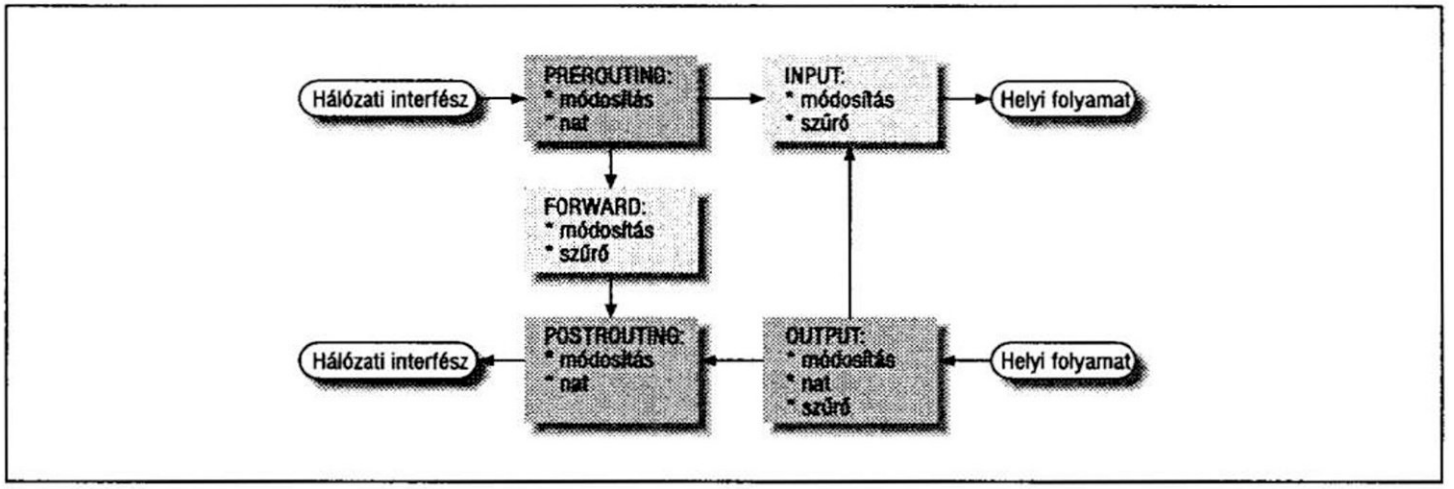
A 7.3. ábrán a szürke négyszögek azokat a láncokat és táblázatokat jelölik, melyek nem vesznek részt a hálózati címfordításban.

A 7.4. ábra bemutatja, hogyan haladnak keresztül a csomagok a rendszeren a csomagszűréskor.

A 7.3. táblázat az öt kapcsolódási pontot mutatja, és ismerteti, hogy a csomagfolyam mely pontjain határozhatjuk meg a feldolgozást.

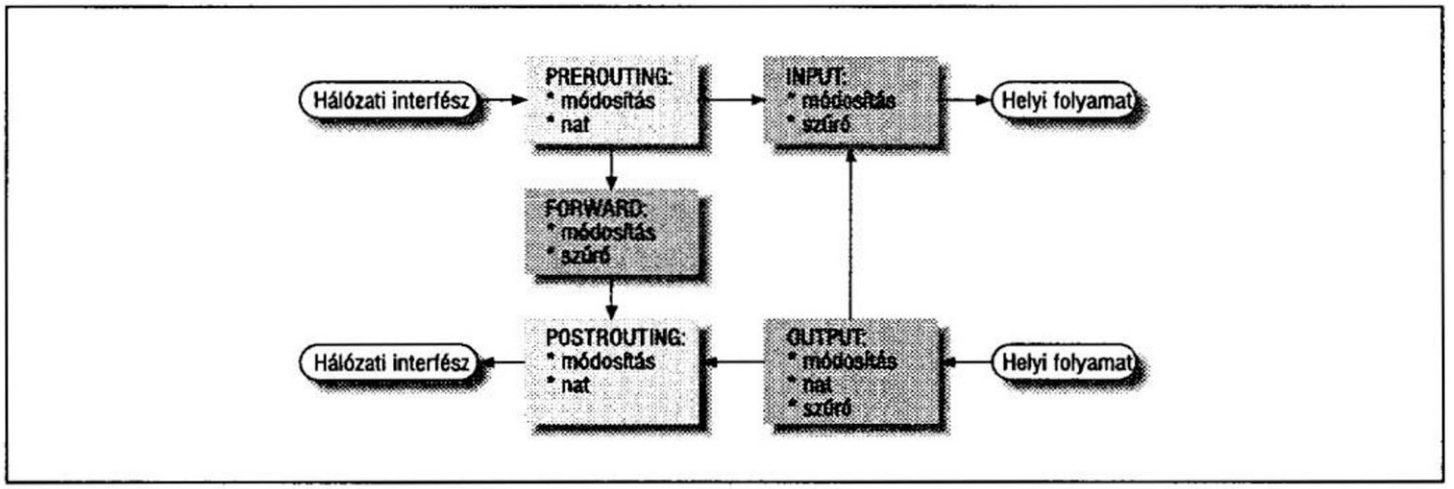


7.2. ábra. A hálózati csomagfolyam kapcsolódási pontjai



7.3. ábra. A hálózati csomagfolyam és a kapcsolódási pontok a NAT esetén

Egy kis csemege az érdeklődők számára: a kapcsolódási pontokat a rendszermag fejlécállománya definiálja *(/usr/include/linux/netfilter_ipv4.h)*, például NF_IP_FORWARD, NF_IP_LOCAL_{IN,OUT} és NF_IP_{PRE,POST}_ROUTING néven.



7.4. ábra. A hálózati csomagfolyam és a kapcsolódási pontok szűrés esetén

7.3. táblázat. A kapcsolódási pontok

Pont	*Lehetővé teszi a csomagok feldolgozását...*
FORWARD	...amikor azok áthaladnak az átjáró számítógépen, beérkezve az egyik interfészen és azonnal kilépve a másikon;
INPUT	...mielőtt egy helyi folyamathoz kerülnek;
OUTPUT	...miután egy helyi folyamat létrehozta őket;
POSTROUTING	...mielőtt kilépnek egy hálózati interfészen;
PREROUTING	...miután megérkeztek egy hálózati interfészről (az interfész „lehallgató" [promiscuous] üzemmódjában a nem nekünk szánt csomagok eldobása, illetve az ellenőrző összeg érvényességének vizsgálata után).

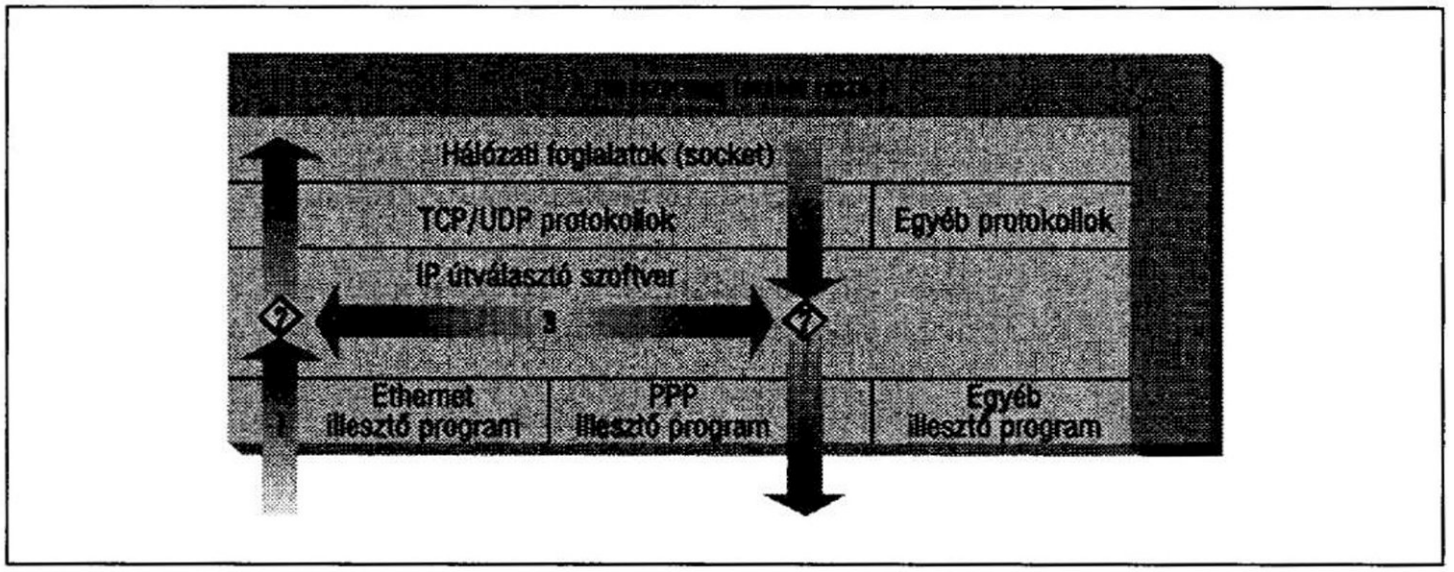
A megfelelő lánc kiválasztása attól függ, hogy a csomag életciklusának mely részén szeretnénk a szabályainkat érvényre juttatni. Ha például a kimenő csomagokat szeretnénk szűrni, általában az OUTPUT láncban tesszük ezt, mivel a POSTROUTING lánc nem kapcsolódik a `filter` táblázathoz.

Három lehetőség a szűrésre

Nézzük meg, hogy egy Unix gazdagép, vagy bármilyen, IP útválasztásra képes gazdagép hogyan dolgozza fel az IP csomagokat. A 7.5. ábrán bemutatott alapvető lépések a következők:

1. Fogadjuk az IP csomagot.
   A bejövő IP csomagot megvizsgáljuk és eldöntjük, hogy a mi gazdagépünk valamely folyamatának szánták-e.
2. Ha a csomag a gazdagép részére érkezett, helyben dolgozzuk fel.
3. Ha nem a mi gazdagépünkre szánták (és az IP továbbítás be van kapcsolva), megpróbáljuk megkeresni a megfelelő útvonalat az útvonalválasztó táblázatban, majd továbbítjuk a megfelelő interfészre, illetve visszautasítjuk, ha nem találunk hozzá útvonalat.
4. A helyi folyamatoktól érkező csomagokat az útválasztó szoftverhez küldjük, és az továbbítja a megfelelő interfészre.
   A kimenő IP csomagoknál megvizsgáljuk, hogy létezik-e hozzájuk érvényes útvonal. Ha nem létezik, eldobjuk (teljesen figyelmen kívül hagyjuk) vagy visszautasítjuk (miután egy ICMP üzenetben jelezzük, hogy nem találtunk megfelelő útvonalat a rendeltetési helyhez, figyelmen kívül hagyjuk).
5. Elküldjük a csomagot.

Ábránkon az 1 › 3 › 5 folyam gazdagépünket jelzi, amint az adatok útvonalát kijelöli két gép: Ethernet hálózatunk egy gazdagépe és a PPP kapcsolatunkon keresztül elérhető egyik gazdagép között. Az 1 › 2 és 4 › 5 folyamok a helyi gazdagépen futó egyik hálózati program be- és kimenő adatfolyamát szemlélteti. A 4 › 3 ›2 folyam egy visszacsatolási kapcsolat adatfolyamát jelzi. Természetesen az adatok a hálózati eszközökön befelé és kifelé is áramlanak. A kérdőjelek azokat a helyeket mutatják, ahol az IP réteg útválasztási döntést hoz.



7.5. ábra. Az IP csomagok feldolgozásának lépései

A Linux rendszermag IP tűzfala képes a folyamat különböző pontjain szűrést alkalmazni, vagyis szűrhetjük a gazdagépre érkező csomagokat, a gazdagéppel továbbított csomagokat és az átvitelre készen álló csomagokat.

Elsőre mindez talán szükségtelenül bonyolultnak látszik, azonban olyannyira rugalmas, hogy rendkívül kifinomult és hatékony konfigurációk kialakítását teszi lehetővé.

A táblázatok

Az *iptables* három beépített táblázatot (table) tartalmaz, ezek a `filter`, a `mangle` és a `nat`. Mindegyikhez tartoznak előre beállított láncok egy vagy több kapcsolódási ponthoz. A beépített táblázatokat a 7.4. táblázatban láthatjuk, míg a kapcsolódási pontokat a 7.2. ábrán foglalja össze.

7.4. táblázat. Beépített táblázatok

Táblázat	Leírás
`Filter`	A számítógépre beengedhető, azon átengedhető, illetve arról kiengedhető forgalomhoz állítja be a szabálygyűjteményt. Alapértelmezés szerint az *iptables* ennek a táblának a láncain hajtja végre a műveleteket, ha nem adunk meg kimondottan másik táblázatot. Beépített láncai a `FORWARD`, az `INPUT` és az `OUTPUT`.
`Mangle`	Különleges csomagmanipulálásra használjuk, például az IP opciók eltávolítására (például az `IPV4OPTSSTRIP` célkiterjesztés esetén). Beépített láncai a `FORWARD`, az `INPUT`, az `OUTPUT`, a `POSTROUTING` és a `PREROUTING`.
`Nat`	A kapcsolatkövetéssel együtt használjuk a `NAT` kapcsolatainak átirányítására, jellemzően a forrás vagy a rendeltetési cím alapján. Beépített láncai az `OUTPUT`, a `POSTROUTING` és a `PREROUTING`.

Az *iptables* felel azért, hogy a hálózati csomagok a forrás és a rendeltetési hely alapján a táblázatok megfelelő láncain haladjanak keresztül, a 7.2. ábrán ismertetett sorrendben.

Az alapértelmezett táblázat a `filter` táblázat. Ha nem adunk meg kifejezetten másik táblázatot az *iptables* parancsban, a rendszer a `filter` táblázaton hajtja végre azt.

A láncok

A csomagok keresztülhaladnak a láncokon (chain), és ezeknek a szabályait egyesével, sorrendben alkalmazzuk rájuk. Ha a csomag nem felel meg a szabályi feltételnek, továbblép a lánc következő szabályához. Amikor egy csomag eléri a lánc utolsó szabályát és még mindig nem illeszkedik, a lánc saját eljárását alkalmazzuk rá.

Alapértelmezés szerint minden táblázat tartalmaz láncokat néhány vagy az összes kapcsolódási ponthoz (a láncok eredetileg üresek). A kapcsolódási pontokat a 7.3. táblázat, az egyes táblázatok beépített láncait a 7.4. táblázat foglalja össze.

Emellett saját egyedi láncokat is létrehozhatunk szabályaink rendszerbe szervezésére.

A lánc *saját eljárása* (policy) határozza meg, hogy mi történik azokkal a csomagokkal, amelyek elérik a lánc végét anélkül, hogy valamely célhoz kerülnének. A beépített láncok saját eljárása csak az `ACCEPT` és `DROP` beépített célok egyike lehet (lásd a fejezet *A célok* című részét), ez pedig alapértelmezésben az `ACCEPT`. A felhasználói láncok implicit saját eljárása a `RETURN`, amin nem lehet változtatni.

Ha egy beépített lánchoz saját összetettebb eljárást szeretnénk használni, illetve a felhasználói lánchoz valami mást a `RETURN` helyett, fűzzünk olyan szabályt a lánc végéhez, amely minden csomaggal egyezik, és olyan célt, amely megfelel elvárásainknak. A lánc saját eljárását állítsuk a `DROP` célra arra az esetre, ha a „mindenevő" szabályunkba hiba csúszna; de akkor is hasznát vehetjük, ha mindenevő szabályunkat módosítjuk (kitöröljük, majd a módosított változatot ismét elhelyezzük), mert így ez alatt az idő alatt sem érkezhet gépünkre nem kívánt csomag.

A szabályok

Egy *iptables* szabály (rule) egy vagy több *illesztési* feltételt és egy *célt* tartalmaz. Az illesztési feltétel határozza meg, hogy a szabály mely hálózati csomagokat érinti. A cél azt határozza meg, hogy a szabály hogyan befolyásolja a hálózati csomagokat. Ahhoz, hogy egy szabály illeszkedjen egy csomagra, minden illesztési feltételnek teljesülnie kell.

A rendszer minden szabályhoz csomag- és bájtszámlálókat tart fenn. Amikor egy csomag egy szabályhoz ér és megfelel a szabály feltételeinek, a csomagszámláló értéke eggyel, a bájtszámláló értéke pedig az illeszkedő csomag méretével nő.

A szabályok illesztési és cél része egyaránt opcionális. Ha nincs illesztési feltétel, minden csomag illeszkedik. Ha nincs cél, semmi nem történik a csomagokkal (a feldolgozás úgy folytatódik, mintha a szabály nem is létezne, kivéve, hogy a rendszer frissíti a csomag- és a bájtszámláló értékét). A `filter` táblázat `FORWARD` láncához ilyen „üres" szabályt az `iptables -t filter -A FORWARD` paranccsal adhatunk.

Az illesztések

Az *iptables* programban számos különféle illesztési feltételt (match) megadhatunk, noha egyesek csupán az olyan rendszermagok esetén működnek, amelyekben bizonyos tulajdonságokat bekapcsoltunk. A Generic Internet Protocol (IP) illesztések (például a protokoll, a forrás vagy a rendeltetési cím) minden IP csomagra érvényesek.

Az általános illesztések mellett az *iptables* sok különleges célú illesztést is elérhetővé tesz a dinamikusan betölthető illesztési bővítményeken keresztül (a bővítmények használatára a `-m` vagy `--match` opcióval utasíthatjuk az *iptables* programot).

Létezik egy illesztési bővítmény az IP réteg alatti hálózati réteg kezelésére is. A `mac` illesztési bővítmény az illesztést az Ethernet Media Access Controller (MAC) címek alapján végzi.

A célok

A célok (target) egyrészt azokat a műveleteket írják le, amelyeket egy csomag illeszkedésekor hajtunk végre, másrészt a láncok saját eljárásait határozzák meg. Az *iptables* négy beépített célt tartalmaz, de vannak olyan bővítő modulok is, amelyek további célokat biztosítanak. A 7.5. táblázat a beépített célokat foglalja össze.

7.5. táblázat. A beépített célok

Cél	*Leírás*
ACCEPT	Továbbengedi a csomagot a feldolgozás következő állomására. Leállítja az aktuális lánc feldolgozását és a 7.2. ábra szerinti következő állomásra lép.
DROP	Véglegesen abbahagyja a csomag feldolgozását. Nem veti össze semmilyen szabállyal, lánccal vagy táblázattal. Ha szeretnénk a küldőnek visszajelzést küldeni, használhatjuk a `REJECT` cél bővítményt.
QUEUE	A csomagot a felhasználói térbe küldi (vagyis olyan kódhoz, amely nem része a rendszermagnak). További információt a `libipq` súgóoldalon találunk.
RETURN	Egy felhasználói lánc szabályában abbahagyja a lánc feldolgozását és folytatja a hívó lánc feldolgozását attól a szabálytól, amelynek ez a lánc volt a célja. Egy beépített lánc szabályában abbahagyja a lánc feldolgozását és a lánc saját eljárását alkalmazza rá. A láncok saját eljárásáról lásd a fejezet A láncok című részét.

A Linux beállítása a tűzfal kezelésére

A Linux rendszermagot be kell állítanunk az IP tűzfal kezelésére. Ezzel kapcsolatban nincs sok dolgunk, csupán a megfelelő opciókat kell kiválasztanunk, amikor a rendszermag beállításához a következő parancsot futtatjuk:*

```
# make menuconfig
```

A 2.4 rendszermagban a következő opciókat szükséges beállítanunk:

```
Networking options  -->
     [*] Network packet filtering (replaces ipchains)
         IP: Netfilter Configuration  -->
              .
             <M> Userspace queueing via NETLINK (EXPERIMENTAL)
             <M> IP tables support (required for filtering/masq/NAT)
             <M>   limit match support
             <M>   MAC address match support
             <M>   netfilter MARK match support
             <M>   Multiple port match support

             <M>   TOS match support
             <M>   Connection state match support
             <M>   Unclean match support (EXPERIMENTAL)
             <M>   Owner match support (EXPERIMENTAL)
             <M>   Packet filtering
             <M>     REJECT target support
             <M>     MIRROR target support (EXPERIMENTAL)
              .
             <M>   Packet mangling
             <M>     TOS target support
             <M>     MARK target support
             <M>   LOG target support
             <M> ipchains (2.2-style) support
             <M> ipfwadm (2.0-style) support
Loading the Kernel Module
```

A rendszermag modul betöltése

Mielőtt az *iptables* parancsot kiadhatnánk, be kell töltenünk a *netfilter* rendszermag modult, hogy támogatást biztosítsunk hozzá. Ezt a legegyszerűbben a *modprobe* utasítással tehetjük meg:

```
# modprobe ip_tables
```

* A tűzfal csomagnaplózása különleges szolgáltatás, amely az adott tűzfalszabálynak megfelelő valamennyi csomagról egysoros információt ír egy speciális eszközre, hogy megjelenítse ezeket a számunkra.

Kompatibilitás visszafelé: az ipfwadm és az ipchains

A Linux *netfilter* kivételes rugalmasságát igazolja, hogy képes az *ipfwadm* és az *ipchains* interfészek emulálására. Ez a képessége alaposan megkönnyíti az átállást a tűzfalszoftverek új generációjára (noha végül úgyis átírjuk szabályainkat az *iptables* számára).

A kompatibilitást visszafelé az `ipfwadm` és az `ipchains` programokhoz a *netfilter* két rendszermag modulja, az *ipfwadm.o* és az *ipchains.o* biztosítja. Egyidejűleg csak az egyiket tölthetjük be, és csak az egyiket használhatjuk, ha az *ip_tables.o* modul nincs betöltve. Amikor a megfelelő modul betöltődik, a *netfilter* pontosan úgy működik, mint a korábbi tűzfal-megvalósítás.

A *netfilter* az *ipchains* interfészt a következő utasításokkal emulálja:

```
# rmmod ip_tables
# modprobe ipchains
# ipchains opciók
```

Az iptables használata

Az *iptables* parancs dinamikusan betöltött könyvtárakkal bővíthető. A könyvtárakat a *netfilter* forráscsomag tartalmazza, melyet a *http://www.netfilter.org/* címről tölthetünk le. A 2.4 sorozatszámú rendszermagokra épülő Linux terjesztések szintén tartalmazni fogják.

Az *iptables* paranccsal az IP szűrést és a NAT-ot állíthatjuk be (illetve más csomagfeldolgozó alkalmazásokat, beleértve a nyomkövetést, a naplózást és a manipulációt). Ezt két szabálytáblázat, a *filter* és a *nat* teszi lehetővé. A *filter* táblázat az alapértelmezett, melyet a `-t` opcióval írhatunk felül. Létezik öt beépített lánc is. Az `INPUT` és `FORWARD` láncok a *filter* táblázathoz, a `PREROUTING` és a `POSTROUTING` láncok a *nat* táblázathoz érhetők el, míg az `OUTPUT` lánc mindkét táblához használható. A fejezetben csak a *filter* táblázattal foglalkozunk. A *nat* táblázat a 9. fejezet témája.

A legtöbb *iptables* parancs általános szintaxisa:

```
# iptables parancs szabály-meghatározás bővítmény
```

Lássunk most néhány opciót részletesen is; később példákat is hozunk.
Az *iptables* parancs legtöbb opciója alparancsokból és szabályillesztési feltételekből áll; a 7.6. táblázat az egyéb opciókat ismerteti.

7.6. táblázat. Az iptables „vegyes" opciói

Opció	*Leírás*
`-c csomagszám bájtszám`	A `-A`, `-I` vagy `-R` alparancsokkal kombinálva beállítja a csomagszámlálót a csomagszám, a bájtszámlálót pedig a bájtszám értékre (az új vagy módosított szabályhoz).
`--exact`	A `-x` szinonimája.

7.6. táblázat. Az iptables „vegyes" opciói (folytatás)

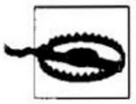
Opció	*Leírás*
`-h`	Az *iptables* felhasználási információját jeleníti meg. Ha a `-m` illesztés vagy a `-j` cél után adjuk meg, akkor további segítséget kaphatunk az illesztés- vagy célbővítményekről.
`--help`	A `-h` szinonimája.
`-j cél [opciók]`	Meghatározza, hogy mi történjék a szabálynak megfelelő csomagokkal. A cél lehet egy felhasználói lánc, valamelyik beépített cél vagy egy *iptables* bővítmény (az utóbbi esetben további opciók lehetségesek).
`--jump`	A `-j` szinonimája.
`—line-numbers`	A `-L` alparanccsal használva az egyes láncokban található szabályok számát adja meg, így amikor egy lánchoz új szabályt fűzünk (`-I`) vagy abból egy szabályt törlünk (`-D`), a szabályra a számuk alapján hivatkozhatunk. Ne feledjük, hogy a számozás a szabályok hozzáfűzésével, illetve törlésével megváltozik.
`-m illesztés [opciók]`	Bővített illesztés meghívása, esetleg további opciókkal.
`--match`	A `-m` szinonimája.
`-M parancs`	Egy *iptables* modul betöltését szolgálja (új cél vagy illesztési bővítménnyel), amikor szabályokat fűzünk a lánc végéhez, szúrunk be vagy törlünk ki.
`--modprobe=parancs`	A `-M` szinonimája.
`-n`	A címeket és portokat számmal jeleníti meg, ahelyett, hogy kikeresné az IP címekhez tartozó tartományneveket, illetve a portszámokhoz tartozó szolgáltatásneveket.
`--numeric`	A `-n` szinonimája.
`--set-counters`	A `-c` szinonimája.
`-t table`	Végrehajtja a táblázaton a megadott alparancsot. Ha a táblázatot nem adjuk meg, az alparancs a `filter` táblázatra vonatkozik alapértelmezés szerint.
`--table`	A `-t` szinonimája.
`-v`	Részletes kimenetet eredményez.
`--verbose`	A `-v` szinonimája.
`-x`	A csomag- és bájtszámlálók értékét pontosan megadja, és nem az alapértelmezett rövidített formátumot használja a mértékegységek jeleivel (K, M vagy G).

Ha segítségre van szükségünk...

Az *iptables* némi on-line segítséget is biztosít. Az alapvető információkhoz a következő parancsokkal juthatunk:

```
iptables -h | —help
iptables -m match -h
```

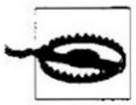
```
iptables -j TARGET -h
man iptables
```



Esetenként az információforrások ellentmondanak egymásnak.

Az iptables alparancsok

Minden *iptables* parancs tartalmazhat egy alparancsot, amely adott táblázaton (egyes esetekben láncon) végez műveletet. A 7.7. táblázat az alparancsok megadására szolgáló opciókat gyűjti egybe.



Az 1.2.7a kiadásban az *iptables* parancs súgóoldala az összegzésben egy -C opciót is említ, de az *iptables* parancsnak nincs -C opciója.

7.7. táblázat. Az iptables alparancsai

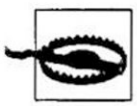
Opció	*Leírás*
`-A` *`lánc szabály`*	A `szabályt` a `lánchoz` fűzi.
`--append`	A `-A` szinonimája.
`-D` *`lánc`* `[index |` *`szabály`*`]`	Törli az `index` pozíciójában lévő szabályt, illetve a `szabályt` a `láncból`.
`--delete`	A `-D` szinonimája.
`--delete-chain`	A `-X` szinonimája.
`-E` *`lánc újlánc`*	A `lánc` nevét az `újlánc` névre változtatja.
`-F` *`[lánc]`*	Töröl minden szabályt a `láncból` (vagy ha a `láncot` nem adjuk meg, akkor minden láncból).
`--flush`	A `-F` szinonimája.
`-I` *`lánc`* `[index]` *`szabály`*	Beszúrja a `szabályt` a `lánc` elejére, vagy az `index` pozíciójában lévő szabály elé.
`--insert`	A `-I` szinonimája.
`-L` *`[lánc]`*	A `lánc` szabályait listázza ki (vagy az összes láncét, ha nem adjuk meg a `láncot`).
`--list`	A `-L` szinonimája.
`-N` *`lánc`*	Új felhasználói `láncot` hoz létre.
`--new-chain`	A `-N` szinonimája. Általában rövidítjük: `--new`.
`-P` *`lánc cél`*	A beépített `lánc` alapértelmezett saját szabályát a `célra` állítja. (csak a beépített láncokra és célokra vonatkozik.)
`--policy`	A `-P` szinonimája.
`-R` *`lánc index szabály`*	Az `index` pozíciójában álló szabályt az új `szabályra` cseréli.
`--rename-chain`	A `-E` szinonimája.

7.7. táblázat. Az iptables alparancsai (folytatás)

Opció	*Leírás*
`--replace`	A `-R` szinonimája.
`-V`	Megjeleníti az *iptables* verziószámát.
`--version`	A `-V` szinonimája.
`-X [lánc]`	Töröl egy felhasználói `lánc`ot, illetve az összes felhasználói láncot, ha nem adjuk meg a `lánc` nevét.
`-Z lánc`	Lenullázza a `lánc` csomag- és bájtszámlálóit (vagy az összes láncét, ha nem adjuk meg a `lánc` nevét).
`--zero`	A `-Z` szinonimája.

Az iptables alapvető illesztései

Az *iptables* rendelkezik néhány beépített illesztéssel és céllal, valamint egy bővítménykészlettel, amely automatikusan betöltődik, ha hivatkozunk rájuk. Az IP esetén az illesztéseket beépítettnek, míg más esetekben bővítményeknek tekintjük (még akkor is, ha az `icmp`, `tcp` és `udp` illesztési bővítmények automatikusan betöltődnek, amikor a protokollokra a `-p` beépített IP illesztő opcióban hivatkozunk).



Egyes opciók jelentését ellenkezőre fordíthatjuk, ha közvetlenül az opció előtt, szóközök között egy felkiáltójelet adunk meg. Az ilyen opciókat [!] jelöli. A következő részben csak a rendes jelentésüket ismertetjük, mivel ebből már következtethetünk megfordított értelmükre.

Az Internet Protokoll (IPv4) illesztések

Ezeket a beépített illesztéseket a megelőző `-m` argumentum nélkül érhetjük el az *iptables* parancsban. A 7.8. táblázat egy Internet Protokoll (IPv4) csomag mezőinek elrendezését mutatja. A mezőket különféle illesztési és célbővítmények kezelik (beleértve az itt leírt beépített illesztéseket). A 7.8. táblázat az illesztések opcióit foglalja össze.

7.8. táblázat. Az Internet Protokoll illesztési opciói

Opció	*Leírás*
`-d [ ! ] cím[/maszk]`	A `cím` egy rendeltetési cím (vagy tartomány, ha a `maszk` szerepel).
`--destination`	A `-d` szinonimája.
`--dest`	A `-d` szinonimája.
`[ ! ] -f`	A fragmentáláson átesett csomag második vagy további szegmense. A kapcsolatkövetés automatikus defragmentálást végez, ezért az opcióra csak ritkán van szükség. Ha nem alkalmazunk kapcsolatkövetést, akkor használhatjuk.
`--fragments`	A `-f` szinonimája. Általános rövidítése (az *iptables* súgóoldalon is) a `--fragment`.

7.8. táblázat. Az Internet Protokoll illesztési opciói (folytatás)

Opció	*Leírás*
`-i [ ! ] in`	Az `in` bemeneti interfész (ha az `in` + jelre végződik, minden `in` kezdetű névvel rendelkező interfész).
`--in-interface`	A `-i` szinonimája.
`-o [ ! ] out`	Az `out` kimeneti interfész (ha az `out` + jelre végződik, minden `out` kezdetű névvel rendelkező interfész).
`--out-interface`	A `-o` szinonimája.
`-p [ ! ] proto`	A `proto` egy protokoll név vagy szám. A 7.9. táblázat a gyakori protokollok nevét és számát mutatja. A hivatalos nevek leképzése számokra a rendszer */etc/protocols* állománya alapján történik (és érzékeny a kis- és nagybetűk különbségére). Az állományban a hivatkozási nevek (alias) nem használhatók. A hivatalos protokoll-lista a *http://www.iana.org/assigments/protocol-numbers* címen tekinthető meg. A `-p proto` egy implicit `-m` kapcsolót is tartalmaz, ha a protokoll az `icmp`, a `tcp` vagy az `udp`.
`--protocol`	A `-p` szinonimája. Gyakran így rövidítik: `--proto`.
`-s [ ! ] cím[/maszk]`	A `cím` egy forráscím (vagy tartomány, ha a `maszk` is szerepel).
`--source`	A `-s` szinonimája.
`--src`	A `-s` szinonimája.

A `-s` és `-d` címeiben a maszkokat megadhatjuk a régi, pontozott négyes jelöléssel is, például 192.168.1.0/255.255.255.0, vagy az újabb Common Inter-Domain Routing (CIDR) jelöléssel, például 192.168.1.0/24 (lásd az RFC 1591 specifikációt, amely elérhető a *http://www.rfc-editor.org/rfc/rfc1591.txt* címen).

7.9. táblázat. Gyakori IP protokollok

Név	*Szám(ok)*	*Leírás*
ALL	1, 6, 17	Megfelel annak, ha nem adunk meg protokollt.
icmp	1	Internet Control Message Protocol.
tcp	6	Transmission Control Protocol.
udp	17	User Datagram Protocol.

Az Ethernet Media Access Controller (MAC) illesztés

Az illesztés alapja a forrás Ethernet interfész Media Access Controller (MAC) címe. A 7.10. táblázat az illesztés egyetlen opcióját ismerteti.

Ez valójában nem IP illesztés. Az Ethernet a hálózati architektúrában alacsonyabb szinten áll, de mivel sok IP hálózat Etherneten fut és a MAC cím könnyen elérhető, az illesztési bővítmény a rendelkezésünkre áll.

Az illesztést csak akkor használhatjuk, ha a rendszermagban bekapcsoltuk a CONFIG_IP_NF_MATCH_MAC opciót.

7.10. táblázat. A MAC illesztési opciók

Opció	*Leírás*
`–mac-source [ ! ] mac`	Akkor illeszkedik, ha az Ethernet keret forrás MAC mezője illeszkedik a *mac*re. A formátum az *XX:XX:XX:XX:XX:XX*, ahol minden *XX* helyére két hexadecimális számjegy kerül.

Csak a `PREROUTING`, a `FORWARD` vagy az `INPUT` láncok szabályaival használjuk, és csak Ethernet eszközökről érkező csomagokhoz.

Ahhoz például, hogy egyetlen Ethernet eszköz kommunikálhasson az interfészen (például egy vezeték nélküli eszközhöz tartozó interfészen) keresztül, adjuk meg a következőt:

```
iptables -A PREROUTING -i eth1 -m mac –mac-source ! 0d:bc:97:02:18:21 -j DROP
```

Az Internet Control Message Protocol illesztés

Az Internet Control Message Protocol (ICMP) illesztési bővítmény automatikusan betöltődik, ha megadjuk a `-p icmp` kifejezést. A 7.11. táblázat az illesztés opcióit tartalmazza.

7.11. táblázat. ICMP illesztési opciók

Opció	*Leírás*
`--icmp-type [ ! ] típusnév`	A `típusnév`nek megfelelő ICMP típusra illeszkedik.
`--icmp-type [ ! ] típus[/kód]`	A megadott ICMP `típus`ra és `kód`ra illeszkedik.

A hivatalos ICMP típusokat és kódokat a hivatalos adatbázisban, a *http://www.iana.org/assignments/icmp-parameters* címen találhatjuk (lásd RFC 3232, „Assigned Numbers: RFC 1700 is Replaced by an On-line Database"; elérhető a *http://www.rfc-editor.org/rfc/rfc3232.txt).*

A User Datagram Protocol illesztés

A User Datagram Protocol (UDP) illesztési bővítmény automatikusan betöltődik, ha a `-p udp` kifejezést megadjuk. A 7.12. táblázat az illesztés opcióit tartalmazza.

7.12. táblázat. Az UDP illesztési opciói

Opció	*Leírás*
`--destination-port [ ! ] port[:port]`	Illeszkedik, ha az UDP rendeltetési port a `port` (ha csak egy portot adunk meg), illetve beleesik a tartományba (ha mindkét `port`ot megadjuk). A portokat megadhatjuk névvel (a rendszerünk */etc/services* állományából) vagy számmal.
`--dport`	A `--destination-port` szinonimája.
`--source-port [ ! ] port[:port]`	Illeszkedik, ha az UDP forrásport a `port` (ha csak egy portot adunk meg), illetve beleesik a tartományba (ha mindkét `port`ot megadjuk). A portokat megadhatjuk névvel (a rendszerünk */etc/services* állományából) vagy számmal.
`--sport`	A `--source-port` szinonimája.

A Transmission Control Protocol illesztés

A Transmission Control Protocol (TCP) illesztési bővítmény automatikusan betöltődik, ha megadjuk a `-p tcp` kifejezést. A 7.13. táblázat az illesztés opcióit tartalmazza.

7.13. táblázat. Az illesztés opciói

Opció	*Leírás*
`--destination-port`	A `--dport` szinonimája.
`--dport [ ! ] port[:port]`	Illeszkedik, ha a TCP rendeltetési port a `port` (ha csak egy portot adunk meg), illetve beleesik a tartományba (ha mindkét `port`ot megadjuk). A portokat megadhatjuk névvel (a rendszerünk */etc/services* állományából) vagy számmal.
`--mss` *érték*`[:`*érték*`]`	Illeszkedik a SYN és ACK csomagokra, ha a TCP protokoll Maximum Segment Size (MSS) mezője egyenlő az *érték*kel (ha csak egy *érték*et adunk meg), illetve beleesik a tartományba (ha mindkét *érték*et megadjuk). Lásd még a `tcpmss` illesztő bővítményt.
`--source-port`	A `--sport` szinonimája.
`--sport [ ! ] port[:port]`	Illeszkedik, ha az TCP forrásport a *port* (ha csak egy *port*ot adunk meg), illetve beleesik a tartományba (ha mindkét *port*ot megadjuk). A portokat megadhatjuk névvel (a rendszerünk */etc/services* állományából) vagy számmal.
`[ ! ] --syn`	A `--tcp-flags SYN,RST,ACK SYN` szinonimája. Az illeszkedő csomagok a „SYN" csomagok.
`--tcp-flags [ ! ]` *maszk comp*	Ellenőrzi a *maszk* kapcsolót, és akkor illeszkedik, ha csak a *comp* kapcsolók vannak beállítva.
`--tcp-option[ ! ]` *szám*	Illeszkedik, ha a *szám* TCP opció be van állítva.

Egy naiv példa

Tegyük fel, hogy vállalatunknak van egy hálózata, amelyen Linux alapú tűzfal gazdagépet futtatunk, hogy a felhasználók elérhessék a világháló kiszolgálóit (kizárólag a 80-as HTTP porton keresztül, a 443-as HTTPS porton nem); minden egyéb forgalmat letiltunk. A következő utasításokkal ehhez a feladathoz készítünk egy egyszerű továbbító szabályrendszert. Noha a megoldásunk valóban egyszerű, a 9. fejezetben bemutatott NAT és álcázó megoldásokat gyakrabban használják az ilyen típusú alkalmazásokhoz.

Ha hálózatunk 24 bites hálózati maszkot használ (C osztályú) és a címe 172.16.1.0, a következő iptables szabályokat definiáljuk:

```
1    # modprobe ip_tables
2    # iptables -F FORWARD
3    # iptables -P FORWARD DROP
4    # iptables -A FORWARD -p tcp -s 0/0 —sport 80 \
          -d 172.16.1.0/24 --syn -j DROP
5    # iptables -A FORWARD -p tcp -s 172.16.1.0/24 \
          --dport 80 -d 0/0 -j ACCEPT
6    # iptables -A FORWARD -p tcp -d 172.16.1.0/24 \
          --sport 80 -s 0/0 -j ACCEPT
```

Az 1–3. sorok telepítik az *iptables* programot a futó rendszermagba, törlik a `filter` táblázat `FORWARD` láncát (amely az alapértelmezett táblázat, ha nem adunk meg mást az *iptables* parancs argumentumában), és a `filter` táblázat `FORWARD` láncának saját eljárását a `DROP` műveletre állítják.

A 4. sor megakadályozza, hogy az internet gazdagépek a belső hálózattal kapcsolatot létesítsenek. Ehhez eldobja a `SYN` csomagokat (de csak akkor, ha a forrásport a 80-as, hiszen a későbbi szabályok csak ezeket engedik át).

Az 5. sor a belső hálózatról bármely gazdagép 80-as portjára tartó csomagokat kiengedi.

A 6. sor bármely gazdagép 80-as portjáról a belső hálózat bármely gazdagépére tartó csomagokat átengedi.

Minta tűzfal-konfiguráció

Megismertük a tűzfalak beállításának alapvető elemeit. Lássunk most egy könnyen testre szabható tűzfal-konfigurációt. A példában a 172.16.1.0/24 tartományt úgy kezeljük, mintha az nyilvános útválasztással elérhető volna, holott valójában magánhálózat, amely útválasztásra nem vehető igénybe. A példában azért használunk útválasztásra alkalmatlan hálózatot, mert *valamilyen* hálózatot használnunk kell, és nem akarunk egy valódi, útválasztásban részt vevő hálózati számot itt megadni. A bemutatott parancsok egy valódi C osztályú, nyilvános útválasztásban részt vevő hálózattal is működnének.

```
#!/bin/bash
#############################################################################
# Ez a minta konfiguráció egyetlen gazdagép tűzfal beállítását szemlélteti
# A tűzfalgép nem támogat semmilyen szolgáltatást
#############################################################################
#
# FELHASZNÁLÓI BEÁLLÍTÁSOK (a listákat vessző választja el)
#
#   OURNET    A belső hálózat címtere
#   OURBCAST  A belső hálózat üzenetszóró címe
#   OURDEV    A belső hálózat interfészének neve
#
#   ANYADDR   A külső hálózat címtere
#   EXTDEV    A külső hálózat interfészének neve
#
#   TCPIN     A beengedhető TCP portok listája (üres = mindegyik)
#   TCPOUT    A kiengedhető TCP portok listája (üres = mindegyik)
#
#   UDPIN     A beengedhető UDP portok listája (üres = mindegyik)
#   UDPOUT    A kiengedhető UDP portok listája (üres = mindegyik)
#
#   LOGGING   A naplózás bekapcsolásához állítsuk 1-re,
#             egyébként hagyjuk üresen
#
#############################################################################

OURNET="172.29.16.0/24"
OURBCAST="172.29.16.255"
OURDEV="eth0"

ANYADDR="0/0"
EXTDEV="eth1"

TCPIN="smtp,www"
TCPOUT="smtp,www,ftp,ftp-data,irc"

UDPIN="domain"
UDPOUT="domain"

LOGGING=

#############################################################################
#
# IMPLEMENTÁCIÓ
#
#############################################################################

#
# A modulok telepítése
#

modprobe ip_tables
modprobe ip_conntrack # Nem kell a töredékekkel foglalkoznunk
```

```
#
# A gazdagépre szánt, kívülről érkező összes csomag eldobása.
#

iptables -A INPUT -i $EXTDEV -j DROP

#
# A filter táblázat FORWARD láncának összes szabályát eltávolítjuk,
# és a lánc saját eljárását a DROP célra állítjuk.
#

iptables -F FORWARD                                          # Törlés
iptables -P FORWARD DROP                                     # Eljárás = DROP
iptables -A FORWARD -s $OURNET -i $EXTDEV -j DROP            # Hamisítás ellen
iptables -A FORWARD -p icmp -i $EXTDEV -d $OURBCAST -j DROP # Smurf támadás ellen

#
# TCP - LÉTEZŐ KAPCSOLATOK
#
# A létező kapcsolatok (ahol az ACK bit be van állítva) összes TCP
# csomagját fogadjuk azokon a TCP portokon, melyeken az áthaladást
# engedélyezzük. Ez az érvényes TCP csomagok több mint 95%-át érinti.
#

iptables -A FORWARD -d $OURNET -p tcp --tcp-flags SYN,ACK ACK \
  -m multiport --dports $TCPIN -j ACCEPT

iptables -A FORWARD -s $OURNET -p tcp --tcp-flags SYN,ACK ACK \
  -m multiport --sports $TCPIN -j ACCEPT

#
# TCP - ÚJ BEJÖVŐ KAPCSOLATOK
#
# A kívülről érkező kapcsolati kéréseket csak az engedélyezett
# TCP portokon fogadjuk.
#

iptables -A FORWARD -i $EXTDEV -d $OURNET -p tcp --syn \
  -m multiport --sports $TCPIN -j ACCEPT

#
# TCP - ÚJ KIMENŐ KAPCSOLATOK
#
# A kifelé tartó összes tcp kapcsolati kérést fogadjuk az engedélyezett
# TCP portokon.
#

iptables -A FORWARD -i $OURDEV -d $ANYADDR -p tcp --syn \
  -m multiport --dports $TCPOUT -j ACCEPT

#
# UDP - BEJÖVŐ
#
# Az UDP csomagokat az engedélyezett portokon engedjük be és vissza.
#
```

```
iptables -A FORWARD -i $EXTDEV -d $OURNET -p udp \
  -m multiport –dports $UDPIN -j ACCEPT

iptables -A FORWARD  -i $EXTDEV -s $OURNET -p udp \
  -m multiport –sports $UDPIN -j ACCEPT

#
# UDP - KIMENŐ
#
# Az UDP csomagokat az engedélyezett portokra engedjük ki és vissza.
#

iptables -A FORWARD -i $OURDEV -d $ANYADDR -p udp \
  -m multiport –dports $UDPOUT -j ACCEPT

iptables -A FORWARD -i $OURDEV -s $ANYADDR -p udp \
  -m multiport –sports $UDPOUT -j ACCEPT

#
# ALAPÉRTELMEZETT SZABÁLY és NAPLÓZÁS
#
# Minden más csomagra az alapértelmezett szabály érvényes, vagyis
# eldobjuk azokat. Ha a LOGGING változót előbb beállítottuk, ezekről
# naplóbejegyzés készül.
#

if [ "$LOGGING" ]
then
  iptables -A FORWARD -p tcp  -j LOG  # Az eldobott TCP csomagok naplózása
  iptables -A FORWARD -p udp  -j LOG  # Az eldobott UDP csomagok naplózása
  iptables -A FORWARD -p icmp -j LOG  # Az eldobott ICMP csomagok naplózása
fi
```

Számos egyszerű esetben a minta használatához mindössze azt kell megadnunk a „FELHASZNÁLÓI BEÁLLÍTÁSOK" részben, hogy milyen protokollokat és csomagokat szeretnénk ki- és beengedni. Összetett esetekben az utolsó rész módosítására is szükség lehet. Ne feledjük, hogy ez csak egyszerű példa, ezért mielőtt alkalmazzuk, alaposan nézzük át, hogy valóban azt teszi, amit elvárunk tőle.

Irodalom

A tűzfalak beállításáról és tervezéséről elegendő információt találhatunk, hogy köteteket töltsünk meg vele. Lássunk most néhány hasznos referenciaművet, melyekkel elmélyíthetjük tudásunkat:

Real World Linux Security, Second Edition
Írta Bob Toxen (Prentice Hall). Nagyszerű kötet, amely átfogó ismereteket kínál a biztonság témaköréről, beleértve a tűzfalakat is.

Building Internet Firewalls, Second Edition
Írta E. Zwicky, S. Cooper és D. Chapman (O'Reilly). A kézikönyv bemutatja, hogyan tervezzünk és telepítsünk tűzfalakat Unix, Linux és Windows NT rendszerekre, és hogyan állítsuk be az internetszolgáltatásokat, hogy együttműködjenek a tűzfalakkal.

Firewalls and Internet Security, Second Edition
Írta W. Cheswick, S. Bellovin és A. Rubin (Addison Wesley). A könyv a tűzfalak tervezésének és megvalósításának elméletébe nyújt betekintést.

Practical Unix & Internet Security, Third Edition
Írta S. Garfinkel, G. Spafford és A. Schwartz (O'Reilly). A könyv a népszerű Unix változatok (köztük a Linux) biztonsági kérdéseinek széles körével foglalkozik, például a „látleletekkel", a behatolásérzékeléssel, a tűzfalakkal és még sok egyébbel.

Linux Security Cookbook (magyar kiadása: Linux biztonság eljárások)
Írta D. Barrett, R. Silverman és R. Byrnes (O'Reilly, Kossuth Kiadó). A könyv több mint 150 azonnal felhasználható szkriptet és konfigurációs állományt tartalmaz a legfontosabb biztonsági feladatok megoldásához, például a hálózati hozzáférés napszaktól függő szabályozásához, a webkiszolgálók tűzfalkezeléséhez, az IP hamisítás megakadályozásához és számos más feladathoz.

Linux iptables Pocket Reference
Írta G. Purdy (O'Reilly). A könyv a tűzfalak alapelvével és a Linux csomagfeldolgozó mechanizmusával foglalkozik, de teljes referenciát tartalmaz az *iptables* parancsokról is, többek között egy enciklopédikus referenciát az illesztési és cél bővítményekhez, amelyeket fejlett alkalmazásokhoz használhatunk.

8

IP nyomkövetés

A kereskedelmi jellegű internetszolgáltatások világában egyre fontosabb tudnunk, hogy hálózati kapcsolatainkon mennyi adatot fogadunk és küldünk el. Mindez létfontosságú cégünk szempontjából, ha vállalkozásunk internetszolgáltatással foglalkozik és felhasználóinknak a forgalmazott adatmennyiség után számlázunk. De egyszerű felhasználóként is fontos lehet az adatgyűjtés, mert ha internetszolgáltatónk a bonyolított forgalom után küldi a számlát, így ellenőrizhetjük, hogy pontosan számolt-e.

A hálózati nyomkövetés azonban másra is alkalmas, olyan feladatokra, amelyeknek semmi közük a forintokhoz és a számlákhoz. Tegyük fel, hogy sokféle hálózati szolgáltatást kínáló kiszolgálót üzemeltetünk. Ekkor célszerű tisztában lennünk azzal, hogy az egyes szolgáltatások mennyi adatot dolgoznak fel. Az információ alapján könnyebben eldönthetjük például, milyen hardvert érdemes vásárolnunk, vagy hány kiszolgálót érdemes futtatnunk.

A Linux rendszermag biztosít egy szolgáltatást, amelynek segítségével a legkülönbözőbb adatokat gyűjthetjük össze a figyelt hálózati forgalomról. A szolgáltatás neve *IP nyomkövetés* (IP Accounting).

Az IP nyomkövetés beállítása a rendszermagban

A Linux IP nyomkövetés szolgáltatása közeli kapcsolatban áll a Linux tűzfal szoftverével. Általában ugyanazokon a pontokon szeretnénk adatokat gyűjteni, ahol a tűzfalas szűrést is végezzük: a hálózati gazdagépek kimenő és bejövő pontjain, illetve a csomagok útválasztásáért felelő szoftverben. Ha még nem olvastuk el a tűzfalakról szóló részt, most itt az alkalom, mert a 7. fejezetben leírt elveket itt is felhasználjuk!

Az IP nyomkövetés beállítása

Mivel az IP nyomkövetés közeli kapcsolatban áll az IP tűzfallal, a beállítását is egyazon eszközzel oldották meg, nevezetesen az **iptables** paranccsal. A parancs szintaxisa hasonlít a tűzfalszabályokéra, ezért ezzel most nem foglalkozunk; inkább azt nézzük meg, mit tudhatunk meg a szolgáltatással a hálózati forgalom természetéről.

Az utasítás általános szintaxisa:

```
# iptables -A lánc szabályleírás
```

Az **iptables** lehetővé teszi, hogy a vizsgálni kívánt forgalom irányát a tűzfalszabályokhoz hasonlóan határozzuk meg.

A parancsokat a tűzfalszabályokkal azonos módon adjuk meg, az eltérés mindössze annyi, hogy az eljárási szabályok (policy rules) itt nem érvényesek. A parancsokkal nyomkövetési szabályokat hozhatunk létre, szúrhatunk be, törölhetünk vagy listázhatunk. Az *ipchains* és **iptables** esetében minden érvényes szabály nyomkövetési szabály, és minden utasítás, amelyben nem adjuk meg a `-j` opciót, kizárólag nyomkövetést végez.

Az IP nyomkövetés szabályainak paraméterei megegyeznek az IP tűzfalaknál leírtakkal. Segítségükkel határozzuk meg pontosan, milyen hálózati forgalmat szeretnénk számlálni és összegezni.

Cím alapú nyomkövetés

Vizsgáljuk meg egy példán keresztül, hogyan használhatjuk az IP nyomkövetést! Képzeljük el, hogy van egy Linux rendszert futtató útválasztónk, amely a Virtuális Sörgyár két részlegét szolgálja ki. Az útválasztóhoz tartozik két Ethernet eszköz, az `eth0` és az `eth1`, ezek egy-egy részlegre kapcsolódnak; hozzátartozik még egy PPP eszköz is, a `ppp0`, amely egy nagy sebességű soros kapcsolaton keresztül a Groucho Marx Egyetemre csatlakozik.

Tegyük fel továbbá, hogy a számlázás miatt szeretnénk tudni, mekkora forgalmat bonyolítanak le összesen a soros kapcsolaton keresztül az egyes részlegek, a rendszer kezelése szempontjából viszont jó volna tudnunk azt is, mekkora a teljes forgalom a két részleg között.

A 8.1. táblázat a példában alkalmazott interfészcímeket mutatja.

8.1. táblázat. Az interfészek és címeik

Interfész	*Cím*	*Hálózatmaszk*
eth0<C	172.16.3.0	255.255.255.0
eth1<C	172.16.4.0	255.255.255.0

A „Mekkora adatforgalmat bonyolítanak az egyes részlegek a PPP kapcsolaton keresztül?" kérdésre a következő szabállyal kaphatunk választ:

```
# iptables -A FORWARD -i ppp0 -d 172.16.3.0/24
# iptables -A FORWARD -o ppp0 -s 172.16.3.0/24
# iptables -A FORWARD -i ppp0 -d 172.16.4.0/24
# iptables -A FORWARD -o ppp0 -s 172.16.4.0/24
```

Az első két szabály azt mondja, „Számold a `ppp0` interfészen bármely irányban keresztülhaladó adatokat, ha azok rendeltetési vagy forráscíme a **172.16.3.0/24**!". A második

szabálycsoport ugyanezt teszi, csupán a hálózati helyünk második Ethernet hálózata esetén.

A „Mekkora forgalmat bonyolít a két részleg egymás között?" kérdés megválaszolásához a következő szabályra van szükségünk:

```
# iptables -A FORWARD -s 172.16.3.0/24 -d 172.16.4.0/24
# iptables -A FORWARD -s 172.16.4.0/24 -d 172.16.3.0/24
```

A szabályok számba vesznek minden csomagot, amelynek forráscíme az egyik részleg hálózatához, a rendeltetési címe pedig a másikhoz tartozik.

Nyomkövetés a szolgáltatási port alapján

Most tegyük fel, hogy azt is szeretnénk tudni, milyen jellegű adatok haladnak keresztül PPP kapcsolatunkon. Szeretnénk például látni, hogy a kapcsolatot milyen arányban terhelik az FTP, az SMTP és a World Wide Web (HTTP) szolgáltatások.

A kívánt adatok begyűjtését végző szkript a következő szabályokat tartalmazza:

```
#!/bin/sh
# A PPP kapcsolat ftp, smtp and www mennyiségi statisztikájának
# begyűjtése az iptables segítségével.
#
iptables -A FORWARD -i ppp0 -p tcp --sport 20:21
iptables -A FORWARD -o ppp0 -p tcp --dport 20:21
iptables -A FORWARD -i ppp0 -p tcp --sport smtp
iptables -A FORWARD -o ppp0 -p tcp --dport smtp
iptables -A FORWARD -i ppp0 -p tcp --sport www
iptables -A FORWARD -o ppp0 -p tcp --dport www
```

Érdemes a konfiguráció néhány részletét külön is szemügyre vennünk. Először is meghatároztuk a protokollt. Amikor szabályainkban portokat adunk meg, egyben a protokollt is meg kell határoznunk, mert a TCP és UDP önálló portkészletet biztosít. Másodszor, az `ftp` és `ftp-data` szolgáltatásokat egyetlen parancsban adtuk meg. Az **iptables** lehetővé teszi, hogy a portokat önállóan vagy tartományként definiáljuk, és itt pontosan ezzel az utóbbi lehetőséggel éltünk. A „20:21" szintaxis jelentése „a 20-as (`ftp-data`) porttól a 21-es (`ftp`) portig"; az **iptables** programban így adhatunk meg porttartományt (a `tcp` illesztési bővítmény megengedi, hogy a tartományok kijelölésénél a portok neveit használjuk, a `multiport` illesztési bővítmény azonban nem – mindenképpen jobban járunk, ha a portokat számokkal jelöljük, így nem fordulhat elő, hogy véletlenül több portot adunk meg, mint szeretnénk). Amikor egy nyomkövető szabályban több portot is meghatározunk, a listában szereplő bármely portra érkező adat a bejegyzés végösszegéhez adódik. Az FTP szolgáltatás két portot használ, egyet a parancsok részére és egyet az adatátvitelhez; mi most mindkettőt az FTP forgalom „számlájához" írtuk.

A második pontot kissé kibővítve eltérő szemszögből figyelhetjük meg a kapcsolaton áthaladó adatokat. Tegyük fel, hogy az FTP, az SMTP és a World Wide Web (HTTP) forgalmat elsődleges forgalomnak nyilvánítjuk, minden más forgalmat viszont másodla-

gosnak. Ha például kíváncsiak lennénk az elsődleges és másodlagos forgalom arányára, így járnánk el:

```
# iptables -A FORWARD -i ppp0 -p tcp -m multiport \
    --sports ftp-data,ftp,smtp,www -j ACCEPT
# iptables -A FORWARD -j ACCEPT
```

Az első szabály az elsődleges forgalmat, a második szabály pedig az egyéb forgalmat számlálja.

További megoldás lehet egy felhasználói lánc alkalmazása (erre akkor van szükség, ha az elsődleges forgalmat meghatározó szabályok nagyon összetettek):

```
# iptables -N a-fontos
# iptables -N a-nemfontos
# iptables -A a-fontos -j ACCEPT
# iptables -A a-nemfontos -j ACCEPT
# iptables -A FORWARD -i ppp0 -p tcp -m multiport \
    --sports ftp-data,ftp,smtp,www -j a-fontos
# iptables -A FORWARD -j a-nemfontos
```

Két felhasználói láncot hoztunk létre – az elsődleges szolgáltatások nyomkövetését az `a-fontos` lánc végzi, a másodlagos szolgáltatások felügyelete pedig az `a-nemfontos` feladata. Ezután a `FORWARD` lánc szabályait adjuk meg, amelyek az elsődleges szolgáltatásokra illeszkednek és az `a-fontos` láncra ugranak; ez utóbbi mindössze egyetlen szabályt tartalmaz, amely minden csomagot elfogad és számba vesz. A `FORWARD` lánc utolsó szabálya egyszerűen az `a-nemfontos` láncra ugrik, ahol ismét csak egy minden csomagot elfogadó és számláló szabályt helyeztünk el. Az `a-nemfontos` láncra ugró szabályhoz egyetlen elsődleges szolgáltatásunk sem ér el, mivel ezeket saját láncaik fogadják. Ezért az elsődleges és másodlagos szolgáltatások eredménye a megfelelő láncok szabályaiban érhető el. Ez csupán egy lehetséges megoldás, természetesen mások is léteznek.

Egyszerűnek tűnik! Sajnos, van azonban egy kicsi, de megkerülhetetlen gondunk a szolgáltatási típusok nyomkövetésével. Emlékezzünk vissza, hogy egy korábbi fejezetben megvizsgáltuk az MTU szerepét az TCP/IP hálózatkezelésben. Az MTU egy hálózati eszközön átvihető csomag maximális méretét határozza meg. Amikor egy útválasztóhoz olyan csomag érkezik, amely nagyobb, mint a továbbítást végző interfész MTU-ja, az útválasztó cselhez folyamodik: a *fragmentáláshoz*. A nagy csomagokat kisebb, az interfész MTU-jánál rövidebb darabokra tördeli és így küldi tovább. Az útválasztó új fejléceket készít és a darabkák elé helyezi, így a távoli gazdagép képes helyreállítani az eredeti adatokat. Sajnos a tördelés azzal jár, hogy a port megjelölése a továbbiakban már csak az első töredékben található meg, vagyis az IP nyomkövetés nem képes a tördelt csomagokat helyesen számlálni. Csupán az első töredéket, illetve a nem tördelt csomagokat számlálja megbízhatóan. Azért, hogy a második és a további töredékeket is biztosan elfogjuk, egy ehhez hasonló szabályt alkalmazhatunk:

```
# iptables -A FORWARD -i ppp0 -m tcp -p tcp -f
```

Ebből ugyan még nem tudjuk meg, hogy az adatok eredetileg melyik portról származtak, de legalább azt látjuk, hogy adataink mekkora részben állnak töredékekből, így figyelembe vehetjük, hogy mekkora részét teszik ki a forgalomnak.

A kapcsolatkövetés automatikusan újra összeállítja (defragmentálja) a tördelt csomagokat, így az eljárás ilyen esetben csak ritkán kecsegtet haszonnal. Ha azonban nem futtatunk kapcsolatkövetést, érdemes használnunk.

Az ICMP csomagok nyomkövetése

Az ICMP protokoll nem alkalmaz szolgáltatási portszámokat, és ezért nehezebb róla adatokat gyűjtenünk. Az ICMP számos különféle csomagot használ, amelyek közül sok ártalmatlan és szabályos, másokkal viszont csak különleges körülmények között lenne szabad találkoznunk. Előfordul azonban, hogy egyesek - nehogy halálra unják magukat - inkább ICMP üzenetekkel árasztják el a felhasználót, tönkretéve hálózati hozzáférését. A rosszindulatú támadás neve *ping flooding* (az ilyen típusú szolgáltatásmegtagadás célú támadások általános elnevezése *packet flooding*, de a ping flooding is elterjedt; a hivatalos neve 'csomag elárasztás'). Noha az IP nyomkövetéssel a támadást nem akadályozhatjuk meg (viszont az IP tűzfal segíthet!), arra azonban jó, hogy a megfelelő nyomkövető szabályok riadót fújjanak, ha valaki ezzel próbálkozik.

Az ICMP nem használ portokat, mint a TCP vagy UDP, rendelkezik viszont ICMP üzenettípusokkal. Készíthetünk tehát olyan szabályokat, amelyek az egyes ICMP üzenettípusoknak felelnek meg. Az ICMP üzenetszámot és típusszámot a nyomkövető parancsok port mezőjének helyén adjuk meg.

A bennünket érő és az általunk küldött ping adatok mennyiségéről egy ehhez hasonló IP nyomkövető szabállyal szerezhetünk információt:

```
# iptables -A FORWARD -m icmp -p icmp --sports echo-request
# iptables -A FORWARD -m icmp -p icmp --sports echo-reply
# iptables -A FORWARD -m icmp -p icmp -f
```

Az első szabály az „ICMP Echo Request" (ping kérés), míg a második szabály az „ICMP Echo Reply" (ping válasz) csomagokról gyűjt adatokat. A harmadik szabály az ICMP csomagtöredékeket figyeli. Az eljárás hasonlít a tördelt TCP és UDP csomagoknál leírtakhoz.

Ha szabályainkban megadjuk a forrás- és/vagy a rendeltetési címet, nyomon követhetjük, honnan indulnak a pingek, például hogy a hálózatunkon belülről vagy azon kívülről származnak-e. Miután már tudjuk, eldönthetjük, hogy tűzfalszabályokkal kívánjuk-e kiszűrni azokat, esetleg más megoldást keresünk, például felhívjuk a támadó hálózat tulajdonosának figyelmét a problémára, vagy ha rosszindulatú tevékenységről van szó, jogi lépéseket teszünk.

Nyomkövetés a protokoll alapján

Tegyük fel, hogy azt szeretnénk tudni, a kapcsolatunk forgalmának mekkora része TCP, UDP vagy ICMP. Ekkor ehhez hasonló szabályokat alkalmazhatunk:

```
# iptables -A FORWARD -i ppp0 -m tcp -p tcp
# iptables -A FORWARD -o ppp0 -m tcp -p tcp
# iptables -A FORWARD -i ppp0 -m udp -p udp
# iptables -A FORWARD -o ppp0 -m udp -p udp
# iptables -A FORWARD -i ppp0 -m icmp -p icmp
# iptables -A FORWARD -o ppp0 -m icmp -p icmp
```

A szabályok a ppp0 interfészen átáramló forgalmat elemezve meghatározzák, hogy az vajon TCP, UDP vagy ICMP forgalom-e, majd frissítik a megfelelő számlálókat.

Munka az IP nyomkövetés eredményével

Nem rossz dolog, ha gyűjtjük az adatokat, de vajon hogyan nézhetjük meg azokat? A begyűjtött nyomkövetési adatokat és a beállított nyomkövetési szabályokat a tűzfal konfigurációs parancsaival tekinthetjük meg, ha lekérjük a szabályok listáját. Az egyes szabályok csomag- és bájtszámlálóit megtaláljuk a kimenetben.

A nyomkövetési adatok listázása

Az **iptables** parancs az *ipchains* parancshoz hasonlóan működik. A szabályok kiíratásakor a nyomkövető számlálók megjelenítéséhez ismét csak meg kell adnunk a `-v` opciót. A nyomkövetési információ listázásához a következő parancsot adjuk ki:

```
# iptables -L -v
```

Mint az *ipchains* parancs esetében is, a `-x` argumentum hatására a kimenetben a rövidített formátum helyett a pontos értékeket láthatjuk.

A számlálók nullázása

Ha elég sokáig használjuk őket, az IP nyomkövetés számlálói túlcsordulnak. Ha túlcsordultak, már igen nehéz megmondanunk, hogy valójában milyen értékeket mutatnak. Azért, hogy ezt elkerüljük, időnként olvassuk le és jegyezzük fel az adatokat, majd állítsuk a számlálókat nullára, hogy újrakezdhessék a számlálást a következő időszakra.

Az **iptables** parancs erre egyszerű lehetőséget biztosít:

```
# iptables -Z
```

Azt is megtehetjük, hogy a listázást és a nullázást kombináljuk, így biztosítva, hogy közben ne álljon le a nyomkövetési adatok gyűjtése:

```
# iptables -L -Z -v
```

A parancs először kilistázza a nyomkövetési információt, majd azonnal lenullázza a számlálókat és újrakezdi a számlálást. Ha az adatokat rendszeresen szeretnénk gyűjteni és használni, a parancsot elhelyezhetjük egy szkriptben, amely valahol rögzíti és tárolja az adatokat, majd a cron paranccsal adott időközönként lefuttathatjuk.

A szabálykészlet törlése

Lássunk még egy hasznos parancsot, amellyel az összes beállított IP nyomkövetési szabályt törölhetjük. Ez akkor tehet jó szolgálatot, ha a szabálygyűjteményt alapjaiban szeretnénk módosítani anélkül, hogy újraindítanánk a gazdagépet.

Az **iptables** parancs `-F` argumentuma a megadott típusú összes szabályt törli:

```
# iptables -F
```

A parancs az összes beállított szabályt eltávolítja (nem csak a nyomkövetési szabályokat), így nem kell azokat egyesével törölnünk.

A nyomkövetési adatok passzív gyűjtése

Még egy apró fortély: ha Linux gazdagépünk Ethernet hálózatra csatlakozik, a teljes hálózati szegmens összes adatára alkalmazhatunk nyomkövetési szabályokat, nem csak azokra, amelyeket a gazdagép közvetít, illetve a gazdagépnek címeztek. A gazdagép a szegmens összes adatát passzívan figyeli és számlálja.

Mindenekelőtt kapcsoljuk ki az IP továbbítást a Linux gépen, hogy ne próbálja a kapott csomagokat továbbítani.* Ezt így tehetjük meg:

```
# echo 0 >/proc/sys/net/ipv4/ip_forward
```

Ezután kapcsoljuk be az Ethernet interfész *lehallgató* (promiscuous) *üzemmódját* az ifconfig utasítással. A lehallgató üzemmód hatására az Ethernet interfész valamennyi csomagot az operációs rendszerhez továbbít, nem csak azokat, amelyeknek a rendeltetési címe a saját Ethernet címe. Ez akkor érdekes, ha az eszköz üzenetszóró médiumra csatlakozik (például nem kapcsolt Ethernetre). Az `eth1` interfészen a lehallgató üzemmódot így kapcsolhatjuk be:

```
# ifconfig eth1 promisc
```

Most már elkészíthetjük az Etherneten áramló csomagok nyomkövetési szabályait, és úgy gyűjthetünk adatokat, hogy a Linux nyomkövető gépünk nem is esik a csomagok útvonalába!

* Ez persze nem lehetséges, ha Linux gépünk útválasztóként üzemel. Ha kikapcsoljuk az IP továbbítást, azzal az útválasztást is megszűntetjük. Csak olyan gépen tegyük ezt, amely egyetlen fizikai hálózati interfésszel rendelkezik.

9

IP álcázás és hálózati címfordítás

Nem is kell hozzá különlegesen jó memória, hogy emlékezzünk arra az időre, amikor még csak a nagyvállalatok engedhették meg maguknak a néhány számítógépből álló lokális hálózatok kialakítását. A hálózatok kialakításához szükséges berendezések árának csökkenésével azonban két dolog történt. Az első, hogy a lokális hálózatok elterjedtek, beférkőzve még otthonainkba is: ma már sok Linux felhasználó rendelkezik két vagy több számítógéppel, amelyeket valamilyen Ethernet köt össze. A második, hogy a hálózati erőforrások, különösen az IP címek kifogyóban vannak, és régen ugyan ingyenesek voltak, ma már adják-veszik őket.

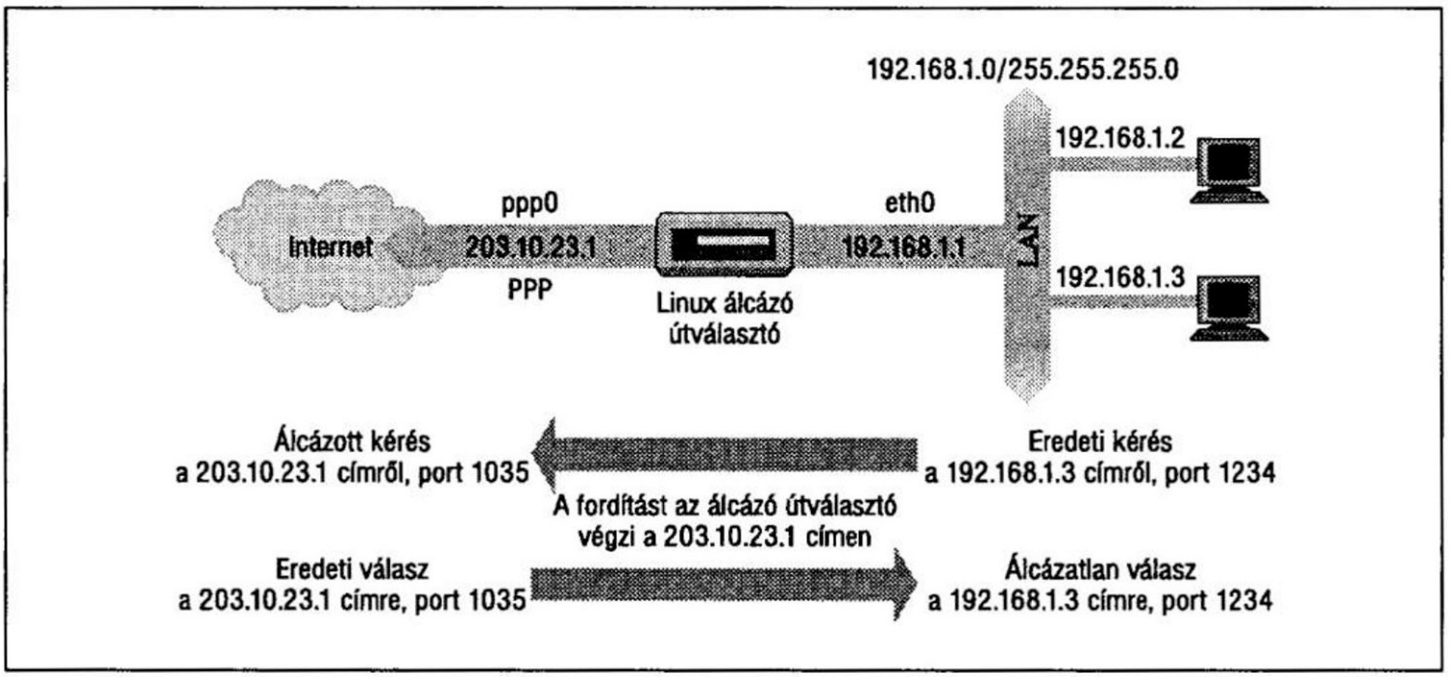
A lokális hálózattal (LAN) rendelkező felhasználók általában igényt tartanak olyan internetkapcsolatra, amelyhez a LAN minden gépe hozzáférhet. Az IP útválasztó szabályok a helyzetet szigorúan kezelik. A hagyományos megoldások esetén ilyenkor kérnünk kellene egy IP hálózati címet - kis hálózati helyekhez például egy C osztályút -, majd e hálózatból rendelnénk a LAN minden gazdagépéhez egy címet, és a lokális hálózatot útválasztón keresztül kapcsolnánk az internetre.

A kereskedelmi jelleget öltő internetes környezetben ez bizony költséges beruházás. Fizetnünk kell a kapott hálózati címért, és valószínűleg az internetszolgáltatónk is pénzt kér azért, hogy a hálózatunkhoz megfelelő útvonalat alakítson ki, és így mások is elérhessenek bennünket. Noha a vállalatok számára ez megfelelő megoldás lehet, az otthoni rendszerek esetén általában nem kifizetődő.

Szerencsére a Linux erre is kínál megoldást, a Linux fejlett hálózatkezelő tulajdonságainak egyik komponensét, a *hálózati címfordítást* (Network Address Translation, NAT). A NAT az átvitel alatt álló csomagok fejlécében tárolt hálózati címek (és esetenként portszámok) módosításának folyamatát írja le. Első hallásra ez talán furcsának tűnhet, de hamarosan látni fogjuk, hogy ideális az imént vázolt probléma megoldására. A hálózati címfordítás egyik típusának neve *IP álcázás* (IP Masquerading). Az IP álcázás lehetővé teszi, hogy egy magánhálózat összes gazdagépe hozzáférjen az internethez, csupán egyetlen dinamikus IP cím áráért. A statikus IP címek esetén az eljárás neve SNAT (*Source NAT*). A következőkben mindkét eljárásra az „álcázás" kifejezést alkalmazzuk.

Az IP álcázással a LAN hálózatunk gazdagépeihez (útválasztásra alkalmatlan) magán IP hálózati címet használhatunk, miközben Linux alapú útválasztónk ravasz, valós idejű IP cím- és portfordítást végez. Amikor az útválasztó a LAN egy gazdagépétől csomagot kap, megvizsgálja a csomag típusát (például TCP, UDP vagy ICMP), és módosítja a csomagot, hogy úgy nézzen ki, mintha azt saját maga állította volna elő (a módosítást pedig számon tartja). Ezután továbbküldi a csomagot az internetre, saját egyetlen kapcso-

latának IP címével. Amikor a rendeltetési gazdagép megkapja a csomagot, feltételezi, hogy az útválasztó gazdagéptől származik, így a válaszcsomagot annak a címére küldi. Amikor a Linux álcázó útválasztóhoz az internetkapcsolatról megérkezik a válaszcsomag, az útválasztó a futó álcázott kapcsolatokat tartalmazó táblázatában ellenőrzi, hogy a csomag a LAN valamelyik gépéhez tartozik-e. Amikor látja, hogy igen, visszaállítja a küldéskor végrehajtott módosításokat és a LAN géphez küldi a csomagot. A 9.1. ábrán erre láthatunk példát.



9.1. ábra. Egy tipikus IP álcázó kialakítása

Tegyük fel, hogy rendelkezünk egy kis Ethernet hálózattal, amely egy fenntartott hálózati címet használ. A hálózatban egy Linux alapú álcázó útválasztó biztosítja a hozzáférést az internethez. A hálózat **192.168.1.3** című munkaállomása szeretne kapcsolatot teremteni a **203.10.23.1** című távoli gazdagéppel. A munkaállomás a csomagokat az álcázó útválasztóhoz irányítja, amely a kapcsolatkérést az álcázási szolgáltatás kéréseként azonosítja. Elfogadja a csomagot, kijelöl számára egy portot (1035), majd az eredeti gazdagép IP címe és portszáma helyére a sajátját helyezi, és elküldi a rendeltetési gazdagéphez. A rendeltetési gazdagép úgy véli, hogy a Linux álcázó gazdagép küldött neki kapcsolati kérést, és előállít egy válaszcsomagot. Amikor az álcázó gazdagép megkapja ezt a csomagot, visszafordítja a kimenő csomagon végzett műveletet, és a válaszcsomagot az eredeti gazdagéphez juttatja.

A helyi gazdagép azt gondolja, hogy közvetlenül a távoli gazdagéppel áll kapcsolatban. A távoli gazdagép viszont egyáltalán nem ismeri a helyi gazdagépet, és azt hiszi, hogy a kapcsolat a Linux álcázó gazdagéptől származik. Csak a Linux álcázó gazdagép tudja, hogy a két másik gép egymással kommunikál; ismeri a használt portokat, és elvégzi a kommunikációhoz szükséges cím- és portfordítást.

Ha mindez kissé zavarosnak tűnik is, ami könnyen meglehet, az eljárás azért még működik, és valójában egyszerűen beállítható. Ezért ne aggódjunk, ha még nem értünk minden részletet.

Mellékhatások és járulékos előnyök

Az IP álcázó szolgáltatásnak megvannak a maga mellékhatásai, amelyek közül egyesek hasznosak, mások egy idő után zavaróak.

Az álcázó útválasztó mögötti hálózat egyetlen gazdagépe sem látszik soha közvetlenül; következésképpen csupán egyetlen érvényes és útválasztásra alkalmas IP címre van szükségünk ahhoz, hogy valamennyi gazdagép létrehozhasson kimenő hálózati kapcsolatot az internethez. Ennek az a hátránya, hogy az internetről egyik gazdagép sem látszik és közvetlenül egyik sem kapcsolódhat az internetre; az álcázott hálózat egyetlen látható gazdagépe maga az álcázó gazdagép. Ez olyan szolgáltatásoknál lehet fontos, amilyen a levelezés vagy az FTP, mert segíthet eldönteni, milyen szolgáltatásokat biztosítson az álcázó gazdagép és milyen szolgáltatásokat kell proxyként vagy egyéb különleges módon kezelnie.

Az útválasztón azonban futhat a DNAT (*Destination NAT*), amellyel a bejövő kapcsolatokat a belső kiszolgálók bizonyos portjaira irányíthatjuk. A webkiszolgálók és levelezőkiszolgálók esetén ez kiválóan működik. A szolgáltatásokat futtathatjuk a magánhálózat gazdagépein, míg a DNAT a megfelelő belső kiszolgálók 80-as és 25-ös portjaira továbbítja a bejövő kapcsolatokat. Ily módon az útválasztó csupán az útválasztásban játszik szerepet, a kívülről elérhető szolgáltatások biztosításában nem. Az eljárás arra is alkalmas, hogy segítségével a magas portszámú (mondjuk 4022) bejövő kapcsolatokat egy gazdagép Secure Shell (SSH) portjára irányítsuk (ez rendszerint a 22-es), így közvetlenül hozhatunk létre SSH kapcsolatot egy belső gazdagéppel az útválasztón keresztül.

Mivel az álcázott gazdagépek egyike sem látható, a külső támadásokkal szemben aránylag védettek. Az egyik gazdagépünk szolgálhat tűzfalként és álcázó útválasztóként. Ekkor a teljes hálózat annyira biztonságos, amennyire az álcázó gazdagép, ezért védjük tűzfalszabályokkal és ne futtassunk rajta más, kívülről is látható szolgáltatásokat.

Az IP álcázás a hálózat teljesítményére is kihat. Egy tipikus konfiguráción ez szinte alig mérhető. Ha azonban sok aktív álcázó munkafolyamat fut, észrevehetjük, hogy az álcázó gazdagéptől megkívánt feldolgozás a hálózat átviteli teljesítményét is befolyásolja. A hagyományos útválasztással összevetve az IP álcázás minden csomag esetén jókora munkát kíván. Ezért a személyes internetkapcsolatunkhoz álcázó gazdagépként kiszemelt kis teljesítményű számítógép megfelelő megoldás lehet, de ne várjunk tőle sokat, ha egyszer úgy döntünk, hogy vállalatunk hálózatában állítjuk üzembe útválasztóként, az Ethernet sebessége mellett.

Végül vannak olyan hálózati szolgáltatások, amelyek egyszerűen nem működnek az álcázáson keresztül, vagy legalábbis rengeteg támogatást igényelnek. E szolgáltatások munkája jellemzően a bejövő munkafolyamatoktól függ; ilyen például a Direct Communications Channels (DCC) némely típusa, az IRC lehetőségei, vagy bizonyos többcímes (multicast) video- és audio-üzenetszóró szolgáltatások egyes típusai. E szolgáltatásokhoz néha találunk külön e célra fejlesztett „kisegítő" rendszermagmodulokat, amelyekről hamarosan szólunk. Másoknál viszont előfordulhat, hogy nem találunk hozzájuk támogatást – ezért ne feledjük: az álcázás nem minden helyzetben megfelelő megoldás!

A rendszermag beállítása az IP álcázáshoz

Az IP álcázó szolgáltatáshoz a rendszermagot a hálózati csomagszűrés támogatása mellett kell fordítanunk. A rendszermag beállításakor a következő opciókat válasszuk ki:

```
Networking options --->
  [M] Network packet filtering (replaces ipchains)
```

A *netfilter* csomag tartalmaz olyan modulokat, amelyek segítik az álcázó szolgáltatást. Az FTP munkafolyamatok kapcsolatkövetésének biztosításához például betölthetjük az *ip_conntrack_ftp* és *ip_nat_ftp.o* modulokat. A kapcsolatkövetés elengedhetetlen ahhoz, hogy az álcázás helyesen működjön együtt egy logikai munkafolyamathoz több kapcsolatot is igénylő protokollokkal, hiszen az álcázás a kapcsolatkövetésen alapul.

Az IP álcázás beállítása

Ha a tűzfalról és a nyomkövetésről szóló fejezeteket már elolvastuk, valószínűleg nem okoz meglepetést, hogy az IP álcázás szabályait is az *iptables* paranccsal állíthatjuk be.

Az álcázás egy különleges típusú csomagmanipuláció (packet mangling; a csomagok módosítására utaló műszaki kifejezés). Egy interfészre érkező csomagok közül csak azokat álcázhatjuk, amelyeket egy másik interfészre irányítunk. Az álcázó szabályokat a tűzfalszabályokhoz hasonló módon hozzuk létre, de egy különleges opcióval a csomag álcázására utasítjuk a rendszermagot. Az *iptables* parancs a `-j MASQUERADE` opcióval jelzi, hogy a szabálynak megfelelő csomagokat álcázni kell (dinamikus IP cím esetén; statikus IP címnél a `-j SNAT` opciót adjuk meg).

Lássunk egy példát! A Groucho Marx Egyetem egyik informatikus hallgatója otthonában felállít egy kis, néhány számítógépet tartalmazó Ethernet alapú LAN hálózatot, és úgy dönt, hogy egy fenntartott hálózati címet választ a hálózathoz. Otthonát azonban más hallgatókkal osztja meg, akik mindannyian szívesen használnák az internetet. Mivel a hallgatók anyagi lehetőségei igen szűkösek, nem engedhetnek meg maguknak állandó internetkapcsolatot, ezért egyetlen internetkapcsolat mellett döntenek. Szeretnék tehát megosztani a kapcsolatot, hogy IRC beszélgetéseket folytassanak, barangoljanak a neten, vagy az FTP segítségével állományokat töltsenek le saját gépükre. A megoldás az IP álcázás.

A hallgató mindenekelőtt beállít egy Linux gazdagépet az internetkapcsolat támogatására, illetve hogy a LAN útválasztójaként működjön. Az IP cím, amelyet a betárcsázáskor kap, lényegtelen. A Linux útválasztót IP álcázásra állítja be, és a hálózathoz az egyik magánhálózati címet használja: **192.168.1.0**. Ezután ellenőrzi, hogy az alapértelmezett útvonal a LAN valamennyi gazdagépén a Linux útválasztóra mutat.

Most már csupán a következő utasításokra van szüksége ahhoz, hogy az álcázást működésre bírja az adott konfigurációban:

```
# iptables -t nat -P POSTROUTING DROP
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Valahányszor a LAN valamely gazdagépe megpróbál egy távoli gazdagép szolgáltatására kapcsolódni, a csomagokat a Linux álcázó útválasztó automatikusan álcázza. A példa első sora megtiltja a Linux gazdagépnek, hogy bármilyen más csomagot irányítson, és némi biztonságot is nyújt.

Az elkészített álcázószabályokat az *iptables* parancs -L argumentumával listázhatjuk ki, ahogy a tűzfalaknál is láttuk:

```
# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy DROP)
target     prot opt source               destination
MASQUERADE  all  --  anywhere             anywhere            MASQUERADE

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Az álcázószabályokat arról ismerjük meg, hogy céljuk (target) a MASQUERADE.

A névkiszolgáló kikereséseinek kezelése

Az IP álcázást alkalmazó LAN hálózat gazdagépeinek tartománynév-kiszolgáló kikereséseit mindig is csak körülményesen lehetett kezelni. A DNS beillesztésére egy álcázó környezetbe két lehetőség kínálkozik. Utasítjuk az összes gazdagépet, hogy azt a DNS-t használják, amelyiket a Linux útválasztó gazdagép is használ, és hagyjuk, hogy az IP álcázás elvégezze a varázslatot a DNS kéréseken. A másik lehetőség, hogy tároló névkiszolgálót futtatunk a Linux gazdagépen, és a LAN összes gépét úgy állítjuk be, hogy ezt a gépet használja DNS-ként. Bár a beavatkozás komolyabb, az utóbbi mégis jobb választásnak tűnik, mert csökkenti az internetkapcsolaton áramló DNS forgalmat, és a legtöbb kérés esetén némileg gyorsabb is lesz, mivel a kérések kiszolgálása a gyorsítótárból történik. A kialakítás hátulütője, hogy jóval összetettebb. Az 5. fejezet *A named csak tároló konfigurációja* című részéből megtudhatjuk, hogyan állíthatunk be tároló névkiszolgálót.

Ismét a hálózati címfordításról

A *netfilter* szoftver számos különféle NAT kezelésére képes, amelyek közül az IP álcázás csupán egyetlen, egyszerű alkalmazás.

Készíthetünk például olyan NAT szabályokat, amelyek csak bizonyos címeket vagy címtartományokat fordítanak és az összes többit érintetlenül hagyják, az olyanokat is, amelyek a címeket több címre fordítják, és nem csak egyre, mint az álcázás. Ami azt illeti, az *iptables* paranccsal – és a szabványos attribútumok, például a forráscím, a rendeltetési cím, a protokolltípus, a portszám stb. illesztések kombinációjával – előállíthatunk olyan NAT szabályokat, amelyek gyakorlatilag bármit le tudnak képezni.

Az *iptables* parancsban a csomagok forráscímének fordítását Source NAT-nak vagy SNAT-nak hívjuk. A csomagok rendeltetési címének fordítását Destination NAT-nak vagy DNAT-nak nevezzük. A SNAT és DNAT az *iptables* paranccsal használható célok, amelyekkel bonyolult szabályokat állíthatunk elő.

10

Fontos hálózati jellemzők

Az IP és a feloldó (DNS) sikeres beállítása után most vessünk egy pillantást a hálózaton biztosítani kívánt szolgáltatásokra. A fejezetben átnézzük néhány egyszerű hálózati alkalmazás, például az *inetd* és a *xinetd* kiszolgálók, illetve az *rlogin* család egyes programjainak beállítását. Röviden érintjük a Remote Procedure Call interfészt is, amely olyan szolgáltatások alapjául szolgál, mint például a Network File System (NFS). Az NFS beállítása azonban összetett feladat, ezzel a könyvben nem foglalkozunk.

Természetesen nem mutathatjuk be valamennyi hálózati alkalmazást. Ha olyan alkalmazást szeretnénk telepíteni, amelyre itt nem térünk ki, a kiszolgáló használati útmutatójából tájékozódhatunk.

Az inetd szuperkiszolgáló

A hálózaton keresztül alkalmazási szolgáltatásokat nyújtó programokat hálózati *daemonoknak* hívjuk. A daemon egy olyan program, amely megnyit egy – általában jól ismert – szolgáltatási portot, és várja a beérkező kapcsolatokat. Amikor ez bekövetkezik, a daemon létrehoz egy gyermek folyamatot, amely fogadja a kapcsolatot, míg a szülő folytatja a port figyelését, további kérésekre várva. Az eljárás jól működik, azonban van néhány hátránya. A biztosítani kívánt összes lehetséges szolgáltatásnak legalább egy példányának mindig aktívnak kell lennie a memóriában. Ráadásul a port figyelését és kezelését végző szoftverrutinok a hálózati daemon minden példányában megismétlődnek.

E hátrányok kiküszöbölésére a legtöbb Unix rendszer egy különleges hálózati daemont futtat, amelyet tekinthetünk „szuperdaemonnak" is. A szuperdaemon egy csomó szolgáltatás nevében foglalatokat (socket) hoz létre, és ezeket egyidejűleg figyeli. Amikor valamelyik foglalatra bejövő kapcsolat érkezik, a szuperkiszolgáló fogadja a kapcsolatot, és életre kelti a porthoz rendelt szolgáltatást, átadva a foglalat kezelését a gyermek folyamatnak. A szuperdaemon ezután visszatér munkájához és tovább figyel.

A legelterjedtebb szuperkiszolgáló neve *inetd,* ez az Internet Daemon. A rendszer betöltésekor indul el, és a kezelendő szolgáltatások listáját egy indítóállományból, az */etc/inetd.conf* állományból veszi. E kiszolgálók mellett az *inetd* további, kevésbé jelentős szolgáltatásokat is végez, ezeket *belső szolgáltatásoknak* nevezzük. Ilyen szolgáltatás többek között a *chargen,* amely egyszerűen csak karakterláncokat állít elő, és a *daytime,* amely a rendszer szerinti pontos időt adja meg.

Az állomány egy bejegyzése egyetlen sorból áll, és különféle mezőket foglal magában:

```
szolgáltatás típus protokoll várakozás felhasználó kiszolgáló parancssor
```

Nézzük meg, mit jelentenek az egyes mezők:

`szolgáltatás`
A szolgáltatás neve. A szolgáltatás nevét - az */etc/services* állományból kikeresve - le kell fordítanunk egy portszámra. A *services* állományról a fejezet *A services és protocols állományok* című részében bővebben szólunk.

`típus`
A foglalat (socket) típusa. Ez lehet `stream` (kapcsolatorientált protokollok esetén) vagy `dgram` (datagram protokollok esetén). A TCP alapú szolgáltatások ezért mindig a `stream`, az UDP alapú szolgáltatások pedig mindig a `dgram` típust használják.

`protokoll`
A szolgáltatás átviteli protokollja. Érvényes és azonos a *protocols* állományban szereplő protokollnévvel, amelyet később mutatunk be.

`várakozás`
Az opció csak a `dgram` foglalatokra vonatkozik, és értéke lehet a `wait` vagy a `nowait`. A `wait` esetén az *inetd* adott időben csak egyetlen kiszolgálót futtat a megadott porthoz. Ellenkező esetben az *inetd,* miután elindította a kiszolgálót, azonnal folytatja a figyelést a porton.

Az első megoldás az egyszálas kiszolgálóknál célravezető, amelyek addig olvassák a bejövő datagramokat, amíg jönnek, ha nem érkezik több, kilépnek. A legtöbb RPC szolgáltatás ilyen, ezért ezekhez a `wait` értéket adjuk meg. Az ellenkező típus, a többszálas kiszolgálók lehetővé teszik, hogy korlátlan számú példányuk fusson egyidejűleg. Ezekhez adjuk meg a `nowait` értéket.

A `stream` foglalatok értéke mindig a `nowait`.

`felhasználó`
Annak a felhasználónak a bejelentkezési azonosítója, aki a folyamat futása közben a folyamat tulajdonosa. Ez gyakran a *root* felhasználó, de egyes szolgáltatások eltérő fiókokat is használhatnak. Érdemes itt „a legkisebb jogosultság" elvét alkalmazni, vagyis ha egy parancs a helyes működéshez nem igényel különleges jogosultsággal rendelkező felhasználót, akkor ne futtassuk ilyen felhasználóként. Az NNTP hírkiszolgáló például *news*ként fut, míg a potenciális biztonsági kockázatot jelentő szolgáltatások (például a tftp vagy a finger) gyakran *nobody*ként.

`kiszolgáló`
A végrehajtandó kiszolgálóprogram teljes elérési útvonala. A belső szolgáltatásokat az *internal* kulcsszó jelzi.

`parancssor`

A kiszolgálónak átadott parancssor. A végrehajtandó kiszolgálóprogram neve nyitja meg, ezt követik a kívánt argumentumok. Amennyiben TCP burkolót (wrapper) használunk, a kiszolgáló teljes útvonalát adjuk meg a parancssorban; egyébként a kiszolgáló nevét úgy választhatjuk meg, ahogy azt a folyamatlistában viszontlátni szeretnénk. A TCP burkolóra hamarosan visszatérünk.

A belső szolgáltatások esetén a mező üres.

A 10.1. példa egy minta *inetd.conf* állomány. A finger szolgáltatást megjegyzésbe helyeztük, így nem érhető el. Biztonsági okokból gyakran választjuk ezt a megoldást, mert a programmal a támadók megszerezhetik a rendszer felhasználóinak nevét és más adatait.

10.1. példa. Minta /etc/inetd.conf állomány

```
#
# inetd szolgáltatások
ftp      stream tcp nowait root  /usr/sbin/ftpd    in.ftpd -l
telnet   stream tcp nowait root  /usr/sbin/telnetd in.telnetd -b/etc/issue
#finger  stream tcp nowait bin   /usr/sbin/fingerd in.fingerd
#tftp    dgram  udp wait  nobody /usr/sbin/tftpd   in.tftpd
#tftp    dgram  udp wait  nobody /usr/sbin/tftpd   in.tftpd /boot/diskless
#login   stream tcp nowait root  /usr/sbin/rlogind in.rlogind
#shell   stream tcp nowait root  /usr/sbin/rshd    in.rshd
#exec    stream tcp nowait root  /usr/sbin/rexecd  in.rexecd
#
#       inetd belső szolgáltatások
#
daytime  stream tcp nowait root internal
daytime  dgram  udp nowait root internal
time     stream tcp nowait root internal
time     dgram  udp nowait root internal
echo     stream tcp nowait root internal
echo     dgram  udp nowait root internal
discard  stream tcp nowait root internal
discard  dgram  udp nowait root internal
chargen  stream tcp nowait root internal
chargen  dgram  udp nowait root internal
```

A *tftp* szintén megjegyzésben áll. A *tftp* a *Trivial File Transfer Protocol* (TFTP) megvalósítása. A protokollal bárki letöltheti rendszerünkből a mindenki számára olvasható állományokat anélkül, hogy jelszóval igazolná magát. Ez különösen veszélyes az */etc/passwd* állomány esetén, és még veszélyesebb, ha nem használunk árnyékjelszavakat.

A lemez nélküli ügyfelek és X terminálok gyakran *tftp* kiszolgálóval töltik le programjukat a rendszerindító kiszolgálóról. Ha ezért futtatjuk a *tftpd* daemont, korlátozzuk hatókörét azokra a könyvtárakra, amelyekből az ügyfelek az állományokat letöltik; a könyvtárak nevét a *tftpd* parancssorában adjuk meg. A példa második *tftp* sora ezt mutatja.

A tcpd hozzáférés-szabályozó szolgáltatása

Amikor egy számítógépet elérhetővé teszünk a hálózaton keresztül, mindig számos biztonsági kockázatot teremtünk. Ezért az alkalmazásokat úgy tervezik, hogy védelmet nyújtsanak bizonyos típusú támadásokkal szemben. A biztonsági szolgáltatások azonban esetenként hibásak (ahogy azt az RTM internetes féreg a messzemenőkig bizonyította - a féreg számos program, többek között a régebbi *sendmail* daemonverziók hibáját használta ki), vagy egyszerűen nem képesek megkülönböztetni a biztonságos gazdagépeket (amelyektől elfogadhatunk kérést) és a nem biztonságos gazdagépeket (amelyek kérését szeretnénk elutasítani). A *finger* és *tftp* szolgáltatásokról röviden már szóltunk. Természetesen a rendszergazda szeretné a „megbízható" gazdagépekre korlátozni a szolgáltatásokhoz hozzáférők körét - ami rendes körülmények között lehetetlen, mert az *inetd* vagy az összes ügyfélnek, vagy egyiknek sem biztosítja a szolgáltatásokat.

A szolgáltatásokhoz való hozzáférés gazdagépek szerinti kezeléséhez hasznos eszköz a *tcpd*, melyet gyakran „daemonburkolónak" (wrapper) is neveznek. A megfigyelni vagy védeni kívánt TCP szolgáltatások esetén ezt az eszközt hívjuk meg a kiszolgálóprogram helyett. A *tcpd* ellenőrzi, hogy a távoli gazdagép használhatja-e a szolgáltatást, és csak akkor indítja el a valódi kiszolgálóprogramot, ha a válasz igenlő. A *tcpd* emellett a kéréseket a syslog daemonban naplózza. Jegyezzük meg, hogy az UDP alapú szolgáltatásokkal nem működik.

A *finger* daemon burkolásához például megváltoztatjuk az *inetd.conf* megfelelő sorát a

```
# nem burkolt finger daemon
finger    stream tcp nowait bin    /usr/sbin/fingerd in.fingerd
```

sorról a következő sorra:

```
# burkolt finger daemon
finger  stream  tcp     nowait  root    /usr/sbin/tcpd   in.fingerd
```

Ha nem jelölünk meg hozzáférés-szabályozást, az ügyfél számára a sor a szokásos finger beállításnak tűnik, kivéve hogy minden kérést a syslog *auth* szolgáltatásába naplózunk.

A hozzáférések szabályozását két állomány, az */etc/hosts.allow* és az */etc/hosts.deny* valósítja meg. Ezek olyan bejegyzéseket tartalmaznak, amelyek engedélyezik vagy tiltják a hozzáférést bizonyos szolgáltatásokhoz és gazdagépekhez. Amikor a *biff.foobar.com* ügyféltől kérés érkezik egy szolgáltatás, például a *finger* használatára, a *tcpd* átvizsgálja a *hosts.allow* és *hosts.deny* állományokat (ebben a sorrendben), és olyan bejegyzést keres, amely a szolgáltatásra és az ügyfélre is illik. Ha talál illeszkedő bejegyzést a *hosts.allow* állományban, engedélyezi a hozzáférést, és nem vizsgálja meg a *hosts.deny* állományt. Ha a *hosts.allow* állomány nem tartalmaz illeszkedő bejegyzést, de a *hosts.deny* igen, a kapcsolatot lezárva elutasítja a kérést. A kérést akkor is elfogadja, ha egyik állományban sem talál egyezést.

A hozzáférési állományok bejegyzései így épülnek fel:

```
szolgáltatáslista: gazdagéplista [:héjparancs]
```

A `szolgáltatáslista` az */etc/services* állományban megadott szolgáltatások nevét tartalmazó lista vagy az `ALL` kulcsszó. Ha a *finger* és a *tftp* kivételével valamennyi szolgáltatást meg szeretnénk adni, helyezzük el az `ALL EXCEPT finger, tftp` kifejezést.

A `gazdagéplista` egy gazdagépneveket és IP címeket, vagy az ALL, LOCAL, UNKNOWN, illetve a PARANOID kulcsszavakat tartalmazó lista. Az ALL valamennyi gazdagépnek, míg a LOCAL az olyan gazdagépneveknek felel meg, amelyek nem tartalmaznak pontot.* Az UNKNOWN azokra a gazdagépekre utal, amelyek névkikeresése sikertelen. A PARANOID azokra a gazdagépekre illeszkedik, amelyek neve nem felel meg IP címüknek.** Egy ponttal kezdődő név az összes gazdagépnek megfelel, amelynek tartománya az adott név. Például a **.foobar.com** illeszkedik a **biff.foobar.com** gépre, de nem felel meg a **nurks.fredsville.com** gépnek. A pontra végződő minta minden gazdagépre illeszkedik, amelynek IP címe a megadott mintával kezdődik, vagyis a **172.16.** megfelel a **172.16.32.0** gépnek, de nem felel meg a **172.15.9.1** gazdagépnek. És végül, a „/" karakterrel kezdődő bármely mintával egy olyan állományt adhatunk meg, amely a gazdagépek vagy IP címek elfogadott illesztési mintáit tárolja. A */var/access/trustedhosts* minta arra utasítja a *tcpd* daemont, hogy olvassa be az állományt és ellenőrizze, hogy annak valamely sora ráillik-e a kapcsolódni kívánó gazdagépre.

Ha a helyi gazdagépeket kivéve minden más gazdagépnek szeretnénk megtiltani a hozzáférést a *finger* és *tftp* szolgáltatásokhoz, helyezzük a következő sort az */etc/hosts.deny* állományba, és hagyjuk az */etc/hosts.allow* állományt üresen.

```
in.tftpd, in.fingerd: ALL EXCEPT LOCAL, .saját.tartományunk
```

Az opcionális *héjparancs* mező egy héjparancsot tartalmazhat, amelyet akkor hívunk meg, ha a bejegyzés illeszkedett. Ezzel a potenciális támadókat leleplező csapdákat állíthatunk fel. A következő példa olyan naplóállományt hoz létre, amely rögzíti a kapcsolódó felhasználót és gazdagépet, és ha a gazdagép nem a **vlager.vbrew.com**, akkor a felhasználó adatait a gazdagéphez fűzi:

```
in.ftpd: ALL EXCEPT LOCAL, .vbrew.com : \
      echo "request from %d@%h: >> /var/log/finger.log; \
      if [ %h != "vlager.vbrew.com:" ]; then \
          finger -l @%h >> /var/log/finger.log \
      fi
```

A `%h` és `%d` argumentumokat a *tcpd* az ügyfél gazdagép nevévé, illetve a szolgáltatás nevévé terjeszti ki. A részleteket megtaláljuk a *hosts_access(5)* súgóoldalon.

Egy másik megoldás: a xinetd

A *xinetd* a szabványos *inetd* alternatívájaként bukkant fel, és ma már széles körben elérhető. A program biztonságosabb és megbízhatóbb, és az *inetd* támadására alkalma-

* Rendszerint csak az */etc/hosts* kikeresésekből származó helyi gazdagépnevek nem tartalmaznak pontot.
** Noha a neve alapján azt gondolhatnánk, hogy csak szélsőséges esetekben alkalmazzuk, a `PARANOID` kulcsszó alapértelmezett beállításnak is megfelelhet, mert védelmet jelent a magukat másnak kiadó rosszindulatú gazdagépekkel szemben. A `PARANOID` támogatását nem minden `tcpd` daemonba fordítják bele; ha a miénkből is hiányzik, újra kell fordítanunk a daemont.

zott egyes DoS támadások ellen is védelmet nyújt. Vonzerejét növeli a számos beállítási lehetősége. Íme egy kis ízelítő:

- Teljes tulajdonságkészletű hozzáférés-szabályozás és naplózás.
- Az egyidejűleg futó szolgáltatások számának korlátozása.
- A `-services` egyedi szolgáltatáskötés, amellyel a szolgáltatásokat adott IP címekhez rendelhetjük.

A *xinetd* ma már a legtöbb Linux terjesztés szabványos része, de ha szeretnénk beszerezni a legfrissebb forráskódot vagy dokumentációt, a daemont megtaláljuk a *http://www.xinetd.org* címen. Ha a programot magunk fordítjuk és az IPv6 szabványt használjuk, ne feledjük el megadni a `--with-inet6` opciót!

A *xinetd* beállítása némileg eltérő, de nem bonyolultabb, mint az *inetd* daemoné. A *xinetd* nem követeli meg, hogy valamennyi szolgáltatáshoz egyetlen mester konfigurációs állományt alkalmazzunk. Beállíthatjuk úgy is, hogy használjon egy mester konfigurációs állományt – ez az */etc/xinetd.conf* –, valamint minden további beállított szolgáltatáshoz egy önálló konfigurációs állományt. Az eljárás, amellett hogy leegyszerűsíti a beállítást, az egyes szolgáltatások finomabb hangolását is lehetővé teszi – ez a *xinetd* rugalmasságának titka.

Először az */etc/xinetd.conf* állományt készítjük el. Lássunk egy mintaállományt!

```
# A xinetd minta konfigurációs állománya

defaults
{
        only_from       = localhost
        instances       = 60
        log_type        = SYSLOG authpriv info
        log_on_success  = HOST PID
        log_on_failure  = HOST
        cps             = 25 30
}

includedir /etc/xinetd.d
```

Noha több beállítási lehetőség is létezik, most csak az itt megadottakkal foglalkozunk. Lássuk, mit jelentenek:

`only_from`
: Azok az IP címek és gazdagépnevek, amelyek részére engedélyezzük a kapcsolatokat. A példában a kapcsolatokat a visszacsatoló interfészre korlátoztuk.

`instances`
: A *xinetd* összesen legfeljebb ennyi kiszolgálót futtat. Az opciónak ésszerű értéket adva megakadályozhatjuk, hogy a rosszindulatú felhasználók sikeres DoS támadást intézzenek gépünk ellen.

`log_type SYSLOG|FILE`
A tervezett naplózás típusát jelölhetjük ki. Két beállítása lehetséges, a `syslog` és a `file`. A `syslog` az összes naplóinformációt a rendszernaplóhoz küldi. A `file` direktíva a naplókat a megadott állományban hozza létre. A két opció további beállítási lehetőségeiről a *xinetd.conf* súgóoldalán tájékozódhatunk.

`log_on_success`
Azt állíthatjuk be, milyen információk kerülnek a naplókba, ha egy felhasználó sikeresen kapcsolódik. Íme néhány lehetőség:

HOST
A távoli gazdagép IP címét jegyzi fel.

PID
Az új kiszolgáló folyamatazonosítóját jegyzi fel.

DURATION
A munkafolyamat teljes időtartamát naplózza.

TRAFFIC
A hálózat kihasználtságát vizsgáló adminisztrátor számára lehet hasznos. A kimenő és bejövő bájtok végösszegét jegyzi fel.

`log_on_failure`

HOST
A távoli gazdagép IP címét jegyzi fel.

ATTEMPT
A szolgáltatások elérésére tett sikertelen kísérleteket naplózza.

`cps`
Újabb biztonsági beállítás. Segítségével megadhatjuk, hogy a szolgáltatáshoz milyen ütemben érkezhetnek a bejövő kapcsolatok. Két argumentumot vár: az első a másodpercenként megengedett kapcsolatok száma, a második a szolgáltatás kikapcsolt állapotának ideje.

A felsorolt opciók mindegyikét felülírhatjuk az egyes szolgáltatásokhoz tartozó konfigurációs állományokban, amelyeket az */etc/xinetd.d* könyvtárban helyezünk el. A mester konfigurációs állományban megadott értékek az alapértelmezett értékek. Az egyes szolgáltatások beállítása hasonlóan egyszerű. Lássunk egy példát az FTP szolgáltatás beállítására a *xinetd* esetén:

```
service ftp
{
        socket_type     = stream
```

```
        wait            = no
        user            = root
        server          = /usr/sbin/vsftpd
        server_args     = /etc/vsftpd/vsftpd.conf
        log_on_success  += DURATION USERID
        log_on_failure  += USERID
        nice            = 10
        disable         = no
}
```

A *xinetd.d* könyvtárban az egyes szolgáltatások elnevezése általában célzatos, mert így az önálló konfigurációs állományok könnyebben azonosíthatók és kezelhetők. Esetünkben az állomány egyszerűen *vsftp*, és az FTP kiszolgáló nevére utal.

A példa első sora a beállítandó szolgáltatás nevét definiálja. Meglepő módon a szolgáltatás típusát a *service* direktíva nem jelöli. A további beállítások kapcsos zárójelben állnak, hasonlóan a C függvényekhez. A szolgáltatás beállítási lehetőségei részben átfedik az alapértelmezett opciókat tartalmazó állomány beállításait. Amikor egy elemet megadunk az alapértelmezések között, majd ismét definiáljuk a szolgáltatáshoz tartozó önálló konfigurációban, ez utóbbi élvez elsőbbséget. Számos beállítási lehetőség létezik, ezekről a *xinetd.conf* súgóállományában részletes ismertetést kaphatunk. Ahhoz azonban, hogy egy szolgáltatást az alapvető beállításokkal elindíthassunk, elegendő csupán egy pár:

`socket_type`
A szolgáltatás által használt foglalat típusa. Aki már találkozott az *inetd* programmal, ismerősként üdvözölheti a rendelkezésre álló opciókat: *stream, dgram, raw* és *seqpacket.*

`wait`
Az opció megadja, hogy a szolgáltatás egy- vagy kétszálú. A `yes` jelenti az egyszálú szolgáltatást, ezért amikor a *xinetd* elindítja azt, addig nem foglalkozik az új kapcsolatokra vonatkozó kérésekkel, amíg az aktuális munkafolyamat véget nem ér. A `no` hatására a *xinetd* feldolgozza az új munkafolyamat-kéréseket.

`user`
A szolgáltatást futtató felhasználó neve.

`server`
A futtatott szolgáltatás helye.

`server_args`
A kiszolgálónak átadni kívánt kiegészítő opciók.

`nice`
A kiszolgáló prioritását szabályozza. Az opció ismét csak lehetőséget ad arra, hogy a kiszolgálók erőforrásait korlátozzuk.

`disable`
Megadja, hogy a szolgáltatás ki- vagy bekapcsolt állapotban van-e.

A services és protocols állományok

A portok számát, amelyeken bizonyos „szabványos" szolgáltatásokat elérhetünk, az Assigned Numbers RFC írja elő. Azért, hogy a kiszolgáló- és ügyfélprogramok számokká alakíthassák a szolgáltatások nevét, a listának legalább egy része minden gazdagépen megtalálható az */etc/services* állományban. A bejegyzések így festenek:

```
szolgáltatás port/protokoll [aliasnevek]
```

A *szolgáltatás* a szolgáltatás neve, a *port* az a port, amelyen a szolgáltatást kínáljuk, és a *protokoll* a használt átviteli protokoll. Általában ez utóbbi vagy az *udp*, vagy a *tcp*. Egy szolgáltatást több protokollal is kínálhatunk, és azonos porton is nyújthatunk különböző szolgáltatásokat, feltéve hogy a protokollok különböznek. Az *aliasnevek* mezőben a szolgáltatáshoz különféle hivatkozási neveket rendelhetünk.

Linux rendszereken általában nincs szükség arra, hogy a hálózati szoftverrel kapott *services* állományt megváltoztassuk. Mindazonáltal a 10.2. példában bemutatjuk az állomány egy részletét.

10.2. ábra. Minta /etc/services állomány

```
# /etc/services

tcpmux          1/tcp                   # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp       sink null
discard         9/udp       sink null
systat         11/tcp       users
daytime        13/tcp
daytime        13/udp
netstat        15/tcp
qotd           17/tcp       quote
msp            18/tcp                   # message send protocol
msp            18/udp                   # message send protocol
chargen        19/tcp       ttytst source
chargen        19/udp       ttytst source
ftp-data       20/tcp
ftp            21/tcp
fsp            21/udp       fspd
ssh            22/tcp                   # SSH Remote Login Protocol
ssh            22/udp                   # SSH Remote Login Protocol
telnet         23/tcp
# 24 - private
smtp           25/tcp       mail
# 26 - unassigned
```

A *services* állományhoz hasonlóan a hálózatkezelő könyvtárhoz is szükségünk van egy eljárásra, amellyel a protokollneveket – például a *services* állományban használtakat – a más gazdagépek IP rétege számára értelmezhető protokollszámokra fordíthatjuk. Ehhez a neveket az */etc/protocols* állományból keressük ki. Az állomány minden sora egyetlen bejegyzésből áll, amely tartalmazza a protokoll nevét és a hozzá tartozó számot. Annak az esélye, hogy ezt az állományt valaha is módosítanunk kell, még kisebb, mint az */etc/services* esetében. A 10.3. példa egy mintaállományt szemléltet.

10.3. példa. Minta /etc/protocols állomány

```
#
# Internet (IP) protocols
#
ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
igmp    2       IGMP            # internet group multicast protocol
tcp     6       TCP             # transmission control protocol
udp     17      UDP             # user datagram protocol
raw     255     RAW             # RAW IP interface

esp     50      ESP             # Encap Security Payload for IPv6
ah      51      AH              # Authentication Header for IPv6
skip    57      SKIP            # SKIP
ipv6-icmp 58    IPv6-ICMP       # ICMP for IPv6
ipv6-nonxt 59   IPv6-NoNxt      # No Next Header for IPv6
ipv6-opts 60    IPv6-Opts       # Destination Options for IPv6
rspf    73      RSPF            # Radio Shortest Path First.
```

Távoli eljáráshívás

Az ügyfél/kiszolgáló alkalmazások általános mechanizmusát a *Remote Procedure Call* (RPC) csomag biztosítja. Az RPC a Sun Microsystems fejlesztése és eszközök, valamint könyvtárfüggvények gyűjteménye. Az RPC-re épülő egyik fontos alkalmazás az NFS.

Az RPC gyűjtemény voltaképpen kiszolgálóeljárásokat tartalmaz. Az eljárásokat az ügyfél úgy hívhatja meg, hogy RPC kérést küld a kiszolgálóhoz az eljárás paramétereivel. A kiszolgáló az ügyfél nevében elindítja a jelzett eljárást, és a visszatérési értéket – ha van ilyen – visszajuttatja az ügyfélhez. A gépfüggetlenség biztosításához az ügyfél és a kiszolgáló között kicserélt összes adatot a küldő *External Data Representation* (XDR) formátumba alakítja, ezt azután a fogadó fordítja a helyi gép saját formátumára. Az RPC az UDP és TCP foglalatok segítségével juttatja el az XDR formátumú adatokat a távoli gazdagépre. A Sun nagyvonalúan nyilvánossá tette az RPC-t, amelyet RFC specifikációk sora ír le.

Előfordul, hogy egy RPC alkalmazás tökéletesítésekor inkompatibilis változásokat vezetnek be az eljáráshívó interfészben. Természetesen ha csak egyszerűen kicserélnénk a kiszolgálót, az összes alkalmazás, amely a korábbi viselkedést várná, összeomlana. Ezért az RPC programok rendelkeznek egy verziószámmal – ez rendszerint 1-gyel kezdődik –, amely az RPC interfész minden egyes új verziójával növekszik. A kiszolgáló gyakran több

verziót is kínál egyidejűleg; az ügyfél a kérésében a verziószámmal jelzi, hogy a szolgáltatás mely implementációját kívánja használni.

Az RPC kiszolgálók és ügyfelek közötti kommunikáció egészen különleges. Az RPC kiszolgáló egy vagy több eljárásgyűjteményt kínál; ezek a *programok,* amelyeket egyedi módon azonosít a *programszám.* A szolgáltatások nevét és a hozzájuk tartozó programszámokat egy lista tartalmazza, általában az */etc/rpc* állományban. A 10.4. példa ennek egy részletét mutatja be.

10.4. példa. Minta /etc/rpc állomány

```
#
# /etc/rpc - különféle RPC alapú szolgáltatások
#
portmapper    100000  portmap sunrpc
rstatd        100001  rstat rstat_svc rup perfmeter
rusersd       100002  rusers
nfs           100003  nfsprog
ypserv        100004  ypprog
mountd        100005  mount showmount
ypbind        100007
walld         100008  rwall shutdown
yppasswdd     100009  yppasswd
bootparam     100026
ypupdated     100028  ypupdate
```

A TCP/IP hálózatokban az RPC szerzői azzal a problémával néztek szembe, hogyan képezhetik le a programszámokat általános hálózati szolgáltatásokra. Végül is a kiszolgálókat úgy tervezték meg, hogy TCP és UDP portot is biztosítsanak minden egyes program minden verziójához. Az RPC alkalmazások az adatküldéshez általában az UDP-t veszik igénybe, és csak akkor használják a TCP-t, ha az átvinni kívánt adat nem fér egyetlen UDP datagramba.

Természetesen az ügyfélprogramoknak tudniuk kell, hogy a programszámok mely portoknak felelnek meg. Erre a célra egy konfigurációs állomány túlságosan rugalmatlan volna: mivel az RPC alkalmazások nem használnak fenntartott portokat, semmi nem garantálja, hogy egy portot, amelyet eredetileg adatbázis-alkalmazásunkhoz szántunk, valamely más folyamat nem veszi használatba. Ezért az RPC alkalmazások kiválasztanak egyet az elérhető portok közül, és bejegyzik azt egy különleges programnál, a *portleképző daemonnál.* Azt is mondhatjuk, hogy a portleképző ügynöki feladatokat lát el a gépen futó RPC kiszolgálóknál. Amikor egy ügyfél egy adott programszámú szolgáltatásra szeretne kapcsolódni, mindenekelőtt a kiszolgáló gazdagépének portleképzőjénél érdeklődik, amely válaszul megadja a TCP és UDP portszámokat, amelyeken a szolgáltatás elérhető.

Az eljárás egyetlen ponton sérülékeny, éppúgy, mint az *inetd* daemon a szabványos Berkeley szolgáltatások esetén. A helyzet azonban most rosszabb, mert ha a portleképzővel történik valami, az RPC portokról meglévő összes információt elveszítjük. Ez rendszerint azt jelenti, hogy manuálisan kell újraindítanunk valamennyi RPC szolgáltatást, vagy pedig a teljes gépet újra kell indítanunk.

Linuxon a portleképző neve */sbin/portmap,* néha */usr/sbin/rpc.portmap.* Azon kívül, hogy a hálózati indító szkriptekből elindítjuk, a portleképző nem igényel más beállítást.

A távoli bejelentkezés és végrehajtás beállítása

Sokszor kapóra jön, ha egy parancsot egy távoli gazdagépen futtathatunk, és a parancs ki- vagy bemenetét a hálózati kapcsolatról olvashatjuk, illetve arra írhatjuk.

A távoli gazdagépeken a parancsokat hagyományosan az *rlogin, rsh* és *rcp* utasításokkal hajthatjuk végre. Ezekkel kapcsolatban az 1. fejezetben már említettük a biztonság kérdését, és az *ssh* használatát javasoltuk helyettük. Az *ssh* csomagban biztosított helyettesítő eszközök az *ssh* és az *scp*.

Az előbbi parancsok mindegyike létrehoz egy héjat a távoli gazdagépen, és lehetővé teszi, hogy ott a felhasználó parancsokat futtasson. Természetesen az ügyfélnek ehhez egy felhasználói fiókkal kell rendelkeznie a távoli gazdagépen. Ezért a parancsok hitelesítési eljárást alkalmaznak. Az *r* parancsok titkosítás nélkül cserélik ki a felhasználónevet és jelszót a két gazdagép között, vagyis a jelszavak könnyen elfoghatók. Az *ssh* parancskészlet biztonságosabb megoldást kínál. Az eljárás neve Nyilvános Kulcsú Titkosítás (Public Key Cryptography), és hitelesítést, valamint titkosítást nyújt a gazdagépek közötti kommunikációhoz, hogy se a jelszavakhoz, se a munkafolyamatban elküldött adatokhoz ne férhessenek hozzá illetéktelenek.

A hitelesítési eljárást az egyes felhasználók esetében tovább enyhíthetjük. Ha például gyakran jelentkezünk be LAN hálózatunk más gépeire, akkor valószínűleg szeretnénk elkerülni, hogy minden egyes alkalommal meg kelljen adnunk a jelszavunkat. Erre az *r* parancsokkal mindig volt lehetőség, de az *ssh* még ezt is egyszerűsíti. Mindazonáltal nem tartjuk jó ötletnek, mert azt jelenti, hogy ha az egyik gépen egy felhasználói fiókba bejut valaki, az illető minden más olyan fiókhoz is hozzáfér, amelyekhez a fiók gazdája jelszó nélkül jelentkezhetett be. A megoldás persze nagyon kényelmes, és ezért a biztonsági kockázat sokakat nem tántorít el.

Beszéljünk most arról, hogyan távolíthatjuk el az *r* parancsokat és hogyan állíthatjuk munkába az *ssh*-t.

Az r parancsok kikapcsolása

Mindenekelőtt távolítsuk el az *r* parancsokat, ha telepítve vannak. A régi *r* parancsok kikapcsolására a legjobb módszer, ha az */etc/inetd.conf* állományban megjegyzésbe tesszük (vagy eltávolítjuk) bejegyzéseiket. A bennünket érdeklő bejegyzések így néznek ki.

```
# A shell, login, exec és talk BSD protokollok.
shell    stream  tcp     nowait  root  /usr/sbin/tcpd /usr/sbin/in.rshd
login    stream  tcp     nowait  root  /usr/sbin/tcpd /usr/sbin/in.rlogind
exec     stream  tcp     nowait  root  /usr/sbin/tcpd /usr/sbin/in.rexecd
```

Megjegyzésbe úgy helyezhetjük őket, hogy a sorok elejére # karaktert írunk, de akár teljes egészében ki is törölhetjük őket. Ne feledjük újraindítani az *inetd* daemont, hogy a módosítások életbe lépjenek. Ideális esetben a daemonokat is eltávolítjuk.

Az ssh telepítése és beállítása

Az *ssh* programkészlet ingyenes változata az OpenSSH. A Linux port az *ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable* címen érhető el, de megtaláljuk a legtöbb korszerű Linux terjesztésben is.* A program fordítását most nem tárgyaljuk, mert a forrásban kiváló útmutatást találunk hozzá. Ha lehetőségünk van előre fordított csomagból telepíteni a programot, érdemes ezt az utat követnünk.

Az *ssh* munkafolyamatnak két résztvevője van: az *ssh* ügyfél, amelyet beállítása után a helyi gazdagépen futtatunk, és az ssh daemon, amely a távoli gazdagépen fut.

Az ssh daemon

Az *sshd* daemon olyan program, amely az *ssh* ügyfelek hálózati kapcsolatait figyeli, kezeli a hitelesítést és végrehajtja a kért parancsokat. Rendelkezik egy fő konfigurációs állománnyal, amelynek neve */etc/ssh/sshd.conf,* és egy különleges állománnyal, amely a hitelesítéshez és titkosításhoz használt, a gazdagépet reprezentáló kulcsot tartalmazza. Minden gazdagépnek és ügyfélnek saját kulcsa van.

A kulcsot az *ssh-keygen* segédprogram állítja elő véletlenszerűen. A programot általában egyszer, a telepítéskor futtatjuk a gazdagép kulcsának előállítására. A kulcsot a rendszergazda többnyire az */etc/ssh/ssh_host_key* állományban tárolja. A kulcsok hossza 512 bit vagy annak többszöröse lehet. Alapértelmezés szerint az *ssh-keygen* 1024 bit hosszúságú kulcsokat állít elő, és a legtöbben ezt az alapértelmezett hosszt választják. Ha az OpenSSH SSH 2-es verzióját alkalmazzuk, RSA és DSA kulcsokat is készítenünk kell. A kulcsok előállításához az *ssh-keygen* parancsot így hívhatjuk meg:

```
# ssh-keygen -t rsa1 -f /etc/openssh/ssh_host_key -N ""
# ssh-keygen -t dsa -f /etc/openssh/ssh_host_dsa_key -N ""
# ssh-keygen -t rsa -f /etc/openssh/ssh_host_rsa_key -N ""
```

Ha elhagyjuk a `-N` opciót, a program egy jelszó megadását kéri. A gazdagépkulcsoknak azonban nem szabad jelszót használniuk, ezért az Enter billentyűt lenyomva hagyjuk üresen a mezőt. A program kimenete ehhez hasonló:

```
Generating public/private dsa key pair.
Your identification has been saved in sshkey.
Your public key has been saved in sshkey.pub.
The key fingerprint is:
fb:bf:d1:53:08:7a:29:6f:fb:45:96:63:7a:6e:04:22 tb@eskimo 1024
```

Vegyük észre, hogy három különböző kulcsot hoztunk létre. Az első, az rsa1 típusú az SSH protokoll 1-es verziójához, a másik két típus pedig, az rsa és dsa típusú az SSH protokoll 2-es verziójához használható. Az SSH protokoll 1-es verziója helyett érdemes inkább az SSH protokoll 2-es verzióját választanunk, a potenciális „középen a támadó" (man-in-the-middle) és egyéb támadások miatt, amelyekkel az 1-es verzió támadható.

Végül azt találjuk, hogy az egyes kulcsokhoz két állomány készült. Az első neve saját kulcs (private key), amelyet titokban tartunk és az */etc/openssh/ssh_host_key* állomány-

* Az OpenSSH-t az OpenBSD projekt fejlesztette, és nagyszerű példája a szabad szoftver előnyeinek.

ban helyezkedik el. A második neve nyilvános kulcs (public key), amelyet megoszthatunk. Helye az */etc/openssh/ssh_host_key.pub*.

Miután felfegyverkeztünk az *ssh* kommunikációhoz szükséges kulcsokkal, létrehozzuk a konfigurációs állományt. Az *ssh* készlet rendkívül hatékony, és a konfigurációs állományban számos opciót megadhatunk.

Az induláshoz egyszerű példát mutatunk be. A további opciókkal kapcsolatban az *ssh* dokumentációjában tájékozódhatunk. A következő kód egy biztonságos, minimális *sshd* konfigurációs állomány. A további beállítási opciók részletes leírása megtalálható az *sshd(8)* súgóoldalon:

```
#       $OpenBSD: sshd_config,v 1.59 2002/09/25 11:17:16 markus Exp $

#Port 22
Protocol 2
#ListenAddress 0.0.0.0
#ListenAddress ::

# Gazdagépkulcsok (HostKey) a protokoll 2-es verziójához
HostKey /etc/openssh/ssh_host_rsa_key
HostKey /etc/openssh/ssh_host_dsa_key

# Az 1-es verzió rövid lejáratú kiszolgálókulcsának élettartama és mérete
#KeyRegenerationInterval 3600
#ServerKeyBits 768

# Hitelesítés:

#LoginGraceTime 120
#PermitRootLogin yes
#StrictModes yes

#RSAAuthentication yes
#PubkeyAuthentication yes
# Állítsuk a yes értékre, ha nem bízzuk az ~/.ssh/known_hosts gazdagépeket
# RhostsRSAAuthentication és HostbasedAuthentication hitelesítésre
#IgnoreUserKnownHosts no

# A csatornán átvitt titkosítatlan jelszavak
# kikapcsolásához adjuk meg a no értéket!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Az s/key jelszavak kikapcsolásához adjuk meg a no értéket
#ChallengeResponseAuthentication yes

#X11Forwarding no
#X11DisplayOffset 10
#X11UseLocalhost yes
#PrintMotd yes
#PrintLastLog yes
#KeepAlive yes
#UseLogin no
#UsePrivilegeSeparation yes
```

```
#PermitUserEnvironment no
MaxStartups 10
# nincs alapértelmezett banner útvonal
#Banner /egy/utvonal
#VerifyReverseMapping no

# Az alrendszerek alapértelmezett beállítását nem írjuk felül
Subsystem       sftp    /usr/lib/misc/sftp-server
```

A rendszer biztonságának fenntartásához elengedhetetlen, hogy meggyőződjünk a konfigurációs állomány jogosultságainak megfelelő beállításáról. Adjuk ki a következő utasításokat:

```
# chown -R root:root /etc/ssh
# chmod 755 /etc/ssh
# chmod 600 /etc/ssh/ssh_host_rsa_key
# chmod 600 /etc/ssh/ssh_host_dsa_key
# chmod 644 /etc/ssh/sshd_config
```

Az utolsó lépés az *sshd* adminisztrációs daemon futtatása. Ehhez általában készítünk egy *rc* állományt, illetve egy már meglévőt használunk fel, hogy a rendszer indulásakor automatikusan futtathassuk a daemont. A daemon önállóan fut, ezért az */etc/inetd.conf* állományba nem szükséges bejegyezni. A daemont *root* felhasználóként futtathatjuk. A szintaxis nagyon egyszerű:

```
/usr/sbin/sshd
```

Futtatáskor az *sshd* automatikusan a háttérbe vonul, és készen állunk az *ssh* kapcsolatok fogadására.

Az ssh ügyfél

Több *ssh* ügyfél is létezik: az *slogin*, az *scp* és az *ssh*. Mindhárom ugyanazt a konfigurációs állományt használja, ez általában az */etc/openssh/ssh_config*. Emellett az őket futtató felhasználó saját könyvtárának *.ssh* alkönyvtárában lévő konfigurációs állományokat is beolvassák. Ezek közül a legfontosabb az *.ssh/config* állomány – az itt megadott opciók felülírják az */etc/openssh/ssh_config* állomány beállításait, ezenkívül az *.ssh/identity* állomány, amely a felhasználó saját kulcsát tartalmazza, valamint a megfelelő *.ssh/identity.pub* állomány a felhasználó nyilvános kulcsával. Jelentősek még az *.ssh/known_hosts* és az *.ssh/authorized_keys* állományok, amelyekről *Az ssh alkalmazása* című következő részben olvashatunk. Először azonban hozzuk létre a globális konfigurációs állományt és a felhasználói kulcs állományát!

Az */etc/ssh/ssh_config* nagyon hasonlít a kiszolgáló konfigurációs állományára. Ismét csak rengeteg tulajdonságot beállíthatunk, egy minimális konfiguráció azonban a 10.5. példához lehet hasonló. A további beállítási lehetőségeket az *sshd(8)* súgóoldal tartalmazza. Létrehozhatunk olyan szekciókat, amelyek megadott gazdagépekre vagy gazdagépek csoportjára illeszkedik. A „`Host`" utasítás paramétere vagy a gazdagép teljes neve, vagy egy joker karakterekkel megadott kifejezés; a példában mi is ilyet használtunk

az összes gazdagép illesztésére. Létrehozhatnánk például egy `Host *.vbrew.com` bejegyzést, amely a **vbrew.com** tartomány összes gazdagépéhez megfelel.

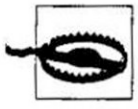
10.5. példa. Minta ssh ügyfél konfigurációs állomány

```
#       $OpenBSD: ssh_config,v 1.19 2003/08/13 08:46:31 markus Exp $


# Az egész hálózati helyre vonatkozó alapértelmezett beállítások

# Host *
#   ForwardAgent no
#   ForwardX11 no
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   BatchMode no
#   CheckHostIP yes
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   Port 22
#   Protocol 2,1
#   Cipher 3des
#   Ciphers aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes2
56-cbc
#   EscapeChar ~
```

A kiszolgálók beállításával kapcsolatban már említettük, hogy minden gazdagép és felhasználó rendelkezik egy kulccsal. A felhasználó kulcsát az *~/.ssh/identity* állomány tárolja. A kulcs elkészítéséhez a gazdagépkulcs létrehozására is használt *ssh-keygen* parancsot adhatjuk ki, ezúttal azonban nem szükséges megmondanunk, hogy melyik állományban szeretnénk a kulcsot tárolni. Az *ssh-keygen* alapértelmezése szerint megfelelő helyet választ, noha más útvonalat is meghatározhatunk. Esetenként érdemes több identity állományt készítenünk, ezt az *ssh* lehetővé teszi. Az *ssh-keygen* egy jelszót is kér. A jelszavak tovább erősítik a biztonságot, és alapvetően nagyon hasznosak. A jelszó nem jelenik meg a képernyőn, amikor beírjuk.



Ha elfelejtünk egy jelszót, nincs rá mód, hogy megtudjuk, mi volt az. Válasszunk olyan jelszót, amelyre biztosan emlékezni fogunk. Mint általában, a jelszó ne legyen túl nyilvánvaló, ne legyen például tulajdonnév, esetleg a saját nevünk. A valóban hatékony jelszavak hossza 10 és 30 karakter között változik, és nem kizárólag egyszerű szövegből állnak. Helyezzünk el benne szokatlan karaktereket. Ha elfelejtjük a jelszavunkat, új kulcsot kell készítenünk.

Kérjük meg felhasználóinkat, hogy futtassák le egyszer az *ssh-keygen* parancsot, így a kulcsállomány megfelelő módon elkészül. Az ssh-keygen a megfelelő jogosultságokkal létrehozza számukra az *~/.ssh/* könyvtárt, és az *.ssh/identity,* valamint az *.ssh/identity.pub* állományokban elkészíti a saját és a nyilvános kulcsukat. Lássunk egy példát a munkafolyamatra:

```
$ ssh-keygen
Key generation complete.
Enter file in which to save the key (/home/maggie/.ssh/identity):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/maggie/.ssh/identity.
Your public key has been saved in /home/maggie/.ssh/identity.pub.
The key fingerprint is:
1024 85:49:53:f4:8a:d6:d9:05:d0:1f:23:c4:d7:2a:11:67 maggie@moria
$
```

Az *ssh* most már készen áll a futtatásra.

Az ssh alkalmazása

Mostanra telepítettük az *ssh* parancsot és a hozzá tartozó programokat, így készek a futtatásra. Nézzük, hogyan futtathatjuk őket.

Elsőként megpróbálunk egy távoli bejelentkezést egy gazdagépre. Amikor egy gazdagéppel először kísérelünk meg kapcsolatot kialakítani, az *ssh* ügyfél beszerzi a gazdagép nyilvános kulcsát, majd felkér bennünket, hogy ellenőrizzük annak azonosságát – ehhez a nyilvános kulcs egy rövidített változatát, az úgynevezett *ujjlenyomatot* (fingerprint) mutatja be nekünk.

A távoli gazdagép rendszergazdájának már korábban át kellett adnia nekünk a nyilvános kulcshoz tartozó ujjlenyomatot, amelyet az *.ssh/known_hosts* állományban kell elhelyeznünk. Ha a távoli rendszergazda nem adta át nekünk a megfelelő kulcsot, rákapcsolódhatunk ugyan a távoli gazdagépre, de az *ssh* figyelmeztet, hogy nem rendelkezik a kulccsal, és megkérdezi, hogy elfogadjuk-e a távoli gazdagép által kínáltat. Ha biztosak vagyunk benne, hogy nem egy DNS álcázás áldozatai vagyunk éppen, és valóban a megfelelő gazdagéppel beszélünk, válaszoljunk igennel. A releváns kulcs ekkor automatikusan az *.ssh/known_hosts* állományba kerül, és a program többször nem kérdez rá. Ha egy jövőbeli kapcsolódási kísérletnél az adott gazdagépről beszerzett nyilvános kulcs nem egyezik meg a tárolt kulccsal, ismét figyelmeztetést kapunk, hiszen lehetséges, hogy biztonsági hiányosságról van szó.

A távoli gazdagépre történő első bejelentkezés így néz ki:

```
$ ssh vlager.vbrew.com
The authenticity of host `vlager.vbrew.com' can't be established.
Key fingerprint is 1024 7b:d4:a8:28:c5:19:52:53:3a:fe:8d:95:dd:14:93:f5.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'vchianti.vbrew.com,172.16.2.3' to the list of/
    known hosts.
maggie@vlager.vbrew.com's password:
Last login: Tue Feb  1 23:28:58 2004 from vstout.vbrew.com
$
```

Meg kell adnunk egy jelszót is; ez nem a helyi, hanem a távoli fiók jelszava. A jelszó a begépeléskor nem jelenik meg.

Argumentumok nélkül az *ssh* a helyi gépen használt felhasználói azonosítóval kísérli meg a bejelentkezést. Ezt a `-l` argumentummal bírálhatjuk felül, megadva a távoli gazdagéphez szükséges bejelentkezési nevet. Pontosan ezt tettük a könyv korábbi példájában. További megoldás, hogy a felhasználónév megadására a *felhasználóazonosító@gazdagépnév.kit* formátumot használjuk.

A helyi és a távoli gép között az *scp* programmal másolhatunk állományokat. Ennek szintaxisa hasonlít a hagyományos cp programéra, azzal a különbséggel, hogy az állománynév előtt megadhatjuk a gazdagép nevét is, ami az állomány útvonalát jelöli az adott gazdagépen (de használhatjuk a korábban említett *felhasználóazonosító@gazdagépnév* formátumot is). A következő példa az *scp* szintaxisát szemlélteti, és a */tmp/fred* helyi állományt a **vlager.vbrew.com** távoli gazdagép */home/maggie/* könyvtárába másolja:

```
$ scp /tmp/fred vlager.vbrew.com:/home/maggie/
maggie@vlager.vbrew.com's password:
fred                  100% |*****************************| 50165   00:01 ETA
```

Itt ismét meg kell adnunk egy jelszót. Az *scp* parancs alapértelmezés szerint hasznos üzenetekben tájékoztat bennünket a folyamatról. Hasonlóan egyszerűen másolhatunk állományokat a távoli gazdagépről is; csupán annak gazdagépnevét és az állomány útvonalát kell megadnunk forrásnak, valamint a helyi útvonalat rendeltetési célnak. Arra is lehetőségünk van, hogy egy állományt a távoli gazdagépről egy másik távoli gazdagépre másoljunk, noha ilyesmit általában nem teszünk, mert az összes adat a mi gazdagépünkön keresztül utazna.

A távoli gazdagépeken az *ssh* utasítással hajthatunk végre parancsokat. Az utasítás szintaxisa ismét csak egyszerű. Tegyük fel, hogy *maggie* felhasználónk szeretné megszerezni a **vlager.vbrew.com** távoli gazdagép gyökérkönyvtárát. Ezt így teheti meg:

```
$ ssh vchianti.vbrew.com ls -CF /
maggie@vchianti.vbrew.com's password:
bin/   ftp/         mnt/   sbin/                                    tmp/
boot/  home/        opt/   service/                                 usr/
dev/   lib/         proc/  stage3-pentium3-1.4-20030726.tar.bz2     var/
etc/   lost+found/  root/
```

Az *ssh* programot csővezetékre is kapcsolhatjuk és a programok be- és kimenetét rácsatolhatjuk, éppen úgy, mint minden más parancs esetén – azzal az eltéréssel, hogy a be- és kimenet az *ssh* kapcsolaton keresztül most a távoli gazdagépre irányítódik. A következő példa azt szemlélteti, hogyan használhatjuk ki ezt a lehetőséget a *tar* parancscsal egy teljes könyvtár és annak alkönyvtárai, valamint állományai átmásolására egy távoli gazdagépről a helyi gazdagépre:

```
$ ssh vchianti.vbrew.com "tar cf - /etc/" | tar xvf -
maggie@vchianti.vbrew.com's password:
etc/GNUstep
```

```
etc/Muttrc
etc/Net
etc/X11
etc/adduser.conf
..
..
```

A példában a végrehajtani kívánt parancsot idézőjelek közé helyeztük, így világosan látszik, hogy mit adunk át argumentumként az *ssh* programnak, és mit használ fel a helyi parancshéj. A parancs végrehajtja a *tar* utasítást a távoli gépen: összecsomagolja az */etc/* könyvtárt és a kimenetet a szabványos kimenetre írja. A kimenetet a helyi gazdagépen egy kicsomagoló üzemmódban futó és a szabványos bemenetet olvasó *tar* példányra csatoltuk.

A jelszót most is meg kellett adnunk. Állítsuk be most a helyi *ssh* ügyfelet úgy, hogy ne kérjen jelszót, amikor a **vchianti.vbrew.com** gazdagépre kapcsolódik. Korábban szóltunk már az *.ssh/authorized_keys* állományról: ilyen esetekben használjuk. Az *.ssh/authorized_keys* állomány azoknak a távoli felhasználói fiókoknak a *nyilvános* kulcsait tartalmazza, amelyekre automatikusan szeretnénk bejelentkezni. Az automatikus bejelentkezést úgy állíthatjuk be, hogy a *távoli* fiók *.ssh/identity/pub* tartalmát a helyi *.ssh/authorized_keys* állományba másoljuk. Rendkívül fontos, hogy az *.ssh/authorized_keys* jogosultságai csak nekünk tegyék lehetővé az állomány olvasását és írását, mert a kulcsokat ellopva bárki bejelentkezhet a távoli fiókba. A jogosultságok beállításához változtassuk meg az *.ssh/authorized_keys* állományt:

```
$ chmod 600 ~/.ssh/authorized_keys
```

A nyilvános kulcs *egyetlen* hosszú, egyszerű szöveges sor. Ha a helyi állományba a „másolás és beillesztés" segítségével másoljuk be a kulcsot, távolítsuk el az esetleg belekerült sorvég karaktereket. Az *.ssh/authorized_keys* állomány számos kulcsot tartalmazhat, amelyek mindegyike külön sorban helyezkedik el.

Az *ssh* eszközkészlet rendkívül hatékony, és még sok más hasznos tulajdonságát és lehetőségét felfedezhetjük. További információt a súgóoldalakon és a csomaggal érkező dokumentációban kaphatunk.

11

Az elektronikus levelezés adminisztrációs kérdései

Amióta hálózatok léteznek, az elektronikus levelezés mindig is az egyik legfontosabb felhasználási terület volt. Az elektronikus levél (e-mail) egyszerű szolgáltatásként indult, amely egy állományt az egyik gépről egy másikra másolt, a címzett *mailbox* állományához fűzve azt. Az elgondolás nem változott, azonban az egyre növekvő hálózatok az összetett útválasztási igények és az üzenetekből fakadó egyre nagyobb terhelés mellett kifinomultabb eljárást tettek szükségessé.

A levelek cseréjére különféle szabványokat állítottak fel. Az internet hálózati helyei az RFC 822 szabványt követik, amelyet egyéb RFC szabványok egészítenek ki, lehetővé téve, hogy szinte *bármit* elküldhessünk levélben, a grafikától kezdve a hangállományokon keresztül a különleges karakterkészletekig.* A CCITT eltérő szabványt definiált, ez az X.400, amelyet a mai napig használnak egyes nagyvállalati és kormányzati környezetekben, noha fokozatosan eltűnőben van.

A Unix rendszerekre meglehetősen nagy számban készültek levelező programok. Az egyik legismertebb közülük a *sendmail,* amelyet Eric Allman fejlesztett ki a berkeleyi California Egyetemen. Eric Allman a *sendmail* programot egy kereskedelmi vállalkozáson keresztül kínálja, a szoftver azonban továbbra is szabad szoftver. Egyes Linux terjesztések a *sendmail* programot tartalmazzák szabványos levélátviteli közvetítőként. A *sendmail* beállításával a 12. fejezetben foglalkozunk.

A *sendmail* több olyan konfigurációs állományt támogat, amelyeket rendszerünk igényeihez kell igazítanunk. A levelező alrendszer futtatásához szükséges információktól eltekintve is (ilyen például a helyi gazdagép neve) számos paramétert beállíthatunk. A *sendmail* fő konfigurációs állományát első ránézésre nagyon nehéz áttekinteni. Látszólag olyan, mintha valaki lenyomott Shift billentyű mellett a billentyűzeten aludt volna. Szerencsére a modern konfigurációs eljárások sok fejfájástól megkímélnek bennünket.

Amikor a felhasználók személyes rendszerükre leveleket töltenek le, önálló protokollra van szükségük ahhoz, hogy a levelezőkiszolgálóra kapcsolódjanak. A 15. fejezetben egy hatékony és egyre népszerűbb kiszolgálótípussal ismerkedünk meg, melynek neve IMAP.

Ebben a fejezetben azt vizsgáljuk, mi is az az elektronikus levél, és milyen teendőik lehetnek a rendszergazdáknak a levelezéssel kapcsolatban. A 12. fejezetben a *sendmail* első beállításához kapunk segítséget. A leírtak alapján képesek leszünk kisebb hálózati

* Aki nem hiszi, olvassa el az RFC 1437 leírást!

helyeket üzembe állítani, noha további lehetőségek is léteznek, és sok boldog órát tölthetünk még el a számítógép előtt a legkülönfélébb tulajdonságok beállításával.

A Linux rendszereken történő elektronikus levelezéssel kapcsolatban részletesebb útbaigazítást találhatunk Guylhem Aznar *Electronic Mail HOWTO* dokumentációjában. A *sendmail* forrás terjesztése szintén átfogó dokumentációt tartalmaz, amelyben a beállításokkal kapcsolatos legtöbb kérdésünkre választ kaphatunk.

Hogyan épül fel az elektronikus üzenet?

Egy elektronikus üzenet általában tartalmaz egy üzenettörzset, amely az üzenet szövege, valamint különleges adminisztratív adatokat, amelyek a címzettet, az átviteli médiumot stb. határozzák meg, valahogy úgy, mint egy valódi levélen látható címzés.

Az adminisztratív adatok két kategóriába sorolhatók. Az első kategóriába azok az adatok tartoznak, amelyek az átviteli médiumra vonatkoznak; ilyen például a küldő és a fogadó címe. Ezért ezt a kategóriát *borítéknak* nevezik. A borítékot az átviteli szoftver módosíthatja, miközben az üzenetet továbbítja.

A második kategóriába azok az adatok tartoznak, amelyek a levélüzenet kezeléséhez szükségesek, és nem kapcsolódnak egyetlen adott átviteli mechanizmushoz sem; ilyen például az üzenet témasora (subject), a címzettek listája és az üzenet elküldésének dátuma. Sok hálózatban szabvánnyá vált, hogy ezeket az adatokat a levélüzenet elé helyezik, létrehozva ezzel a *levél fejlécét.* A fejlécet a *levéltörzstől* egy üres sor választja el.* A Unix világában a legtöbb levélátviteli szoftver az RFC 822 szabványban leírt fejlécformátumot alkalmazza. A szabvány eredetileg az ARPANET-hez definiálta a szabványt, de mivel úgy tervezték, hogy mindenféle környezettől független legyen, könnyen át lehetett alakítani más hálózatokhoz, többek között számos UUCP alapú hálózathoz.

Az RFC 822 azonban csak a legkisebb közös nevező. Újabb szabványokat is összeállítottak az egyre növekvő igények kezelésére, például a titkosítás, a nemzetközi karakterkészlet és a Multipurpose Internet Mail Extensions (MIME) támogatására, amelyeket az RFC 1341 és más RFC specifikációk írnak le.

Mindezen szabványokban a fejléc több sort tartalmaz, amelyeket sorvéget jelentő karaktersorozatok választanak el egymástól. Az egyes sorok tartalmaznak egy mezőnevet, amely az első oszlopban kezdődik, és magát a mezőt, melyet egy kettőspont és egy whitespace karakter választ el. Az egyes mezők formátuma és szemantikája a mező nevétől függően változik. A fejléc mezői több sort is elfoglalhatnak, ha a következő sorok whitespace karakterrel kezdődnek, például tabulátorral. A mezők sorrendje tetszőleges.

Tipikus levélfejlécre láthatunk itt példát:

```
Return-Path: <root@oreilly.com>
X-Original-To: spam@xtivix.com
Delivered-To: spam@ xtivix.com
Received: from smtp2.oreilly.com (smtp2.oreilly.com [209.58.173.10])
    by www.berkeleywireless.net (Postfix) with ESMTP id B05C520DF0A
```

* A levélüzenethez szokás egy *aláírást* (*.sig*) fűzni, amely rendszerint a szerzőről tárol információt. Az aláírás és a levélüzenet között egy sor áll, amelynek tartalma a „— ", és helyköz van a végén. A netikett megköveteli a tömörséget.

```
    for <spam@ xtivix.com>; Wed, 16 Jul 2003 06:08:44 -0700 (PDT)
Received: (from root@localhost)
    by smtp2.oreilly.com (8.11.2/8.11.2) id h6GD5f920140;
    Wed, 16 Jul 2003 09:05:41 -0400 (EDT)
Date: Wed, 16 Jul 2003 09:05:41 -0400 (EDT)
Message-Id: <200307161305.h6GD5f920140@smtp2.oreilly.com>
From: Andy Oram <root@oreilly.com>
To: spam@ xtivix.com
Subject: Article on IPv6
```

Rendszerint a szükséges fejlécmezőket a használt levélolvasó program generálja; ilyen program például az *elm*, az *Outlook*, az *Evolution* vagy a *pine*. Más mezők opcionálisak, és a felhasználó hozza létre azokat. Az *elm* például lehetővé teszi, hogy az üzenet fejlécének egy részét szerkeszthessük. Más mezőket a levélátviteli szoftver hoz létre. Ha belepillantunk egy helyi *mailbox* állományba, az egyes levélüzenetek előtt a „From" szót láthatjuk (figyelem: kettőspont nélkül). Ez nem RFC 822 fejléc; a használt levelező szoftver illesztette be, hogy a levelesládát olvasó programok dolgát megkönnyítse. Azért, hogy elkerülhessük a potenciális hibákat az olyan sorok esetén, amelyek szintén a „From" kifejezéssel kezdődnek, szabványos eljárássá vált, hogy a levélüzenet törzsében előforduló azonos sorozatokat kiváltjuk (escape), egy > karaktert helyezve eléjük.

A következő lista a gyakori fejlécmezőket és jelentésüket foglalja össze:

`From:`
: A küldő e-mail címét és esetleg a „valódi nevét" tartalmazza. Számos különböző formátum fordul elő, mivel szinte minden levelező másképpen valósítja meg.

`To:`
: A rendeltetési e-mail címek listáját tartalmazza. Ha több cím is szerepel, azokat vessző választja el.

`Cc:`
: Azon e-mail címek listája, amelyek „indigómásolatot" (carbon copy) kapnak az üzenetről. Ha több cím is szerepel, azokat vessző választja el.

`Bcc:`
: Azon e-mail címek titkos listája, amelyek „indigómásolatot" kapnak az üzenetről. A „Cc:" és a „Bcc:" közötti alapvető különbség, hogy a „Bcc:" mezőben megadott címek nem jelennek meg a fogadókhoz eljuttatott levélüzenetek fejlécében. Így másolatküldés címzettjeit megváltoztathatjuk anélkül, hogy a többiek értesülnének erről. Ha több cím is szerepel, azokat vessző választja el.

`Subject:`
: A levél tartalmának pár szavas leírása.

`Date:`
: A levél küldésének ideje és dátuma.

`Reply-To:`
Azt a címet határozza meg, amelyre a küldő a fogadó válaszát várja. Akkor lehet hasznos, ha több fiókunk is van, de a levelek zömét a leggyakrabban használt fiókba szeretnénk irányítani. A mező opcionális.

`Organization:`
A szervezet neve, amelynek gépéről a levél származik. Ha a számítógép személyes tulajdonunk, hagyjuk ki a mezőt, vagy helyezzük el benne a „private" kifejezést, illetve valamilyen teljesen értelmetlen szöveget. A mezőt egyetlen RFC szabványban sem találjuk meg, és teljesen opcionális. Egyes levelező programok közvetlenül támogatják, míg sok más program nem.

`Message-ID:`
A levelet küldő rendszer levélátvivője által előállított karakterlánc. Az üzenet egyedi azonosítására szolgál.

`Received:`
A levelet feldolgozó hálózati helyek (beleértve a küldőt és a fogadót is) mindegyike elhelyez ilyen mezőt a fejlécben, megadva a hálózati hely nevét, egy üzenetazonosítót (ID), az üzenet vételének idejét és dátumát, a hálózati helyet, amelyről az üzenet érkezett, és a használt átviteli szoftvert. A sorok alapján ellenőrizhetjük, milyen útvonalon haladt a levél, és ha hiba történt, az illetékeshez fordulhatunk.

`X-bármi:`
A levelek kezelését végző programok mindegyike elfogadja az `X-` kezdetű fejléceket. Segítségükkel olyan kiegészítő szolgáltatásokat valósíthatunk meg, amelyek még nem kerültek be egyetlen RFC szabványba sem, esetleg nem is fognak. Például létezett valamikor egy hatalmas linuxos levelezőlista-kiszolgáló, amelyen az `X-Mn-Key:` és egy csatorna megadásával kijelölhettük, mely csatornára szeretnénk a levelet küldeni.

Hogyan történik a levél kézbesítése?

A leveleket általában egy program, például a *mail*, a *mailx*, esetleg fejlettebb programok, például a *mutt*, a *tkrat* vagy a *pine* segítségével hozzuk létre. Ezeknek a programoknak az összefoglaló neve *levelezési felhasználói közvetítő* (mail user agent, MUA). Amikor egy levélüzenetet küldünk, az interfész program a legtöbb esetben egy további programra bízza a kézbesítést: a *levélátviteli közvetítőre* (mail transfer agent, MTA). A legtöbb rendszeren ugyanaz az MTA végzi a helyi és távoli kézbesítéseket, és rendszerint az */usr/sbin/sendmail*, illetve nem FSSTND kompatibilis rendszereken az */usr/lib/sendmail* hívásával érhető el.

Természetesen a helyi levélkézbesítés nem egyszerűen azt jelenti, hogy a bejövő üzenetet a címzett levelesládájához fűzzük. Rendszerint a helyi MTA végzi a hivatkozásicímkezelést (angolul aliasing; amikor más címekre mutató helyi fogadó címeket hozunk létre) és a továbbítást (amikor egy felhasználó levelezését más rendeltetési helyre irányít-

juk át). Mindemellett a nem kézbesíthető leveleket a megfelelő hibaüzenettel *visszairányítja* (bounce) a küldőhöz.

A távoli kézbesítésnél a használt átviteli szoftver a kapcsolat természetétől függ. A TCP/IP hálózatokon a levélkézbesítéshez általában a *Simple Mail Transfer Protocol-t* (SMTP) használjuk, melyet az RFC 821 ír le. Az SMTP-t arra tervezték, hogy a leveleket közvetlenül a címzett gépére továbbítsa, egyeztetve az üzenetátvitelt a távoli oldal SMTP daemonával. A szervezetek jelenleg gyakran állítanak fel különleges célú gazdagépeket, amelyek a szervezethez tartozók valamennyi levelét fogadják, majd eljuttatják a címzetthez.

Az e-mail címek

Az e-mail címek legalább két részből állnak. Az egyik rész a *levelezőtartomány* neve, amely végül a címzett gazdagépére, illetve olyan gazdagépre fordítható le, amely a címzett helyett fogadja a leveleket. A másik rész a felhasználó egyedi azonosítását teszi lehetővé; erre szolgál a felhasználó bejelentkezési neve, a felhasználó igazi neve „keresztnév.vezetéknév" formában, vagy egy tetszőleges alias név, amely egy vagy több felhasználóra mutat. A levelezés más címzési eljárásai, például az X.400 általánosabb „attribútumkészletet" alkalmaz, amelynek segítségével egy X.500 könyvtárkiszolgálóban a fogadó gazdagépe kikereshető.

Az e-mail címek értelmezése nagyrészt attól függ, hogy milyen típusú hálózatot használunk. Mi most a TCP/IP hálózatokban vizsgáljuk meg az e-mail cím értelmezésének módját.

Az RFC 822

Az internet hálózati helyei az RFC 822 szabványt követik, amely az ismerős *felhasználó@gazdagép.tartomány* jelölést írja elő, ahol a *gazdagép.tartomány* a gazdagép teljes minősített tartományneve. A két részt elválasztó karakter pontos angol megnevezése „commercial at", magyarul at („et") vagy kukac. Az ilyen módon készült jelölés azonban nem határozza meg a rendeltetési gazdagéphez vezető útvonalat, ami egy külön mechanizmus, a levélüzenet-irányítás feladata. Hamarosan szó lesz róla.

Elavult levélformátumok

Mielőtt tovább lépnénk, vessünk egy pillantást a múltra. Az eredeti UUCP környezetben az uralkodó forma az *útvonal!gazdagép!felhasználó* volt, ahol az *útvonal* azokat a gazdagépeket határozta meg, melyeken az üzenetnek át kellett haladnia a rendeltetési helyét jelentő *gazdagéphez.* A szerkezet neve *bang útvonal,* mert a felkiáltójelet angolul a hétköznapi beszédben gyakran „bang"-nek nevezik.

Más hálózatok további, eltérő címzési módot használtak. A DECnet alapú hálózatok például a cím felosztására két kettőspontot alkalmaztak, így a cím a *gazdagép::felhasználó* volt. Az X.400 szabvány teljesen eltérő eljárást használ, amely a fogadót attribútum–érték párokból álló készlettel írta le, például megye vagy szervezet alapján.

És végül, a FidoNet hálózaton az egyes felhasználókat egy kód azonosította, például a *2:320/204.9*. A kód négy számot tartalmazott, amelyek a zónát (a 2 Európát), a hálózatot (a 320 Párizst és Banlieue-t), a csomópontot (a helyi jelelosztót, hubot) és a pontot (a felhasználó személyi számítógépet) írták le. A Fidonet címeket az RFC 822-re képezték le; az iménti példa így nézett ki:*Thomas.Quinot@p9.f204.n320.z2.fidonet.org*. Azt hiszem, mindannyian boldogok lehetünk, hogy ma már egyszerűbb tartománynéveket használunk!

Hogyan jelöljük ki a levél útvonalát?

Azt a folyamatot, amikor egy üzenetet a címzett gazdagépére irányítunk, *útválasztásnak* (routing) hívjuk. Amellett hogy megfelelő útvonalat kell találnunk a küldő helytől a rendeltetési helyig, az útválasztás hibaellenőrzést és esetenként a sebesség és a költség optimalizálását is magában foglalja.

Az interneten az adatok irányítását a címzetthez (ha már ismerjük a címzett IP címét) az IP hálózatkezelő réteg végzi.

Levelezési útválasztás az interneten

Az interneten a rendeltetési gazdagép konfigurációja határozza meg, végbemegy-e bármilyen specifikus levelezési útválasztás. Alapértelmezés szerint az üzenet kézbesítése a rendeltetési helyre úgy történik, hogy először meghatározzuk, melyik gazdagépre kell az üzenetet küldenünk, majd közvetlenül arra a gazdagépre juttatjuk. A legtöbb internetes hálózati hely az összes bejövő levelet egy megfelelő teljesítményű levelezőkiszolgálóra irányítja, amely képes ezt a forgalmat kezelni; a leveleket ez a kiszolgáló osztja szét, helyben. A szolgáltatás bejelentéséhez a hálózati hely úgynevezett MX rekordot bocsát ki a DNS adatbázisában a helyi tartomány számára. Az MX a *Mail Exchanger,* a levelezőkiszolgáló rövidítése, és lényegében arra utal, hogy a kiszolgáló-gazdagép hajlandó levéltovábbítóként fellépni a tartomány minden levelezési címéhez. Az MX rekordok arra is alkalmasak, hogy olyan gazdagépek forgalmát kezeljék, amelyek maguk nem kapcsolódnak az internetre. Az ilyen gazdagépek levelei egy átjárón haladnak keresztül. Az elgondolásról bővebben a 6. fejezetben olvashatunk.

Az MX rekordokhoz mindig tartozik egy *preferencia,* amely egy pozitív egész szám. Ha egy gazdagépnek több levelezőkiszolgálója is van, a levélátviteli közvetítő a legalacsonyabb preferenciaértékkel rendelkező kiszolgálóhoz próbálja az üzenetet eljuttatni, és csak ha ez nem sikerül, akkor próbálkozik egy magasabb értékű gazdagéppel. Ha a helyi gazdagép maga is a rendeltetési cím egyik levelezőkiszolgálója, csak a sajátjánál kisebb preferenciájú MX gazdagépekhez továbbíthatja az üzeneteket; ily módon kiküszöbölhető, hogy a levelek körbe-körbe járjanak. Ha egy tartományhoz nem létezik MX rekord, illetve nem maradt megfelelő MX rekord, akkor a levélátviteli közvetítő már megnézheti, hogy a tartományhoz tartozik-e IP cím, és megpróbálhatja a kézbesítést közvetlenül a gazdagépre.

Tegyük fel, hogy egy szervezet, mondjuk a Kocka Kft. az összes levél kezelését a **levelkezelo** számítógépére szeretné bízni. Ekkor DNS adatbázisában elhelyez egy ehhez hasonló MX rekordot:

```
barna.kocka.com.        IN   MX      5    levelkezelo.kocka.com.
```

A rekord a **levelkezelo.kocka.com** gépet a **barna.kocka.com** tartomány levelezőkiszolgálójának jelöli ki, 5-ös preferencia értékkel. Amikor egy gazdagép a *joe@barna.kocka.com* címre szeretne egy üzenetet eljuttatni, ellenőrzi a DNS-t és megtalálja a **levelkezelo** gépre mutató MX rekordot. Ha nem létezik 5-ösnél kisebb preferenciájú MX, az üzenet a **levelkezelore** kerül, amely azután továbbítja a **barna** gépre.

Ez a rövid összefoglalás az MX rekordok működésének nagyon egyszerű bemutatása. Az internetes levelezési útválasztásról többet is megtudhatunk az RFC 821, RFC 974 és az RFC 1123 specifikációkból.

12

A sendmail

Az a mondás járja, hogy addig senkiből nem lehet *igazi* Unix rendszergazda, amíg nem készített egy *sendmail.cf* állományt. Azt is szokták mondani, hogy bolond, aki másodszor is megpróbálja.

Szerencsére többé nincs szükség arra, hogy közvetlenül szerkesszük a titokzatos *sendmail.cf* állományt. A *sendmail* újabb verziói biztosítanak egy konfigurációs eszközt, amely jóval egyszerűbb makróállományok alapján hozza létre helyettünk a *sendmail.cf* állományt. Nem szükséges értenünk az állomány összetett szintaxisát, mert a makró nyelvvel határozzuk meg, hogy milyen szolgáltatásokat szeretnénk a konfigurációban beállítani, és adjuk meg a szolgáltatások működését leíró paramétereket. Ezután az *m4* elnevezésű hagyományos Unix segédprogram – a tulajdonképpeni *sendmail.cf* szintaxist tartalmazó sablonállományok alapján – létrehozza a saját *sendmail.cf* állományunkat a makró konfigurációs adataiból.

A *sendmail* hihetetlenül hatékony levelező program, és elsajátítása igen nehéz. Olyan program, melynek az átfogó referenciája 1200 oldalt tesz ki, a legtöbb emberre riasztóan hat (a *sendmail* című referenciakönyvet Bryan Costales és Eric Allman írta, kiadója az O'Reilly; erről a témáról a Kossuth Kiadónál megjelent Richard Blum *sendmail Linuxra* című kötete). Azonkívül egy program teljes körű bemutatására, amely olyan összetett, mint a *sendmail,* nem elegendő egyetlen fejezet. Ebben a fejezetben bemutatjuk a *sendmail* programot, és a Virtuális Sörgyár példáján keresztül egyszerű konfigurációban ismertetjük a telepítését, beállítását és tesztelését. Ha a fejezetet elolvasva kevésbé találjuk majd ijesztőnek a *sendmail* beállítását, akkor reméljük, hogy elegendő önbizalmat adtunk, hogy saját, összetettebb konfigurációk létrehozásával is megpróbálkozzunk!

A sendmail disztribúció telepítése

A legtöbb Linux terjesztésben a *sendmail* megtalálható előre csomagolt formában. Ennek ellenére érdemes inkább forrásból telepítenünk, különösen ha fontos számunkra a biztonság. A *sendmail* gyakran változik, mert biztonsági hiányosságokat küszöbölnek ki benne és új tulajdonságokkal bővítik. A lezárt biztonsági rések és az új tulajdonságok miatt érdemes rendszerünk *sendmail* programját frissítenünk. Mindemellett ha forrásból fordítjuk, jobban beleszólhatunk a *sendmail* környezet kialakításába. Iratkozzunk fel a *sendmail-announce* levelezőlistára, így mindig idejében értesülünk az új *send-*

mail kibocsátásokról, és figyeljük a *http://www.sendmail.org/* oldalt, ahol tájékozódhatunk a potenciális biztonsági hiányosságokról és a legfrissebb *sendmail* fejlesztésekről.

A sendmail forráskódjának letöltése

A *sendmail* forráskódos terjesztése és a forráskódos terjesztés aláírás-állománya letölthető a *http://www.sendmail.org/current-release.html* címről, a tükröroldalak valamelyikéről vagy az *ftp://ftp.sendmail.org/pub/sendmail/* címről. Lássunk most példát az *ftp* használatára!

```
# ftp ftp.sendmail.org
Connected to ftp.sendmail.org (209.246.26.22).
220 services.sendmail.org FTP server (Version 6.00LS) ready.
Name (ftp.sendmail.org:craig): anonymous
331 Guest login ok, send your email address as password.
Password: win@vstout.com
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /pub/sendmail
250 CWD command successful.
ftp> get sendmail.8.12.11.tar.gz
local: sendmail.8.12.11.tar.gz remote: sendmail.8.12.11.tar.gz
227 Entering Passive Mode (209,246,26,22,244,234)
150 Opening BINARY mode data connection for 'sendmail.8.12.11.tar.gz'
(1899112 bytes).
226 Transfer complete.
1899112 bytes received in 5.7 secs (3.3e+02 Kbytes/sec)
ftp> get sendmail.8.12.11.tar.gz.sig
local: sendmail.8.12.11.tar.gz.sig remote: sendmail.8.12.11.tar.gz.sig
227 Entering Passive Mode (209,246,26,22,244,237)
150 Opening BINARY mode data connection for 'sendmail.8.12.11.tar.gz.sig'
(152 bytes).
226 Transfer complete.
152 bytes received in 0.000949 secs (1.6e+02 Kbytes/sec)
```

Ha az aktuális *sendmail* PGP kulcsok nincsenek a kulcskarikánkon, az aláírás ellenőrzéséhez töltsük le azokat. Az `ftp` kapcsolatot folytatva a következő lépéssel tölthetjük le az aktuális év kulcsait:

```
ftp> get PGPKEYS
local: PGPKEYS remote: PGPKEYS
227 Entering Passive Mode (209,246,26,22,244,238)
150 Opening BINARY mode data connection for 'PGPKEYS' (61916 bytes).
226 Transfer complete.
61916 bytes received in 0.338 secs (1.8e+02 Kbytes/sec)
ftp> quit
221 Goodbye.
```

Ha új PGP kulcsokat töltöttünk le, fűzzük rá a kulcskarikánkra. A következő példában a gpg-t (Gnu Privacy Guard) használjuk:

```
# gpg --import PGPKEYS
gpg: key 16F4CCE9: not changed
gpg: key 95F61771: public key imported
gpg: key 396F0789: not changed
gpg: key 678C0A03: not changed
gpg: key CC374F2D: not changed
gpg: key E35C5635: not changed
gpg: key A39BA655: not changed
gpg: key D432E19D: not changed
gpg: key 12D3461D: not changed
gpg: key BF7BA421: not changed
gpg: key A00E1563: non exportable signature (class 10) - skipped
gpg: key A00E1563: not changed
gpg: key 22327A01: not changed
gpg: Total number processed: 12
gpg:               imported: 1  (RSA: 1)
gpg:              unchanged: 11
```

A *PGPKEYS* állomány tizenkét exportálható kulcsából mindössze egy kerül a kulcskarikánkra. A *not changed* (nem változott) megjegyzés a maradék tizenegy kulcs esetében azt jelzi, hogy már a kulcskarikán voltak. Amikor először importáljuk a *PGPKEYS* állományt, mind a tizenkét kulcs felkerül a kulcskarikára.

Mielőtt az új kulcsot használnánk, ellenőrizzük az ujjlenyomatát, ahogy a következő gpg példában láthatjuk:

```
# gpg --fingerprint 95F61771
pub  1024R/95F61771 2003-12-10 Sendmail Signing Key/2004 <sendmail@Sendmail.ORG>
     Key fingerprint = 46 FE 81 99 48 75 30 B1  3E A9 79 43 BB 78 C1 D4
```

Hasonlítsuk össze a megjelenített ujjlenyomatot a 12.1. táblázattal, amely a *sendmail* aláírási kulcsokhoz tartozó ujjlenyomatokat tartalmazza.

12.1. táblázat. A sendmail aláírási kulcsok ujjlenyomatai

Év	*Ujjlenyomat*
1997	CA AE F2 94 3B 1D 41 3C 94 7B 72 5F AE 0B 6A 11
1998	F9 32 40 A1 3B 3A B6 DE B2 98 6A 70 AF 54 9D 26
1999	25 73 4C 8E 94 B1 E8 EA EA 9B A4 D6 00 51 C3 71
2000	81 8C 58 EA 7A 9D 7C 1B 09 78 AC 5E EB 99 08 5D
2001	59 AF DC 3E A2 7D 29 56 89 FA 25 70 90 0D 7E C1
2002	7B 02 F4 AA FC C0 22 DA 47 3E 2A 9A 9B 35 22 45
2003	C4 73 DF 4A 97 9C 27 A9 EE 4F B2 BD 55 B5 E0 0F
2004	46 FE 81 99 48 75 30 B1 3E A9 79 43 BB 78 C1 D4

Amennyiben az ujjlenyomat helyes, aláírhatjuk – és így érvényesíthetjük – a kulcsot. A következő gpg példában az újonnan importált *sendmail* kulcsot írjuk alá:

```
# gpg --edit-key 95F61771
gpg (GnuPG) 1.0.7; Copyright (C) 2002 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

gpg: checking the trustdb
gpg: checking at depth 0 signed=1 ot(-/q/n/m/f/u)=0/0/0/0/0/1
gpg: checking at depth 1 signed=1 ot(-/q/n/m/f/u)=1/0/0/0/0/0
pub  1024R/95F61771  created: 2003-12-10 expires: never      trust: -/q
(1). Sendmail Signing Key/2004 <sendmail@Sendmail.ORG>

Command> sign

pub  1024R/95F61771  created: 2003-12-10 expires: never      trust: -/q
             Fingerprint: 46 FE 81 99 48 75 30 B1  3E A9 79 43 BB 78 C1 D4

     Sendmail Signing Key/2004 <sendmail@Sendmail.ORG>

How carefully have you verified the key you are about to sign actually
belongs to the person named above?  If you don't know what to answer,
enter "0".
```
Milyen gondosan ellenőrizte, hogy az aláírni kívánt kulcs valóban a fent nevezett személyé? Ha nem tudja, mit válaszoljon, írja be: „0"

```
    (0) I will not answer. (default)
```
Nem válaszolok.
```
    (1) I have not checked at all.
```
Egyáltalán nem ellenőriztem.
```
    (2) I have done casual checking.
```
Utánanéztem.
```
    (3) I have done very careful checking.
```
Alaposan megvizsgáltam.

```
Your selection? 3
```
Mi a választása?

```
Are you really sure that you want to sign this key
with your key: "Winslow Henson <win.henson@vstout.vbrew.com>"
```
Biztos benne, hogy szeretné a kulcsot saját "Winslow Henson <win.henson@vstout.vbrew.com>" kulcsával aláírni?

```
I have checked this key very carefully.
```
Alaposan megvizsgáltam a kulcsot.

```
Really sign? y
```
Valóban aláírja?

```
You need a passphrase to unlock the secret key for
user: "Winslow Henson <win.henson@vstout.vbrew.com>"
```
A "Winslow Henson <win.henson@vstout.vbrew.com>" felhasználó titkos kulcsának megnyitásához jelszó szükséges.
```
1024-bit DSA key, ID 34C9B515, created 2003-07-23

Command> quit
Save changes? y
```
Menti a változtatásokat?

Miután a *sendmail* kulcsokat a kulcskarikára fűztük és aláírtuk,* ellenőrizzük a *sendmail* terjesztés archívumát. Az alábbi példában a *sendmail.8.12.11.tar.gz.sig* aláírási állománnyal ellenőrizzük a *sendmail.8.12.11.tar.gz* tömörített archívumot:

```
# gpg --verify sendmail.8.12.11.tar.gz.sig sendmail.8.12.11.tar.gz
gpg: Signature made Sun 18 Jan 2004 01:08:52 PM EST using RSA key ID 95F61771
gpg: Good signature from "Sendmail Signing Key/2004 <sendmail@Sendmail.ORG>"
gpg: checking the trustdb
gpg: checking at depth 0 signed=2 ot(-/q/n/m/f/u)=0/0/0/0/0/1
gpg: checking at depth 1 signed=0 ot(-/q/n/m/f/u)=2/0/0/0/0/0
```

A kimenet szerint a disztribúciós archívumot biztonságosan helyreállíthatjuk. Az archívum egy könyvtárt hoz létre és a *sendmail* kibocsátási száma alapján nevet ad neki. A példában letöltött archívum a *sendmail-8.12.11* nevű könyvtárat készíti el. A *sendmail* fordításához és beállításához szükséges állományok és alkönyvtárak ebben a könyvtárban találhatók.

A sendmail fordítása

A *sendmail* fordításához használjuk a fejlesztők által készített `Build` segédprogramot. A fordításhoz a legtöbb rendszeren elegendő az olyan parancsokhoz hasonló néhány utasítás, mint:

```
# cd sendmail-8.12.11
# ./Build
```

Elegendő az egyszerű `Build` parancs, ha nincsenek különleges kívánságaink. Amennyiben vannak, készítsünk a hálózati helyünkhöz egyedi konfigurációt a `Build` parancs részére. A *sendmail* a hálózati hely konfigurációs állományait a */devtools/Site* könyvtárban figyeli. Linux rendszeren a `Build` a *site.linux.m4*, a *site.config.m4* és a *site.post.m4* állományokat keresi. Ha ettől eltérő állománynevet alkalmazunk, azonosításához adjuk meg a `-f` argumentumot a `Build` parancssorában. Például:

```
$ ./Build -f ourconfig.m4
```

Ahogy az *.m4* állomány-kiterjesztés is sejteti, a `Build` konfiguráció *m4* parancsokból áll. A `Build` változóinak beállítására három parancsot használunk.

`define`
: A *define* parancs a változóban tárolt aktuális értéket módosítja.

`APPENDDEF`
: Az `APPENDDEF` makró egy változóban tárolt értékek után helyez el újabb értéket.

* A *PGPKEYS* állomány letöltésére és importálására nagyjából évente egyszer lehet szükség.

PREPENDDEF

A PREPENDDEF makró egy változóban tárolt értékek elé helyez el újabb értéket.

Tegyük fel, hogy a *deftools/OS/Linux* állomány, amely valamennyi Linux rendszerünk számára definiálja a Build jellemzőit, a súgóoldalakat az */usr/man* könyvtárba helyezi:*

```
define(`confMANROOT', `/usr/man/man')
```

Tegyük fel továbbá, hogy Linux rendszereink a súgóoldalakat az */usr/share/man* könyvtárban tárolják. A következő sort a *devtools/Site/site.config.m4* állományhoz adva utasíthatjuk a Build programot, hogy a súgóoldalak útvonalát az */usr/share/man* könyvtárra állítsa:

```
define(`confMANROOT', `/usr/share/man/man')
```

Lássunk még egy példát! Tegyük fel, hogy a *sendmail* programot úgy kell beállítanunk, hogy egy LDAP kiszolgálóról olvasson adatokat. Képzeljük el továbbá, hogy a sendmail -bt -d0.1 utasítással ellenőrizzük a *sendmail* fordítási opcióit, és az LDAPMAP nem szerepel a „Compiled with:" listában. Az LDAP támogatást úgy adhatjuk meg, hogy beállítjuk az LDAP értékeket a *site.config.m4* állományban és az itt látható módon újrafordítjuk a *sendmail* programot:

```
# cd devtools/Site
# cat >> site.config.m4
APPENDDEF(`confMAPDEF', `-DLDAPMAP')
APPENDDEF(`confLIBS', `-lldap -llber')
Ctrl-D
# cd ../../
# ./Build -c
```

Figyeljük meg a Build parancsot! Ha megváltoztatjuk a *siteconfig.m4* állományt és ismételten futtatjuk a Build utasítást, a -c parancssori kapcsolóval figyelmeztethetjük a programot a változtatásokra.

A legtöbb egyedi Build konfiguráció semmivel nem bonyolultabb az itt bemutatottnál, noha több mint száz változót állíthatunk be – ez jóval több, mint amennyit egy fejezetben bemutathatunk. A teljes listát megtaláljuk a *devtools/README* állományban.

A sendmail bináris állomány telepítése

A *sendmail* bináris állomány telepítésekor a felhasználói azonosítót többé már nem a rootra állítja a program, ezért külön felhasználói azonosítót és csoportos azonosítót kell létrehoznunk a telepítés előtt. Régebben szokás volt, hogy a *sendmail* bináris állomány felhasználói azonosítója a root volt, így bármely felhasználó a parancssorból küldhetett levelet, amely így az elküldésre váró levelek könyvtárába került. Ehhez azonban nem feltétlenül szükséges root beállítású bináris állomány. A megfelelő könyvtár-jogosultsá-

* Vegyük észre, hogy az m4 „kiegyensúlyozatlan" idézőjeleket használ: ` '.

gokkal a csoportra beállított bináris állomány is használható, és kisebb biztonsági kockázatot jelent.

Készítsünk egy *smmsp* felhasználót és csoportot, amelyet a *sendmail* akkor használhat, amikor levelezőként fut. Ehhez a rendszer megfelelő segédprogramjait vesszük igénybe. Lássuk most az */etc/passwd* és */etc/group* bejegyzéseket, melyeket minta Linux rendszerünkön létrehoztunk:

```
# grep smmsp /etc/passwd
smmsp:x:25:25:Mail Submission:/var/spool/clientmqueue:/sbin/nologin
# grep smmsp /etc/group
smmsp:x:25:
```

Mielőtt a frissen fordított *sendmail* programot telepítenénk, készítsünk biztonsági másolatot a *sendmail* bináris állományról, a *sendmail* segédprogramokról és az aktuális *sendmail* konfigurációs állományokról. (Sohase lehet tudni; esetleg vissza kell térnünk a régi *sendmail* konfigurációhoz, ha az új nem úgy működik, ahogy vártuk.) Miután a rendszerről biztonsági másolatot készítettünk, telepítsük az új *sendmail* programot és a segédprogramjait:

```
# ./Build install
```

A `Build install` telepíti a *sendmail* programot és a segédprogramokat, ami egy több mint 100 sorból álló kimenetet eredményez. Futásánál nem keletkezhet hiba! Vegyük észre, hogy a `Build` az *smmsp* felhasználót és csoportot használja, amikor a */var/spool/clientmqueue* könyvtárat létrehozza és amikor a *sendmail* bináris állományt telepíti. A várólista könyvtárának (queue) és a *sendmail* bináris állománynak a gyors ellenőrzésével megtekinthetjük a tulajdonjogokat és jogosultságokat:

```
drwxrwx---    2 smmsp    smmsp        4096 Jun  7 16:22 clientmqueue
-r-xr-sr-x    1 root     smmsp      568701 Jun  7 16:51 /usr/sbin/sendmail
```

A *sendmail* telepítését a program beállítása követi. A fejezetben elsősorban a *sendmail* konfigurálásával foglalkozunk.

A sendmail konfigurációs állományai

A *sendmail* olyan konfigurációs állományt olvas be (ennek neve általában */etc/mail/sendmail.cf*, vagy a régebbi terjesztésekben */etc/sendmail.cf*, esetleg */usr/lib/sendmail.cf*), amelynek értelmezése a *sendmail* számára meg se kottyan, a rendszergazdák viszont csak komoly munkával képesek elolvasni vagy szerkeszteni. Szerencsére a legtöbb *sendmail* konfigurációhoz nem szükséges a *sendmail.cf* állományt olvasnunk vagy szerkesztenünk. Az esetek többségében a *sendmail* konfiguráció makrókon alapul. A makróeljárás a legtöbb telepítéshez megfelelő konfigurációkat állít elő, de a kapott *sendmail.cf* állományt bármikor „feljavíthatjuk" manuálisan is.

Az *m4* makróprocesszor program egy makró konfigurációs állományt feldolgozva állítja elő a *sendmail.cf* állományt. Kényelmünk érdekében a makró konfigurációs állományra a fejezetben *sendmail.mc* néven hivatkozunk, de saját konfigurációs állományunknak ne adjuk a *sendmail.mc* nevet! Válasszunk olyan elnevezést, amely jobban utal az állomány céljára. Elnevezhetjük például arról a gazdagépről, amelyhez létrehoztuk – esetünkben például lehet *vstout.m4*. Ha minden konfigurációs állománynak egyedi nevet adunk, akkor egyetlen könyvtárban tárolhatjuk őket, de ez a karbantartás szempontjából is előnyös.

A konfigurációs folyamat alapvetően a *sendmail.mc* állomány elkészítését jelenti (az állomány a kívánt konfigurációt leíró makrókat tartalmazza), illetve a *sendmail.mc* feldolgozását az *m4* segítségével. A *sendmail.mc* állomány tartalmazhat alapvető *m4* parancsokat, például a *define* és *divert* utasítást, de a kimenet szempontjából az állomány legfontosabb sorai a *sendmail* makrók. A *sendmail.mc* állományban használt makrókat a *sendmail* fejlesztői definiálják. Az *m4* makróprocesszor a makrókat a *sendmail.cf* szintaxisának megfelelő kódra alakítja át. A *sendmail.mc* állomány makrókifejezései a makró nevével kezdődnek (ezeket nagybetűvel írjuk), ezt követik a makrókifejezésben használt paraméterek (zárójelben). A paramétereket „szó szerint" is átadhatjuk és beleírhatjuk a *sendmail.cf* állományba, de a makrófeldolgozás folyamatának szabályozására is használhatjuk őket.

A gyakran 1000 sornál is hosszabb *sendmail.cf* állománytól eltérően egy egyszerű *sendmail.mc* állomány sokszor még 10 soros sincs, ha a megjegyzéseket nem számítjuk.

A megjegyzések

A *sendmail.mc* állomány # karakterrel kezdődő sorait az *m4* nem értelmezi, és a kimenetet alapértelmezés szerint közvetlenül a *sendmail.cf* állományba írja. Ennek akkor vehetjük hasznát, ha a *sendmail.mc* és a *sendmail.cf* állományokban megjegyzéseket szeretnénk fűzni a konfiguráció működéséhez.

Ha olyan megjegyzést szeretnénk a *sendmail.mc* állományba helyezni, amely *nem* kerül bele a *sendmail.cf* állományba, használhatjuk az *m4 divert* és *dnl* parancsait. A `divert(-1)` hatására egyáltalán nem lesz kimenet; a `divert(0)` visszaállítja a kimenetet az alapértelmezettre. A két utasítás között elhelyezett sorokat a feldolgozó elveti, azokkal nem foglalkozik. Ezért az olyan megjegyzéseket, amelyeknek csak a *sendmail.mc* állományban kell megjelenniük, általában a `divert(-1)` és a `divert(0)` parancsok fogják közre. Egyetlen sor esetén ugyanezt az eredményt a `dnl` paranccsal érhetjük el. A parancsot annak a sornak az elejére helyezzük, amelyet csak a *sendmail.mc* állományban szeretnénk megadni. A `dnl` parancs jelentése: „törölj minden karaktert egészen a következő újsor karakterig, beleértve azt is". A `dnl` parancsot esetenként a makró parancssorok végéhez fűzzük, így minden további karaktert megjegyzésként kezel a rendszer.

A *sendmail.mc* állományok gyakran több megjegyzést tartalmaznak, mint konfigurációs parancsot! Nézzük meg most a *sendmail.mc* állomány szerkezetét és az állományban használt utasításokat.

Gyakori sendmail.mc parancsok

A legtöbb *sendmail.mc* állomány létrehozásához csupán néhány parancsot alkalmazunk. E jellemző parancsok közül láthatunk most néhányat, abban a sorrendben, ahogy ezek a *sendmail.mc* állományban általában megjelennek:

```
VERSIONID
OSTYPE
DOMAIN
FEATURE
define
MAILER
LOCAL_*
```

A listában szereplő, nagybetűvel írt parancsok a *sendmail* makrók. A *sendmailfejlesztők* az általuk készített makrókhoz hagyományosan nagybetűvel írott neveket választanak. Természetesen az itt bemutatottaknál több makró létezik. A *sendmail* makrók teljes listáját a *cf/README* állományban találhatjuk meg. Az előbbi listában a `define` parancsot kivéve minden tétel egy *sendmail* makró. A `define` parancs, amely kisbetűvel jelenik meg, egy alapvető *m4* parancs. Minden alapvető *m4* parancs kisbetűs. A *sendmail.mc* állományokban más alapvető *m4* parancsokat is használhatunk, sőt bármelyik érvényes *m4* parancsot elhelyezhetjük. Az előbbi egyszerű parancskészlet segítségével bemutatjuk a parancsok sorrendjét a *sendmail.mc* állományban. A következőkben különkülön is megvizsgáljuk az utasításokat.

VERSIONID

A `VERSIONID` makró a verzióval kapcsolatos információkat kezeli. A makró opcionális, de a legtöbb *sendmail.m4* állományban megtalálható. A parancs argumentumait tetszőleges formátumban adhatjuk meg, és a verzióról tetszőleges információt közölhetünk, de általában figyelembe vesszük verziókövető rendszerünk elvárásait. Ha nem használunk verziókövető rendszert, helyezzünk el leíró jellegű megjegyzést a mezőben. Egy verziókövetést nem alkalmazó rendszer *sendmail.mc* állományának `VERSIONID` makrója valahogy így nézne ki:

```
VERSIONID(`sendmail.mc, 6/11/2004 18:31 by Win Henson')
```

Figyeljük meg, hogy az argumentum egy fordított nyitó (`) és egy záró aposztróf (') között áll. Amikor a *sendmail* makrónak átadott kifejezés szóközt, különleges karaktereket vagy félreérthető értékeket tartalmaz, akkor két aposztróf közé zárjuk. Ez minden makróra érvényes, nem csak a `VERSIONID`-re.

OSTYPE

Az `OSTYPE` makró a makró konfigurációs állomány kötelező része. Az `OSTYPE` makró betölt egy *m4* forrásállományt, amely operációsrendszer-specifikus információkat határoz meg, például az állományok és könyvtárak útvonalát, a levelező útvonalneveit és a rendszerre jellemző levelező argumentumokat. Az `OSTYPE` parancsnak átadott egyetlen argumentum annak az *m4* forrásállománynak a neve, amely az operációs rendszerre vo-

natkozó információkat tárolja. Az OSTYPE állományok a *cf/ostype* könyvtárban találhatók. Az OSTYPE(`linux') parancs a *cf/ostype/linux.m4* állományt dolgozza fel.

A *sendmail* terjesztés több mint 40 operációs rendszerhez biztosít előre definiált makróállományt a *cf/ostype* könyvtárban, de magunk is létrehozhatjuk saját változatunkat egyedi Linux disztribúciónkhoz. Egyes Linux terjesztések – és itt kiemelhetjük a Debian terjesztéseket – saját definíciós állománnyal rendelkeznek, amely teljesen Linux-FHS kompatibilis. Ha operációs rendszerünkben létezik ilyen állomány, használjuk a *generic-linux.m4* állomány helyett. Az OSTYPE makró a *sendmail.mc* állomány egyik első parancsa, ugyanis számos más definíció függ tőle.

DOMAIN

A DOMAIN makró a *cf/domain* könyvtárból kiválasztott és megadott állományt dolgozza fel. A DOMAIN állomány nagy segítséget jelent, amikor egyazon hálózaton nagyszámú számítógépet konfigurálunk szabványos módon; jellemzően olyan elemeket állít be, mint például a levélközvetítő gazdagépek vagy a hub neve, melyeket a hálózat összes gazdagépe használ.

Azért, hogy a DOMAIN makrót a gyakorlatban alkalmazhassuk, létre kell hoznunk saját makróállományunkat, amely a hálózati helyhez szükséges szabványos definíciókat tartalmazza; ezt a *domain* alkönyvtárban helyezzük el. Ha a tartomány makróállományát *cf/domain/vbrew.m4* néven mentettük el, a *sendmail.mc* állományból így hívhatjuk meg:

```
DOMAIN(`vbrew')
```

A *sendmail* terjesztés több minta tartomány-makróállományt is tartalmaz, ezek modellként szolgálhatnak sajátunk létrehozásakor. A minták egyike a *domain/generic.m4*, melylyel később, a 12.4. példában találkozunk majd.

FEATURE

A FEATURE makró segítségével előre definiált *sendmail* szolgáltatásokat adhatunk a konfigurációhoz. A szolgáltatások széles skáláját megtaláljuk – a *cf/features* könyvtár körülbelül 50 szolgáltatási állományt tartalmaz. A fejezetben csak néhány, gyakran használt szolgáltatással ismerkedünk meg. Az összes szolgáltatásról részletes leírását olvashatunk a forrás csomagban található *cf/README* állományban.

Egy szolgáltatás használatba vételéhez helyezzünk el a következőhöz hasonló sort a *sendmail.mc* állományban:

```
FEATURE(név)
```

A *név* a szolgáltatás neve. Egyes szolgáltatások opcionális paramétereket is fogadnak a következő formátumban:

```
FEATURE(név,param)
```

A *param* a megadni kívánt paraméter.

define

Az *m4* `define` paranccsal beállíthatjuk a belső *sendmail.cf* makrók, opciók és osztályok értékeit. A `define` parancs első argumentuma a beállítandó változó *m4* neve, a második mező pedig az az érték, amelyre a változót állítani kívánjuk. Lássunk egy példát, hogyan használjuk a `define` parancsot a *sendmail.cf* makróban:

```
define(`confDOMAIN_NAME', `vstout.vbrew.com')
```

A parancs a következő bejegyzést helyezi el a *sendmail.cf* állományban:

```
Djvstout.vbrew.com
```

Ezzel a *sendmail.cf* makró `$j` változóját, amely a *sendmail* gazdagép teljes tartománynevét tartalmazza, a **vstout.vbrew.com** értékre állítjuk. A `$j` értékét általában nem szükséges manuálisan megadnunk, mert alapértelmezésben a *sendmail* magától a rendszertől kérdezi le a helyi gazdagép nevét.

A legtöbb *m4* változó alapértelmezett értéke megfelelő, ezért nem szükséges azokat az *m4* forrásállományban külön beállítanunk. Az `undefine` parancs a változók értékét az alapértelmezett értékre állítja vissza. Például:

```
undefine(`confDOMAIN_NAME')
```

Az utasítás a `confDOMAIN_NAME` értékét az alapértelmezett értékre állítja, még akkor is, ha a konfigurációban korábban megváltoztattuk azt egy másik gazdagépnévre.

A `define` paranccsal beállítható *m4* változók listája meglehetősen hosszú. A *cf/README* állomány mindegyiküket felsorolja. A listában megtaláljuk az *m4* változó nevét, a megfelelő *sendmail.cf* opció, makró vagy osztály nevét, a változó leírását és alapértelmezett értékét, melyet a rendszer használ, ha a változónak nem adunk meg más értéket.

A `define` paranccsal nem csak a *sendmail.cf* makrók, opciók és osztályok értékeit állíthatjuk be, a parancs arra is alkalmas, hogy az *m4* konfigurációkban használt értékeket, illetve a *sendmail* belső értékeit módosítsuk vele.

MAILER

Ha azt szeretnénk, hogy a *sendmail* a helyi kézbesítésen kívül bármilyen más módon is közvetítsen leveleket, a MAILER makróval utasíthatjuk, hogy milyen átvitelt alkalmazzon. A *sendmail* különféle levélátviteli protokollokat ismer; egyesek rendkívül fontosak, másokat csak ritkán használunk, és van néhány kísérleti jellegű is. A MAILER makró levelező argumentumait a 12.2. táblázat gyűjti egybe.

A legtöbb gazdagépnek csupán SMTP átvitelre van szüksége ahhoz, hogy más gépekre levelet küldhessen és azoktól leveleket fogadhasson, illetve a `local` levelezőre, hogy a rendszer felhasználóihoz eljuttathassa a leveleket. E kettő beállításához helyezzük el a `MAILER(`local')` és `MAILER(`smtp')` parancsokat a makró konfigurációs állományban. (Noha a `local` levélátvitel alapértelmezés szerint már beállított, az érthetőség kedvéért rendszerint megadjuk a konfigurációs állományban is.)

12.2. táblázat. A MAILER makró argumentumai

Argumentum	*Rendeltetés*
`local`	A *local* és a *prog* levelezők.
`smtp`	Az összes SMTP levelező: *smtp*, *esmtp*, *smtp8*, *dsmtp* és *relay*.
`uucp`	Az összes UUCP levelező: *uucp-old* (*uucp*) és *uucp-new* (*suucp*).
`usenet`	Usenet hírek támogatása a *sendmailben*.
`fax`	FAX támogatást biztosít a HylaFAX szoftverrel.
`pop`	Post Office Portocol (POP) támogatást biztosít a sendmailben.
`procmail`	Interfészt biztosít a *procmail*hez.
`mail11`	A DECnet *mail11* levelező.
`phquery`	A *phquery* program a CSO telefonkönyvekhez.
`qpage`	A QuickPage levelező, amellyel személyhívókra küldhetünk e-mail üzenetet.
`cyrus`	A *cyrus* és *cyrusbb* levelező.

A `MAILER(`local')` makró bekapcsolja a `local` levelezőt, amely a helyi leveleket kézbesíti a rendszer felhasználói között, valamint a `prog` levelezőt, amely a levélállományokat a rendszeren futó programokhoz küldi. A `MAILER(`smtp')` makró az összes levelezőt tartalmazza, amely ahhoz szükséges, hogy a hálózaton SMTP levelezést folytassunk. A `MAILER(`smtp')` makró a következő levelezőket helyezi el a *sendmail.cf* állományban:

`smtp`
A levelező csak a hagyományos 7 bites ASCII SMTP leveleket kezeli.

`esmtp`
A levelező az Extended SMTP-t (ESMTP) támogatja, amely érti az ESMTP protokollkiterjesztéseket, a MIME levelek összetett üzenettörzseit és fejlett adattípusait. Az SMTP levelezés esetén ez az alapértelmezett levelező.

`smtp8`
A levelező 8 bites adatokat küld a távoli kiszolgálóra, még akkor is, ha a távoli kiszolgáló nem támogatja az ESMTP-t.

`dsmtp`
A levelező támogatja az ESMTP ETRN parancsot, amely lehetővé teszi a rendeltetési rendszernek, hogy a kiszolgálón sorakozó leveleket lekérje.

`relay`
A levelező SMTP levelezést közvetít egy másik levelező kiszolgálón keresztül.

Az internetre kapcsolódó vagy azzal kommunikáló rendszereken szükség van a `MAILER(`smtp')` levelezőkészletre; az önálló hálózatok rendszereinek többsége is ezeket a levelezőket használja, mert a vállalati hálózatukon TCP/IP-t alkalmaznak. Annak ellenére, hogy a *sendmail* rendszerek döntő többsége ezeket a levelezőket igényli, alap-

értelmezés szerint telepítésük nem történik meg. Az SMTP levelezés támogatásához a `MAILER(`smtp')` makrót magunknak kell a konfigurációnkhoz adnunk.

*LOCAL_**

A LOCAL_CONFIG, LOCAL_NET_CONFIG, LOCAL_RULESET és a LOCAL_RULE_*n* makrókkal a *sendmail.cf* konfigurációs parancsokat közvetlenül elhelyezhetjük az *m4* forrásállományban. A parancsok - pontosan abban a formában, ahogy szerepelnek - bemásolódnak a *sendmail.cf* állomány megfelelő részébe. A következő összefoglalásból megtudhatjuk, hová helyezik a makrók a megadott parancsokat a *sendmail.cf* konfigurációban.

LOCAL_CONFIG
: Egy *sendmail.cf* parancsblokk kezdetét jelzi. A blokk a *sendmail.cf* helyi információkat tároló részébe kerül.

LOCAL_NET_CONFIG
: Újraíró szabályok blokkjának a kezdetét jelzi. Ezek a 0 jelű szabálykészletbe kerülnek, amelyet *parse* szabálykészletnek is nevezünk.

LOCAL_RULE_*n*
: Újraíró szabályok blokkjának a kezdetét jelzi. Ezek a 0, 1, 2 vagy 3 jelű szabálykészletbe kerülnek. Az *n* határozza meg, melyik szabálykészletbe kerülnek az újraíró szabályok.

LOCAL_RULESET
: A konfigurációhoz elhelyezni kívánt egyedi szabálykészlet kezdetét jelzi.

A makrók azt jelentik, hogy mindent, amit megtehetünk a *sendmail.cf* állományban, megtehetjük az *m4* makró konfigurációs állományban is, mert nem csak az összes *m4* makróhoz férhetünk hozzá, de az összes *sendmail.cf* parancshoz is. Természetesen mielőtt a *sendmail.cf* parancsokat használhatnánk, tudnunk kell, hogyan működnek. A következő részben röviden átnézzük a *sendmail.cf* konfigurációs parancsait.

A sendmail.cf konfigurációs nyelv

A *sendmail.cf* parancsokat csak ritkán használjuk konfigurációnkban, mert a *sendmail* fejlesztői által készített *sendmail* makrók képesek kezelni a gyakori konfigurációkat. Ennek ellenére érdemes ismernünk a *sendmail.cf* parancsokat azokra a ritka esetekre, amikor olyan igényű konfigurációval találkozunk, amelyre a *sendmail* fejlesztői egyszerűen nem gondoltak. A 12.3. táblázat a *sendmail.cf* konfigurációs parancsokat foglalja össze.

A két utolsó utasítás kivételével a táblázat minden parancsát használhatjuk a LOCAL_CONFIG makróval. A konfiguráció értékeit beállító *sendmail.cf* parancsokat tartalmazó szekciót a LOCAL_CONFIG makró nyitja meg. A parancsok lehetnek *sendmail.cf* adatbázis-deklarációk, makrók vagy osztályértékek. Alapvetően bármit elhelyezhetünk a szekcióban az újraíró szabályok kivételével. Ennek ellenére a 12.3. táblázatban bemu-

tatott *sendmail.cf* parancsok némelyikére egyszerűen nincs szükségünk a *sendmail.mc* állományban, még akkor sem, ha különleges konfigurációt állítunk össze.

12.3. táblázat. A sendmail.cf konfigurációs parancsok

Parancs	*Szintaxis*	*Jelentés*
Verziószint	`[Vszint/forgalmazó]`	A verziószintet jelzi.
Makró megadása	`Dxérték`	Az *x* makrót az *érték*re állítja.
Osztály megadása	`Ccszó1[ szó2] ...`	A *c* osztályt a *szó1*-re, *szó2*-re stb. állítja.
Állomány beolvasása	`Fcfájl`	Beolvassa a *c* osztályt a *fájl*ból.
Kulcsállomány	`Knévtípus [argumentum]`	A *név* adatbázist definiálja.
Opció beállítása	`Ooopció=érték`	Az *opció*t az *érték*re állítja.
Megbízható felhasználók	`Tfelh1[ felh2 ...]`	A megbízható felhasználók a *felh1*, *felh2*...
Precedencia megadása	`Pnév=szám`	A *név* precedenciáját a *szám* értékre állítja.
Levelező megadása	`Mnév, [mező=érték]`	A *név* levelezőt definiálja.
Fejléc megadása	`H[?mkapcsoló ?]név :formátum`	A fejlécformátumot állítja be.
Szabálykészlet beállítása	`Sn`	Az *n* számú szabálykészlet indítása.
Szabály megadása	`Rlhs rhs megjegyzés`	Az *lhs* minták átírása *rhs* formátumra.

Soha nem kell például a *sendmail.cf* `O` parancsokat a *sendmail.mc* konfigurációban megadnunk, mert az összes *sendmail.cf* opció beállítható a `define` parancs és az *m4* változók segítségével. Hasonlóképpen, az *m4* `MAILER` makrók az összes szükséges `M` parancsot elhelyezik a *sendmail.cf* állományban, ezért valószínűtlen, hogy a `LOCAL_CONFIG` segítségével kellene `M` parancsokat definiálnunk. A `T` és `P` parancsok szerepe korlátozott. A `T` parancs felhasználóneveket ad ahhoz a listához, amely a más felhasználónéven levél küldésére jogosult felhasználók nevét tartalmazza. A biztonság kérdését szem előtt tartva azt mondhatjuk, hogy a lista bővítésével érdemes csínján bánni, de még ha bővítjük is, használhatjuk a `confTRUSTED_USERS` define parancsot az *m4* állományban, vagy a `FEATURE(use_ct_file)` makrót és a define parancsot az */etc/mail/trusted-users* állományban. A `P` parancs a levelezési precedenciát határozza meg, de őszintén szólva az alapértelmezett *sendmail.cf* konfiguráció eleve több levelezési precedenciát definiál, mint amennyire valaha is szükségünk lehet.

A `LOCAL_CONFIG` parancsot leggyakrabban a `D`, `C`, `F` és `K` *sendmail.cf* parancsok követik. Ezek mindegyikével egyedi értékeket állíthatunk be, amelyeket később egyedi szabálykészletekben használhatunk. A `D` parancs egy *sendmail.cf* makró értéket állít be. A `C` parancs egy *sendmail.cf* osztályhoz állít be értéket a parancssorból. Az `F` parancs egy *sendmail.cf* osztályhoz rendel értéket valamely állományból. A `K` parancs egy adatbázist határoz meg, amelyből a *sendmail* értékeket emelhet ki. Az összes szabványos *sendmail.cf* makró, osztály és adatbázis használható a szabványos *m4* makrókon keresz-

tül. A D, C, F és K parancsokat csak azon ritka esetekben helyezzük el a *sendmail.mc* konfigurációban, amikor saját egyéni makrókat, osztályokat vagy adatbázisokat hozunk létre.

A H parancs egy levélfejlécet definiál. Az alapértelmezett konfiguráció már minden szabványos levélfejlécet tartalmaz, így valószínűtlen, hogy valaha is új típusú fejlécet kell definiálnunk. A fejléc-definíció elhelyezésének leggyakoribb oka, hogy különleges eljárással szeretnénk a fejléceket feldolgozni. (A *cf/cf/knecht.mc* állományban példát is láthatunk a fejléc-definícióra egy különleges feldolgozás esetén. Craig Hunt könyve, az O'Reilly kiadásában megjelent *sendmail Cookbook* 6.9. eljárása nagyszerűen mutatja, hogyan hívhatunk meg különleges fejlécfeldolgozást.) Természetesen ha különleges fejlécfeldolgozást alkalmazunk, a feldolgozást végző szabálykészletet is meg kell írnunk. Az S és R parancsokat egyedi szabálykészletek létrehozására használhatjuk. Ez fejezetünk következő témája.

A sendmail.cf R és S parancsai

A *sendmail* vitathatóan leghatékonyabb tulajdonsága az újraíró szabály. Az újraíró szabályok határozzák meg, hogyan dolgozza fel a *sendmail* a levélüzeneteket. A *sendmail* a levélüzenet *fejléceiből* vett címeket átküldi az újraíró szabályok gyűjteményén. A gyűjtemény neve *szabálykészlet.* A *sendmail.cf* állományban minden szabálykészletnek egy S parancs ad nevet Sn formában, ahol az *n* az aktuális szabálykészlethez rendelendő név vagy szám.

Magukat a szabályokat a szabálykészletként csoportba sorolt R parancsok határozzák meg. Minden szabálynak van egy bal és egy jobb oldala, amelyeket legalább egy tabulátor karakter választ el.* Amikor a *sendmail* egy levél címét feldolgozza, egyezést keresve végignézi az újraíró szabályok bal oldalát. Ha a cím megfelel egy újraíró szabály bal oldalának, a cím helyére a jobb oldal tartalmát helyezi, és a feldolgozás megismétlődik. Ily módon az újraíró szabályok a levélcímet egyik formából egy másik formába alakítják át. Felfoghatjuk úgy is, mintha egy keress-és-helyettesíts szerkesztő parancs futna le, amely minden, a megadott mintával megegyező szöveget egy másik szövegre cserél fel.

A *sendmail* szabálykészlet ezért így néz ki:

```
Sn
Rlhs    rhs
Rlhs2   rhs2
```

A bal oldal

Az újraíró szabály bal oldala egy mintát határoz meg. Ha a minta illeszkedik egy címre, a szabály átalakítja a címet. A minta tartalmazhat szó szerint értelmezett karaktereket, *sendmail.cf* makrókat és osztályokat, valamint a következő metaszimbólumokat:

* A bal és jobb oldalt csak tabulátor választhatja el.

`$@`	Pontosan nulla tokenre illeszkedik.
`$*`	Nulla vagy több tokenre illeszkedik.
`$+`	Egy vagy több tokenre illeszkedik.
`$-`	Pontosan egy tokenre illeszkedik.
`$=`*x*	Az <*x* osztály bármely értékére illeszkedik.
`$~`*x*	Bármely értékre illeszkedik, ami nem szerepel az *x* osztályban.

A token vagy egy operátorral határolt karakterlánc, vagy egy határoló operátor. Az operátorokat a *sendmail.cf* `OperatorChars` opciója definiálja:

```
O OperatorChars=.:%@!^/[ ]+
```

Vegyük most a következő címet:

```
alana@ipa.vbrew.com
```

Az e-mail cím hét tokent tartalmaz: alana, `@`, `ipa`, `.`, `vbrew`, `.`, és `com`. A két pont (`.`) és az `@` operátorok, a maradék négy token pedig karakterlánc. A cím illeszkedne a `$+` szimbólumra, mert több mint egy tokent tartalmaz, de nem illeszkedne a `$-` szimbólumra, mert nem pontosan egy tokent tartalmaz.

Amikor egy szabály illeszkedik egy címre, a kifejezésben lévő egyes mintákra illeszkedő szövegrészek különleges változókba kerülnek, amelyeket *határozatlan tokeneknek* nevezünk, és amelyeket a jobb oldalon most már felhasználhatunk. Az egyetlen kivétel a `$@`, amely egyik tokenre sem illeszkedik és ezért soha nem hoz létre a jobb oldalon felhasználható szöveget.

A jobb oldal

Amikor egy újraíró szabály bal oldala illeszkedik egy címre, az eredeti szöveg törlődik és a helyére a szabály jobb oldala kerül. A szó szerint értendő szövegek betűről betűre másolódnak az új címbe. A jobb oldalon szereplő *sendmail.cf* makrókat a program értelmezi, és a makró kimenetét írja az új címbe. Ahogy a bal oldalon is használhattunk különféle mintákra illeszkedő szimbólumokat, a jobb oldalhoz is létezik egy különleges szintaxis a címek átalakításához. Ezeket a következő listában foglaltuk össze:

`$`*n*
: A metaszimbólum helyére balról számítva az *n*-edik határozatlan token kerül.

`$[`*név*`$]`
: A karakterlánc helyére a megadott gazdagépnév kanonikus formája kerül.

`$(`*leképző kulcs* `$:`*alapértelmezett* `$)`
: Ez a különleges szintaxis a *kulcs*ot keresi ki a *leképző* nevű adatbázisban, és a kikeresés eredményével tér vissza. Ha a kikeresés sikertelen, az *alapértelmezett* értékkel tér vissza. Ha az alapértelmezett érték nincs megadva, és a kikeresés sikertelen, a kulcs értékével tér vissza.

`$>n`
A metaszimbólum meghívja az *n* számú szabálykészletet a sor további részének a feldolgozásához.

Egy illeszkedő újraíró szabályt addig próbálunk újra és újra illeszteni, amíg végül már nem illeszkedik. Ekkor a feldolgozás a következő szabályhoz lép tovább. Ezt a viselkedést megváltoztathatjuk, ha a jobb oldal elé helyezzük a két különleges hurokszabályozó metaszimbólum egyikét:

`$@`	A metaszimbólum befejezi a szabálykészletet.
`$:`	A metaszimbólum az adott szabályt fejezi be.

Létezik egy különleges jobb oldali szintaxis a levélkézbesítési művelet levelező, gazdagép és felhasználó hármasának létrehozására is. A szintaxis leggyakrabban a 0 szabálykészletben található, amely a levél kézbesítési címét dolgozza fel. A szimbólumok a következők:

`$#levelező`
A metaszimbólum megállítja a szabálykészlet kiértékelését és meghatározza az üzenet átviteléhez a kézbesítés következő lépésében használandó levelezőt. Az *error* különleges levelezőnek ezzel a meghívásával hibaüzenetet generálhatunk.

`$@gazdagép`
A metaszimbólum azt a gazdagépet adja meg, amelyhez az üzenetet kézbesítjük. Ha a rendeltetési gazdagép a helyi gazdagép, a szintaxis elhagyható a levélkézbesítési hármasból. A gazdagép a rendeltetési gépek kettősponttal elválasztott listája is lehet – a rendszer a megadott sorrendben próbálja elvégezni a kézbesítést.

`$:felhasználó`
A metaszimbólum a levélüzenetet fogadó felhasználót adja meg.

Egyszerű példa a szabálymintára

Azért, hogy jobban átlássuk, hogyan is működnek a makró behelyettesítő minták, nézzük meg a következő, bal oldali kifejezést:

```
$* < $+ >
```

A szabály nulla vagy több tokenre illeszkedik, amelyet a < karakter követ, amelyet egy vagy több token követ, amelyet a > karakter követ.

Ha a szabályt a `brewer@vbrew.com` vagy a `Head Brewer < >` kifejezésekre alkalmaznánk, nem illeszkedne. Az első karakterlánc azért nem egyezne, mert nem tartalmaz < karaktert, a második pedig azért nem, mert a `$+` *egy* vagy *több* tokenre illeszkedik, és a < és > karakterek között nincs token. Az olyan esetekben, amikor egy szabály nem illeszkedik, a szabály jobb oldala sem érvényesül.

Ha a szabályt a `Head Brewer < brewer@vbrew.com >` kifejezésre alkalmazzuk, illeszkedni fog, és a jobb oldalon a `$1` helyére a `Head Brewer`, a `$2` helyére pedig a `brewer@vbrew.com` kerül.

Ha a szabályt a `< brewer@vbrew.com >` kifejezésre alkalmazzuk, illeszkedni fog, mert a `$*` *nulla* vagy több tokenre illeszkedik; ekkor a jobb oldalon a `$1` helyére üres karakterlánc kerül.

Példa egy teljes újraíró szabályra

A következő példa a `LOCAL_NET_CONFIG` makró segítségével deklarál egy helyi szabályt és elhelyezi azt a 0 szabálykészlet végére. A 0 szabálykészlet a kézbesítési címet alakítja át levélkézbesítési hármassá, megjelölve a levelezőt, a felhasználót és a gazdagépet. A 12.1. példában egy minta újraíró szabályt látunk:

12.1. példa. Minta újraíró szabály

```
LOCAL_NET_CONFIG
R$*<@$*.$m.>$*      $#esmtp $@$2.$m. $:$1<@$2.$m.>$3
```

A `LOCAL_NET_CONFIG` makró meghatározza az *m4* számára, hogy az újraíró szabályt a 0 szabálykészletben helyezze el. Maga a szabály az `R` karakterrel kezdődik. Nézzük meg először a szabály bal, majd a jobb oldalát.

A bal oldal így fest: `$*<@$*.$m.>$*`.

A < és > karakterek fókusz karakterek, amelyeket a 3-as szabálykészlet illeszt be a címfeldolgozás korai szakaszában, és amelyek a levélcím gazdagépre vonatkozó részét fogják közre. A fókusz karakterek minden címbe bekerülnek az újraírás alkalmával. Az @ „szó szerint" az @ karakter, melyet internetes e-mail címekben használunk a felhasználói és a gazdagéprész elválasztására. A pontok (.) a tartománynevekben használt és itt szintén szó szerint értelmezett pontok. A `$m` a helyi tartománynév tárolására szolgáló *sendmail.cf* makró. A maradék három elem `$*` metaszimbólumok.

A szabály minden ehhez hasonló e-mail címre illeszkedik: `CímzettFelhasználó<@Gazdagép.Tartományunk.> Valami Szöveg`. Magyarul a szabály a tartományunk bármely gazdagépének bármely felhasználójához címzett levélre illeszkedik.

Egy újraíró szabály bal oldalának metaszimbólumaira illeszkedő minden szöveg egy határozatlan tokenbe kerül, ezeket a jobb oldalon felhasználjuk. A példánkban a `$*` a cím elejétől a <@ kifejezésig minden szövegre illeszkedik. A karakterlánc a `$1` változóba kerül, melyet a jobb oldalon felhasználhatunk. Hasonlóképpen, az újraíró szabály második `$*` metaszimbólumával egyező szöveg a `$2` változóba kerül, míg az utolsó `$*` illesztése a `$3` változóba.

Amikor a szabály a tartományunk valamely gazdagépe valamely felhasználójának címére illeszkedik, a felhasználónevet a `$1`, a gazdagépnevet a `$2`, míg a maradék szöveget (ha van) a `$3` változóban kapjuk meg. Az értékeket azután a jobb oldal dolgozza fel.

Újraíró szabályunk jobb oldala így fest: `$#esmtp $@$2.$m. $:$1<@$2.$m.>$3`.

A szabálykészlet jobb oldalának feldolgozásakor értelmezzük a metaszimbólumokat és elvégezzük a megfelelő behelyettesítéseket.

A $# metaszimbólum hatására a szabály egy adott levelezőt választ - esetünkben ez az *esmtp*.

A $@ metaszimbólum a rendeltetési gazdagépet határozza meg. Példánkban a rendeltetési gazdagépet a $2 . $m kifejezés hozza létre, amely a tartományunkban lévő gazdagép teljes hiteles neve. Az FQDN-t a bal oldalon a $2 változóba helyezett gazdagépnév komponensből és a tartománynévből (. $m .) állítjuk össze.

A $: metaszimbólum a fogadó felhasználó címét adja meg. Ez a fogadó teljes e-mail címe, amelyet esetünkben a $1 < @ $2 . $m . > $3 kifejezés ad meg: felhasználó, zárójel, at jel (kukac), gazdagép, pont, tartomány, pont, zárójel, további szöveg.

Miután a szabály megjelöl egy levelezőt, az üzenet a levelezőhöz kerül kézbesítésre. Példánkban az üzenet az SMTP protokoll segítségével kerülne a rendeltetési gazdagépre.

A sendmail konfiguráció létrehozása

A fejezetben a *sendmail.mc* és a *sendmail.cf* állományokról tanultak alapján most már képesek vagyunk egyszerű *sendmail* konfiguráció készítésére. Lássunk is hozzá, és nézzünk meg egy minta *sendmail.mc* állományt!

A *sendmail* terjesztésben számos minta makró konfigurációs állományt találunk a *cf/cf* könyvtárban. Közöttük sok az általános célú konfigurációs állomány a különböző operációs rendszerekhez. A *generic-linux.mc* állomány például Linux rendszerekhez készült. A 12.2. példa az állomány tartalmát mutatja:

12.2. példa. a generic-linux.mc állomány

```
divert(-1)
#
# Copyright (c) 1998, 1999 Sendmail, Inc. and its suppliers.
#       All rights reserved.
# Copyright (c) 1983 Eric P. Allman.  All rights reserved.
# Copyright (c) 1988, 1993
#       The Regents of the University of California.  All rights reserved.
#
# By using this file, you agree to the terms and conditions set
# forth in the LICENSE file which can be found at the top level of
# the sendmail distribution.
#
#

#
#  This is a generic configuration file for Linux.
#  It has support for local and SMTP mail only.  If you want to
#  customize it, copy it to a name appropriate for your environment
#  and do the modifications there.
#

divert(0)dnl
VERSIONID(`$Id: ch12,v 1.6 2005/01/19 03:22:50 free2 Exp adam $')
OSTYPE(`linux')dnl
DOMAIN(`generic')dnl
MAILER(`local')dnl
MAILER(`smtp')dnl
```

A konfigurációs állománnyal kapcsolatban néhány dolog akkor is világos, ha semmit nem tudunk az állomány szintaxisáról. Először is, a *sendmail.mc* név nyilvánvalóan nem szent és sérthetetlen. A *generic-linux.mc* éppen úgy megfelel, és jóval világosabban utal az állomány céljára. Másodszor, a konfigurációs állomány nagyon rövid. A 12.2. példa sorainak legnagyobb részét a megjegyzések teszik ki, és csupán az utolsó öt sor valódi *sendmail* konfigurációs parancs. Harmadszor, a *sendmail* konfigurációs parancsok rövidek, és szintaxisuk meglehetősen egyszerű.

A *generic-linux.mc* állomány öt aktív sora négy különböző makróból áll össze. A *generic-linux.mc* állomány `VERSIONID` makrója a következő:

```
VERSIONID(`$Id: ch12,v 1.6 2005/01/19 03:22:50 free2 Exp adam $')
```

A *generic-linux.mc* állomány `OSTYPE(`linux')` parancsa betölti a *cf/ostype/linux.m4* állományt, amelyet röviden mi is ismertetünk a későbbiekben. A `DOMAIN(`generic')` makró feldolgozza a *cf/domain/generic.m4* állományt, amelyről rövidesen szintén szó lesz.

Végül a `MAILER(`local')` és `MAILER(`smtp')` makrók feladata a `local`, `prog`, `smtp`, `esmtp`, `smtp8`, `dsmtp` és `relay` levelezők hozzáadása a *sendmail.cf* konfigurációhoz.

Ez az öt makrósor hozza létre a teljes általános Linux *sendmail* konfigurációt. A *linux.m4* és *generic.m4* állományokban azonban további részleteket is találunk.

A linux.m4 OSTYPE állomány

A 12.3. példa a *cf/ostype/linux.m4* állományt mutatja be.

12.3. példa. A linux.m4 OSTYPE állomány

```
divert(-1)
#
# Copyright (c) 1998, 1999 Sendmail, Inc. and its suppliers.
#       All rights reserved.
# Copyright (c) 1983 Eric P. Allman.  All rights reserved.
# Copyright (c) 1988, 1993
#       The Regents of the University of California.  All rights reserved.
#
# By using this file, you agree to the terms and conditions set
# forth in the LICENSE file which can be found at the top level of
# the sendmail distribution.
#
#

divert(0)
VERSIONID(`$Id: ch12,v 1.6 2005/01/19 03:22:50 free2 Exp adam $')
define(`confEBINDIR', `/usr/sbin')
ifdef(`PROCMAIL_MAILER_PATH',,
        define(`PROCMAIL_MAILER_PATH', `/usr/bin/procmail'))
FEATURE(local_procmail)
```

Az állomány megjegyzésekkel és egy VERSIONID makróval kezdődik. Ezután a végrehajtható állományokat tartalmazó könyvtár útvonalát határozzuk meg. Az útvonalat a confEBINDIR *m4* változóban tároljuk. A *linux.m4* állomány ezt az útvonalértéket állítja a */usr/sbin* útvonalra. Ezt követően, ha a procmail útvonala még nem definiált, az */usr/bin/procmail* útvonalra állítjuk. Az állomány utolsó sora egy FEATURE makró, amely a local_procmail szolgáltatást tölti be; hatására a *sendmail* a *procmailt* használja local levelezőként. Az *m4* a local_procmail szolgáltatás létrehozásakor a PROCMAIL_MAILER_PATH útvonalra támaszkodik. Kiváló példa arra, hogyan definiálunk először egy értéket és utána hogyan használjuk fel egy konfiguráció elkészítéséhez. A *linux.m4* állomány azt is jól mutatja, hogy általában milyen típusú konfigurációs parancsokat találhatunk egy OSTYPE állományban.

A generic.m4 DOMAIN állomány

A *cf/domain/generic.m4* állomány mintául szolgál a DOMAIN állományokhoz, amelyet a *sendmail* fejlesztői biztosítanak számunkra. Az állományt a 12.2. példában bemutatott *generic-linux.mc* igényli. A *generic.m4* állományt a 12.4. példában mutatjuk be.

12.4. példa. A generic.m4 DOMAIN állomány

```
divert(-1)
#
# Copyright (c) 1998, 1999 Sendmail, Inc. and its suppliers.
#       All rights reserved.
# Copyright (c) 1983 Eric P. Allman.  All rights reserved.
# Copyright (c) 1988, 1993
#       The Regents of the University of California.  All rights reserved.
#
# By using this file, you agree to the terms and conditions set
# forth in the LICENSE file which can be found at the top level of
# the sendmail distribution.
#
#


#
#  The following is a generic domain file.  You should be able to
#  use it anywhere.  If you want to customize it, copy it to a file
#  named with your domain and make the edits; then, copy the appropriate
#  .mc files and change `DOMAIN(generic)' to reference your updated domain
#  files.
#
divert(0)
VERSIONID(`$Id: ch12,v 1.6 2005/01/19 03:22:50 free2 Exp adam $')
define(`confFORWARD_PATH', `$z/.forward.$w+$h:$z/.forward+$h:$z/.for-
ward.$w:$z/.forward')dnl
define(`confMAX_HEADERS_LENGTH', `32768')dnl
FEATURE(`redirect')dnl
FEATURE(`use_cw_file')dnl
EXPOSED_USER(`root')
```

Az állomány ismét csak megjegyzésekkel és egy VERSIONID makróval kezdődik. A define parancs a *sendmail* számára határoz meg keresési útvonalat, amelyen a felhasználó *forward* állományát keresheti. A parancs a *sendmail.cf* állományban állítja be a ForwardPath opciót. A $z, $w és a $h a *sendmail.cf* belső makrói.*

A második define parancs az egyetlen levélben található fejlécek maximális hosszát 32 768 bájtban határozza meg. Ehhez a *sendmail.cf* állomány MaxHeadersLength opcióját állítja be.

A következő két sor szolgáltatásokat ad a *sendmail* konfigurációhoz. A FEATURE(`redirect') makró a .REDIRECT pszeudotartomány támogatását kapcsolja be. A pszeudotartomány voltaképpen tartományszerű kiterjesztés, amelyet a *sendmail* belsőleg helyez el az e-mail címben, hogy a cím különleges kezelését biztosítsa. A .REDIRECT pszeudotartomány az *aliases* adatbázissal együttműködve kezeli olyan személyek leveleit, akik már nem olvassák leveleiket a mi hálózati helyünkön, de továbbra is kapnak leveleket a régi címükre.** A szolgáltatás bekapcsolása után minden elavult levelezési címhez adjunk egy alias címet, ebben a formában:

```
régi-cím új-cím.REDIRECT
```

Ennek a sornak a hatására hibaüzenetben értesítjük a levél küldőjét, milyen címen próbálhatja meg elérni a címzettet:

```
551 User not local; please try <új-cím>
```

A FEATURE(`use_cw_file') parancs beolvassa az */etc/mail/local-host-names* állományt, és az ott felsorolt gazdagépneveket a *sendmail.cf* $=w osztályába helyezi. A $=w osztály tartalmazza azon gazdagépek nevét, amelyek részére a helyi számítógép helyi leveleket fogad. Általában amikor egy *sendmailt* futtató rendszer a hálózatról egy másik gazdagépnévre címzett levelet fogad, felételezi, hogy a levél ahhoz a gazdagéphez tartozik és továbbítja hozzá, ha átvitelre konfigurálták, illetve eldobja, ha nem. Ha a rendszer egy másik gazdagéphez címzett levelet elfogad helyi levélként, a másik gazdagép neve bekerül a $=w osztályba. Amikor a use_cw_file tulajdonságot használjuk, minden, a *local-host-names* állományban szereplő gazdagépnév a $=w osztályba kerül.

A *generic.m4* állomány utolsó sora az EXPOSED_USER makró. Az EXPOSED_USER makró felhasználóneveket ad a $=E osztályhoz. A $=E osztályban szereplő felhasználókat a rendszer nem álcázza, még akkor sem, ha az álcázás (masquerading) bekapcsolt állapotban van. (Az álcázás a kimenő levelekben elrejti a valódi gazdagépnevet, és helyette a külvilág számára mutatni kívánt gazdagép nevét helyezi.) Bizonyos felhasználónevek, például a *root* sok rendszeren előfordulnak, ezért a tartományban nem egyediek. Az ilyen felhasználónevek esetében a cím gazdagéprészének átalakítása megnehezíti annak eldöntését, hogy valójában honnan is érkezett az üzenet, és lehetetlenné teszi a vá-

* Ezekről a *sendmail.cf* makrókról további információt találunk a *cf/README* állományban. A *Sendmail Installation and Operations Guide* leírás a *doc/op.ps* állományban a *sendmail.cf* makrók teljes listáját tartalmazza.

** Az *aliases* adatbázissal a fejezet későbbi részében foglalkozunk.

laszadást. A *generic.m4* állomány `EXPOSED_USER` parancsa megakadályozza, hogy ez előforduljon, biztosítva, hogy a *root* nem álcázott.

A *generic-linux.mc*, a *linux.m4* és a *generic.m4* állomány parancsai alapján az általános Linux konfiguráció a következő műveleteket végzi:

- A végrehajtható állományok útvonalának az */usr/sbin* útvonalat állítja be.
- A `procmail` útvonalát az */usr/bin/procmail* útvonalra állítja be.
- A `procmail`t használja `local` levelezőként.
- Megadja a .forward állományok keresési útvonalát.
- Támogatást biztosít a konfigurációban a `.REDIRECT` pszeudotartományhoz.
- Feltölti a `$=w` osztályt az */etc/local-host-names* állományból.
- A root felhasználót a `$=E` osztályhoz adja.
- A konfigurációhoz adja a `local`, `prog`, `smtp`, `esmtp`, `smpt8`, `dsmtp` és `relay` levelezőket.

A *sendmail* programot beállíthatjuk a Linux terjesztésünkben biztosított konfiguráció módosításával, illetve egyedi konfiguráció létrehozásával a *sendmail* terjesztésben kapott *generic-linux.mc* állomány alapján. A következő részben a *generic-linux.mc* állományt alapul véve hozunk létre egy *sendmai* konfigurációt.

Minta sendmail konfiguráció készítése Linux rendszerre

Egyedi konfigurációnk megépítését azzal kezdjük, hogy megváltoztatjuk a konfigurációs könyvtárt, és a *generic-linux.mc* konfigurációs állományt egy munkaállományba másoljuk. Mivel a konfigurációs állományt a **vstout.vbrew.com** géphez hozzuk létre, a munkaállománynak a *vstout.mc* nevet adjuk.

```
$ cd sendmail-8.12.11/cf/cf
$ cp generic-linux.mc vstout.mc
```

A *vstout* gépet levelezőkiszolgálóvá tesszük csoportunk számára. Elvárásaink:

- Fogadja a különféle ügyfelek leveleit, amelyek a kiszolgálón kívánják tárolni levelezésüket. Ehhez nincs szükség az *m4* konfiguráció módosítására, mert a Linux konfigurációban használt *domain/generic.m4* állomány már tartalmazza a `FEATURE(`use_cw_file')` parancsot.
- Továbbítsa a leveleket a **vbrew.com** tartományban. Ehhez nem szükséges megváltoztatnunk az *m4* konfigurációt. Alapértelmezés szerint a *sendmail* konfiguráció támogatja a továbbítást a *relay-domains* állományban megadott tartományokhoz. A fejezetben a *sendmail* adatbázisok ismertetésekor látni fogjuk, hogyan állíthatjuk be a *relay-domains* állományt.
- Írja át a kimenő levelekben a küldő címét a **vbrew.com** tartományban használt általános formátumra. Ehhez beállítjuk a támogatást a *genericstable* adatbázishoz. A *genericstable* adatbázis tartalmáról a fejezet *sendmail* adatbázisokkal foglalkozó részében lesz szó.
- Támogassa a kéretlen levelek (spam) elleni védekezést. Ehhez az *access* adatbázist veszszük igénybe.
- Legyen a beállítása egyszerű akkor is, ha további védelmi intézkedéseket szeretnénk hozni. A kéretlen levelek kezelésére alkalmazott *access* adatbázis számos más, könnyen

beállítható szolgáltatást is biztosít. Az *access* adatbázisról bővebben is szólunk a fejezet *sendmail* adatbázisokkal foglalkozó részében.

Azért, hogy a *sendmail* programot beállítsuk az előbbi feladatok elvégzésére, módosítjuk a *vstout.mc* állományt a következő szolgáltatások megadásával:

```
FEATURE(`genericstable')
GENERICS_DOMAIN(`vbrew.com')
FEATURE(`generics_entire_domain')
FEATURE(`access_db')
```

Az első sor a *genericstable* támogatását hozza létre. A második sor a **vbrew.com** tartományra alkalmazza a *genericstable* adatbázist. A harmadik sor arra utasítja a *sendmail* programot, hogy a *genericstable* adatbázist a **vbrew.com** valamennyi gazdagépére alkalmazza. Az utolsó sor az *access* adatbázis támogatását biztosítja.

A 12.5. példa a *vstout.mc* állományt mutatja a felesleges megjegyzések eltávolítása, a `VERSIONID` frissítése és az új sorok megadása után:

12.5. példa. Minta egyedi konfiguráció

```
VERSIONID(`Sample vstout configuration by Craig Hunt')
OSTYPE(`linux')dnl
DOMAIN(`generic')dnl
dnl Add support for the genericstable
FEATURE(`genericstable')
dnl Apply the genericstable to the vbrew.com domain
GENERICS_DOMAIN(`vbrew.com')
dnl Apply the genericstable to every host in the domain
FEATURE(`generics_entire_domain')
dnl Add support for the versatile access database
FEATURE(`access_db')
MAILER(`local')dnl
MAILER(`smtp')dnl
```

Most hozzunk létre egy *sendmail.cf* állományt a törzs konfigurációs állományból, telepítsük az új *sendmail.cf* állományt, és győződjünk meg róla, hogy a *sendmail* beolvassa!

A sendmail.cf elkészítése

A *sendmail.cf* állományt általában ugyanabban a *cf/cf* könyvtárban hozzuk létre, ahol a törzs konfigurációs állomány is készül. Ha jelenleg nem ebben a könyvtárban vagyunk, az állomány elkészítése előtt lépjünk be oda.

A `Build` paranccsal elkészíthetjük a *sendmail.cf* állományt az *m4* törzs konfigurációs állományból. A `Build` szkript használata nagyon egyszerű. A `Build` parancssorában argumentumként adjuk meg a kimeneti állományunknak szánt nevet. A szkript a kimeneti állomány *.cf* kiterjesztése helyére az *.mc* kiterjesztést helyezi, és az ilyen nevű makró konfigurációs állományt használja a kimeneti állomány létrehozásához. Más

szóval, ha a `Build` parancssorában a *vstout.cf* nevet adjuk meg, a *vstout.cf* létrehozásához a *vstout.mc* állományt használja fel. Íme egy példa:

```
$ ./Build vstout.cf
Using M4=/usr/bin/m4
rm -f vstout.cf
/usr/bin/m4 ../m4/cf.m4 vstout.mc > vstout.cf || ( rm -f vstout.cf && exit 1 )
chmod 444 vstout.cf
```

Bár a `Build` parancs egyszerű, sok rendszergazda nem alkalmazza, mert a *sendmail* konfiguráció elkészítéséhez használt *m4* parancssor szintén nagyon egyszerű. A *vstout.cf* állományt a *vstout.mc* állományból előállító *m4* parancssor a következő lenne:

```
$ m4 ../m4/cf.m4 vstout.mc > vstout.cf
```

A praktikus szemléletű *sendmail* adminisztrátor számára a `Build` szkript nem kínál különleges előnyöket. A legtöbbünknél a `Build` szkript és az *m4* parancs közötti választás csupán ízlés kérdése. A *Makefile-t* közvetlenül is meghívhatjuk egy egyszerű `make` paranccsal. Válasszuk a kényelmesebbnek tetsző megoldást.

Megépítése után ellenőrizzük az új *.cf* állományt a fejezetben később leírtak szerint, mielőtt arra a helyre másolnánk, ahol a *sendmail* a *sendmail.cf* konfigurációs állományt keresi – ez általában az */etc/mail/sendmail.cf*. Az állományt átmásolhatjuk például így:

```
# cp vstout.cf /etc/mail/sendmail.cf
```

De megtehetjük például a `Build` paranccsal is:

```
# mv vstout.cf sendmail.cf
# ./Build install-cf
Using M4=/usr/bin/m4
../../devtools/bin/install.sh -c -o root -g bin -m 0444 sendmail.cf
        /etc/mail/sendmail.cf
../../devtools/bin/install.sh -c -o root -g bin -m 0444 submit.cf
        /etc/mail/submit.cf
```

Az így megadott `Build install-cf` parancs két konfigurációs állományt telepít: a *sendmail.cf* állományt, és egy második állományt, melynek neve *submit.cf*. A *sendmail.cf* nem létezik, hacsak nem hozzuk létre. (Esetünkben *vstout.cf* néven valósítottuk meg, és ezt neveztük át *sendmail.cf* névre.) A *sendmail* terjesztés tartalmaz azonban egy teljes *submit.cf* állományt, melyet általában nem kell létrehoznunk, se módosítanunk. A *submit.cf* egy különleges konfiguráció, amelyet a *sendmail* akkor vesz igénybe, amikor levélküldő (mail submission) programként üzemel, míg a *sendmail.cf* a *sendmail* daemon által használt konfigurációs állomány. A `Build install-cf` parancsot általában akkor futtatjuk, amikor először telepítünk egy új *sendmail* terjesztést, így biztosítva, hogy mind a *sendmail.cf*, mind a *submit.cf* állomány telepítése megtörténik. Az első telepítéstől eltekintve azonban ritkán szükséges mindkét állományt egyidejűleg bemásolni, mert egy új *sendmail.cf* állomány létrehozásakor általában nincs szükség egy új *submit.cf* állomány létrehozására.

Miután a konfigurációt telepítettük, egy HUP jellel indítsuk újra a *sendmail* programot, hogy beolvassa az új konfigurációt. Az eljárás a szabványos *sendmail* jelfeldolgozást használja, amely csak Linux rendszereken létezik:

```
# kill -HUP `head -1 /var/run/sendmail.pid`
```

Egyes Linux rendszerek saját eszközöket biztosítanak a daemonok kezeléséhez. Bizonyos rendszerek például a `service` paranccsal indíthatják újra a *sendmailt:*

```
# service sendmail start
Starting sendmail: [  OK  ]
```

Függetlenül a *sendmail* újraindításának módjától, amikor a daemon futni kezd, beolvassa az */etc/mail/sendmail.cf* konfigurációs állományt, amely most már az új beállításokat tartalmazza.

A sendmail adatbázisok

A fent létrehozott minta konfiguráció több *sendmail* adatbázist is alkalmaz. (Az „adatbázis" kifejezést itt egy kissé engedékenyen használjuk, és a valódi adatbázisok mellett beleértjük az egyszerű szöveges állományokat is.) A *sendmail* adatbázisok a *sendmail* konfiguráció gyakran elhanyagolt elemei, noha fontos szerepet játszanak a konfigurációban. A napi változtatásokat ezekben az adatbázisokban végezzük, és nem az *m4* állományokban vagy a *sendmail.cf* állományban. A minta konfigurációnk *sendmail* adatbázisai a következők:

aliases
: Az *aliases* adatbázis alapértelmezés szerint a konfiguráció része. Az adatbázis alapvető összetevője a helyi levélkézbesítési eljárásnak és a levéltovábbításnak. Az *aliases* adatbázis használatához semmit nem szükséges változtatnunk a konfiguráción.

local-host-names
: A *local-host-names* állományt a *use_cw_file* szolgáltatással adhatjuk a konfigurációhoz. Az állomány határozza meg, milyen leveleket fogad el a rendszer helyi kézbesítésre.

relay-domains
: A *relay-domains* állomány alapértelmezés szerint a konfiguráció része. Ezért a *sendmail* konfiguráción semmit nem szükséges változtatnunk, ha használni akarjuk az állományt. A *relay-domains* állomány engedélyezi az átvitelt (relaying); ez alapértelmezés szerint kikapcsolt állapotban van.

genericstable
: A *genericstable* szolgáltatás a *genericstable* adatbázis támogatását biztosítja. Segítségével egy szervezeten belül használt e-mail címeket olyan egységes formára alakíthatjuk át, amelyet a szervezet a külvilág felé mutatni kíván.

access

Az *access_db* tulajdonság az *access* adatbázis támogatását biztosítja. Az *access* adatbázis rendkívül sok hasznos szolgáltatást biztosít.

A következőkben ezekkel, illetve más, a minta konfigurációban nem alkalmazott adatbázisokkal ismerkedünk meg.

Az aliases adatbázis

A levelek alias nevei nagyon jól használható szolgáltatás a levelek irányítására egy rendeltetési gazdagépen. Gyakran alkalmazzák például arra, hogy a World Wide Web kiszolgálókra vonatkozó visszajelzéseket, megjegyzéseket a „webmesterhez" irányítsák. A célgépen gyakran nincs is „webmester" nevű felhasználó, csupán egy hivatkozási név, amely egy másik felhasználóra mutat. Egy másik gyakori feladata a levelezőlisták létrehozása úgy, hogy egyetlen alias nevet több fogadóhoz irányítunk, vagy egy alias névre érkező üzeneteket a lista kiszolgáló programjához továbbítunk. Az alias nevek lehetőségei:

- Egy rövid vagy jól ismert névre írhatunk leveleket, amelyek egy vagy több személyhez is eljutnak. Az RFC kompatibilis rendszereken léteznie kell például olyan jól ismert neveknek, amilyen a *Postmaster* vagy a *MAILER-DAEMON*.
- Meghívhatunk egy programot úgy, hogy a bemenete a levélüzenet lesz. Amikor programok meghívására vagy programokba történő írásra állítunk be alias neveket, mindig nagyon ügyeljünk a biztonságra, mert előfordul, hogy a *sendmail* root jogosultságokkal fut.
- Leveleket állományba kézbesíthetünk.

A levelezés és az alias nevek kapcsolatáról részletesen olvashatunk az *aliases(5)* súgóoldalon. A 12.6. példa egy minta *aliases* állományt mutat be.

12.6. példa. Minta aliases állomány

```
#
# Az RFC kompatibilitáshoz a következő két alias feltétlenül szükséges.
# Fontos, hogy olyan 'személyre' mutassanak,
# aki rendszeresen olvassa a leveleket.
#
postmaster:    root                              # kötelező bejegyzés
MAILER-DAEMON: postmaster                         # kötelező bejegyzés
#
#
# Gyakori aliastípusok:
#
usenet:        janet                             # egy személyhez
admin:         joe,janet                         # több személyhez
newspak-users: :include:/usr/lib/lists/newspak # a címzettek kiolvasása
                  állományból
changefeed:    |/usr/local/lib/gup                # program hívására
complaints:    /var/log/complaints               # a levél állományba írására
#
```

Valahányszor frissítjük az */etc/aliases* szöveges állományt, mindig futtassuk a következő parancsot:

```
# /usr/bin/newaliases
```

A parancs újraépíti a *sendmail* által belsőleg használt adatbázist. A `newaliases` parancs egy szimbolikus link a *sendmail* futtatható állományhoz, amely pontosan úgy működik, mintha a *sendmail* programot a `-bt` parancssori argumentummal hívnánk meg.

A *sendmail* program az *aliases* adatbázis segítségével határozza meg, hogyan kezelje a helyi kézbesítésre elfogadott bejövő levélüzeneteket. Ha a levélüzenet kézbesítési címének felhasználói része megegyezik az *aliases* adatbázis egy bejegyzésével, a *sendmail* az üzenetet a bejegyzésnek megfelelően átirányítja. Ez azonban csak akkor történik meg, ha a *sendmail* elfogadta a levelet helyi kézbesítésre. A *sendmail* a *local-host-names* állomány alapján dönti el, melyik üzenetet fogadhatja el helyi kézbesítésre.

A local-host-names állomány

A bejövő leveleket vagy közvetlenül a címzetthez juttatjuk el, vagy egy másik gazdagéphez közvetítjük kézbesítésre. A *sendmail* csak olyan leveleket fogad helyi kézbesítésre, amelyeket egy helyi gazdagépre címeztek. Minden más levelet továbbít. A rendszer a `$=w` osztály ellenőrzésével dönti el, elfogad-e egy levelet helyi kézbesítésre. A `$=w` osztály egy tömb, amely a *sendmail* által a helyi kézbesítésre érvényesnek tekintett neveket tartalmazza.

A *use_cw_file* parancs hatására a *sendmail* betölti az */etc/mail/local-host-names* állományt a `$=w` osztályba. Ehhez a következő `F` parancsot helyezi el a *sendmail.cf* állományban:

```
Fw/etc/mail/local-host-names
```

Amikor a *use_cw_file* szolgáltatást megadjuk a konfigurációban, a *sendmail* megkísérli megtalálni a *local-host-names* állományt, és „nem végzetes hiba" (non-fatal error) hibaüzenetet küld, ha erre nem képes. Ha még nem kívánunk gazdagépneveket adni az állományhoz, egyszerűen készítsünk egy üres állományt.

Ha úgy szeretnénk a *sendmail* kiszolgálót beállítani, hogy más gazdagépeknek címzett leveleket fogadjon, egyszerűen adjuk meg a gazdagépek nevét a *local-host-names* állományban. A *local-host-names* állomány csupán a gazdagépnevek listáját tartalmazza – minden sorban egy gazdagép neve áll. Lássunk egy példát a *local-host-names* állományra a **vbrew.com** tartomány esetén:

```
vbrew.com
vporter.vbrew.com
vale.vbrew.com
vlager.vbrew.com
vpils.vbrew.com
vipa.vbrew.com
```

A *local-host-names* állomány értékei a `$=w` osztály egyéb értékeihez adódnak. A `$=w` osztályban tárolt egyéb értékek a gazdagéphez rendelt összes gazdagépnév, a gazdagép-

név alias nevei és IP címei, amelyeket a *sendmail* a különféle hálózati interfészek próbájával határozott meg. A *sendmail* által végzett interfészpróbákat korlátozhatjuk, ha a következő `define` parancsot elhelyezzük a *sendmail* konfigurációban:

```
define(`confDONT_PROBE_INTERFACES', `true')
```

A `confDONT_PROBE_INTERFACES` define parancsot általában csak akkor adjuk meg, ha a próbákkal a *sendmail* hibás információkhoz jut, vagy amikor sok virtuális interfészt alkalmazunk.

A `LOCAL_DOMAIN` makróval a *sendmail* konfigurációs állományban is hozzáadhatjuk a gazdagépneveket a `$=w` osztályhoz:

```
LOCAL_DOMAIN(`vbrew.com')
LOCAL_DOMAIN(`vipa.vbrew.com')
```

Azonban valahányszor egy `LOCAL_DOMAIN` parancsot helyezünk el a konfigurációban, a *sendmail.cf* állományt mindig újra kell építenünk, tesztelnünk és az */etc/mail* könyvtárba másolnunk. Amikor a *local-host-names* állományt alkalmazzuk, nincs szükség a *sendmail.cf* állomány újrafordítására csupán azért, mert a *local-host-names* állományt módosítottuk.

A bestmx_is_local tulajdonság

A *bestmx_is_local* tulajdonság további lehetőséget biztosít, hogy egy másik gazdagépnévre címzett levelet helyi kézbesítésre elfogadjunk. A megoldás jól működik, ha csupán azért adunk gazdagépneveket a *local-host-names* állományhoz, mert a helyi gazdagép e kiszolgálók előnyben részesített levélváltója (mail exchanger). A *bestmx_is_local* tulajdonság alkalmazásakor minden rendszerhez címzett levél helyi levélnek minősül, ha a rendszer a helyi gazdagépet jelöli meg előnyben részesített levélváltóként. Ha így szeretnénk eljárni, helyezzük el a következő sort a konfigurációban:

```
FEATURE(`bestmx_is_local', `vbrew.com')
```

A *bestmx_is_local* tulajdonság nagy előnye, hogy egyszerű - nem szükséges az MX ügyfelek gazdagépnevét hozzáadnunk a *local-host-names* állományhoz. A *bestmx_is_local* megoldás hátránya lehet, hogy minden egyes levél esetében megnöveli a feldolgozással járó munkát. Kis rendszerek esetében ez nem jelent gondot, a valóban nagy mennyiségű levelet kezelő rendszerek esetében azonban igen. A megoldás korlátja még, hogy a *bestmx_is_local* teljes mértékben az MX rekordoktól függ, pedig más oka is lehet, hogy leveleket helyiként fogadjunk el. A *local-host-names* állomány bármilyen gazdagépnevet képes tárolni, és nem kell azokra a gazdagépekre szorítkoznia, amelyek rendszerünket jelölik meg levélváltóként.

Azokat a leveleket, amelyeket nem a helyi gazdagépre címeztek, továbbítjuk. A *relay-domains* állomány a továbbítás beállításának egyik lehetséges eszköze.

A relay-domains állomány

Alapértelmezés szerint a *sendmail* nem engedélyezi a levelek továbbítását – még a helyi tartomány más gazdagépeiről sem. Ha valaki az alapértelmezett konfiguráció mellett egy rendszeren keresztül levelet próbál továbbítani, a „Relaying denied" (a továbbítás megtagadva) hibaüzenetet kapja. A *sendmail* azonban hajlandó a `$=R` osztályban felsorolt tartományokhoz levelet továbbítani, és a *relay-domains* állományban szereplő valamennyi bejegyzés hozzáadódik a `$=R` osztályhoz. A következő parancsok például úgy terjesztik ki a *relay-domains* állományt, hogy a **vbrew.com** tartományra is lehessen leveleket továbbítani.

```
# cat >> /etc/mail/relay-domains
vbrew.com
Ctrl-D
```

A *sendmail* újraindításával biztosítjuk, hogy beolvassa a *relay-domains* állományt:

```
# kill -HUP `head -1 /var/run/sendmail.pid`
```

Most már a helyi tartomány gazdagépei továbbíthatják a leveleket a **vstout.vbrew.com** gazdagépen keresztül – anélkül, hogy az *m4* konfigurációs állományt meg kellett volna változtatnunk, vagy a *sendmail.cf* állományt újra kellett volna fordítanunk. A **vbrew.com** tartományba érkező és a tartományt elhagyó leveleket továbbítjuk. Ezután sem továbbítjuk azonban azokat a leveleket, amelyek nem a **vbrew.com** tartomány valamely gazdagépére érkeznek vagy nem arról távoznak.

A levelek továbbítását más módon is bekapcsolhatjuk. Egyik sem olyan egyszerű azonban, mintha a helyi tartományt a *local-domains* állományhoz adnánk, és egyesek potenciális biztonsági kockázatot is hordoznak. Jó alternatíva lehet, ha a helyi tartománynevet a `RELAY_DOMAIN` makró segítségével a `$=R` osztályhoz adjuk. A következő sorokat a makró konfigurációhoz adva ugyanazt a hatást érjük el, mint az említett *relay-domains* állománnyal:

```
dnl a RELAY_DOMAIN egy tartománynevet ad az R osztályhoz
RELAY_DOMAIN(`vbrew.com')
```

A `RELAY_DOMAIN` parancs azonban az *m4* konfiguráció módosítását, valamint a *sendmail.cf* állomány újbóli megépítését és telepítését igényli. A *relay-domains* állomány esetén erre nincs szükség, ezért a *relay-domains* állományt egyszerűbben használhatjuk.

Egy további jó alternatíva a *relay-entire-domain* tulajdonság. A makró konfigurációhoz fűzött következő parancs a helyi tartomány összes gazdagépe számára bekapcsolja a levelek továbbítását:

```
dnl A tulajdonság levéltovábbítást végez a helyi tartomány számára
FEATURE(`relay_entire_domain')
```

A *relay-entire-domain* tulajdonság a `$=m` osztályban megadott tartományok bármely gazdagépéről továbbítja a leveleket. Alapértelmezés szerint a `$=m` osztály a kiszolgálórend-

szer tartománynevét tartalmazza, amely a *vstout.vbrew.comkiszolgálón* a *vbrew.com.* Ez az alternatív megoldás ugyan működik, de kissé összetettebb, mint a *relay-domains* állomány használata. Ráadásul a *relay-domains* állomány nagyon rugalmas, nem korlátozódik a helyi tartományra. A *relay-domains* állományban bármilyen tartományt megadhatunk, és e tartomány bármely gazdagépére érkező vagy arról távozó leveleket továbbíthatunk.

A levelek továbbítását más eljárásokkal is bekapcsolhatjuk, egyeseket azonban biztonsági okokból jobb elkerülnünk. Nézzünk most két ilyen eljárást:

promiscuous_relay
: A szolgáltatás minden gazdagéphez bekapcsolja a továbbítást. Természetesen ehhez a helyi tartomány is hozzátartozik, hogy a szolgáltatás működhessen. Ily módon azonban a rendszer nyílt továbbítást végez, ami megkönnyíti a levélszemét küldőinek feladatát. A *promiscuous_relay* szolgáltatást akkor is kerüljük, ha gazdagépünket tűzfal védi.

relay_local_from
: A szolgáltatással olyan levelezést továbbíthatunk, ahol az e-mail cím a levélboríték küldőjének címében a helyi tartomány valamely gazdagépének nevét tartalmazza. Mivel a levélboríték küldőjének címe hamisítható, a kéretlen levelek küldői kijátszhatják a kiszolgálót és rávehetik, hogy továbbítsa a levélszemetet.

Miután a *relay-domains* állományt beállítottuk, hogy a helyi tartományba, illetve a helyi tartományból leveleket továbbítson, a helyi hálózat ügyfelei a kiszolgálón keresztül leveleket küldhetnek a külvilágba. A *genericstable,* amelyről a következő részben lesz szó, lehetővé teszi, hogy miközben a levelek áthaladnak a kiszolgálón, átírjuk a feladók címét.

A genericstable adatbázis

A *genericstable* támogatására a minta *sendmail* konfigurációhoz adtuk a *genericstable* szolgáltatást, a `GENERICS_DOMAIN` makrót és a *generics_entire_domain* szolgáltatást. A következő parancsokat helyeztük el:

```
FEATURE(`genericstable')
GENERICS_DOMAIN(`vbrew.com')
FEATURE(`generics_entire_domain')
```

A *genericstable* szolgáltatás a *sendmail* számára a *genericstable* használatához nélkülözhetetlen kódot helyezi el. A `GENERICS_DOMAIN` makró a makró parancssorában megadott értéket írja a *sendmail* `$=G` osztályába. A `$=G` osztályban megadott értékek rendesen gazdagépnévként értelmeződnek, és csak a pontos egyezések indítják el a *genericstable* feldolgozást. A `generics_entire_domain` parancs hatására a *sendmail* a `$=G` osztályban tárolt értékeket tartománynévként értelmezi, és a tartományok bármely gazdagépét a *genericstable* segítségével dolgozza fel. Ennek értelmében a **vipa.vbrew.com**

gazdagépnevet, mivel a **vbrew.com** tartománynevet tartalmazza, a *genericstable* dolgozza fel a konfigurációnkban.

A *genericstable* minden bejegyzése két mezőt foglal magában: a kulcsot, és a kulcs visszatérési értékét. A kulcs mező vagy egy teljes e-mail cím, vagy egy felhasználónév. A visszatérési érték általában a felhasználónevet és a gazdagép nevét is tartalmazó teljes e-mail cím. A *genericstable* létrehozásához először készítsünk egy szöveges állományt a megfelelő adatbázis-bejegyzésekkel, majd a *genericstable* adatbázis létrehozásához hajtsuk végre az állományon a *makemap* parancsot. A **vstout.vbrew.com** kiszolgálóhoz a következő *genericstable* adatbázist hoztuk létre:

```
# cd /etc/mail
# cat > genericstable
kathy                          kathy.mccafferty@vbrew.com
win                            winslow.smiley@vbrew.com
sara                           sara.henson@vbrew.com
dave                           david.craig@vbrew.com
becky                          rebecca.fro@vbrew.com
jay                            jay.james@vbrew.com
alana@vpils.vbrew.com          alana.darling@vbrew.com
alana@vale.vbrew.com           alana.henson@vbrew.com
alana                          alana.sweet@vbrew.com
Ctrl-D
# makemap hash genericstable < genericstable
```

A *genericstable* esetén a *win@vipa.vbrew.com* a fejlécben szereplő küldő címét *winslow.smiley@vbrew.com* címre írjuk át; a `win` kulcshoz a *genericstable* ezt az értéket adja vissza. A példában a *vbrew.com* tartomány minden *win* fiókja Winslow Smiley-hoz tartozik. Nem számít, hogy a tartomány mely gépéről küld levelet, amikor a levél áthalad ezen a rendszeren, a *winslow.smiley@vbrew.com-ra* íródik át. Azért, hogy a válaszok az átírt cím mellett is a megfelelő személyhez kerüljenek, az átírt gazdagépnévnek olyan gazdagépet kell jelölnie, amely fogadja a leveleket és rendelkezik a *winslow.smiley* névhez alias névvel, amely a valódi *win* fiókra kézbesíti a leveleket.

A *genericstable* leképzés tetszőleges lehet. A példában bejelentkezési neveket képeztünk le a felhasználók valódi nevére, a helyi tartománynevet pedig *keresztnév.vezetéknév@tartomány* formátumban hoztuk létre.* Természetesen ha a kiszolgálóra a *keresztnév.vezetéknév@tartomány* címre levél érkezik, alias nevekkel kell a felhasználók valódi címére eljuttatnunk. A fent bemutatott *genericstable* bejegyzésekhez szükséges alias neveket a következőképpen helyezhetjük el az *aliases* adatbázisban:

```
# cd /etc/mail
# cat > aliases
kathy.mccafferty: kathy
win.strong:       craig
sara.henson:      sara
david.craig:      dave
rebecca.fro:      becky
```

* A *keresztnév.vezetéknév@tartomány* formátum nem általánosan támogatott. A sendmail FAQ dokumentumból megismerhetünk néhány érvet az ilyen címformátum használata ellen.

```
alana.smiley:       alana
alana.darling:      alana@vpils.vbrew.com
alana.henson:       alana@vale.vbrew.com
jay.james:          jay
Ctrl-D
# newaliases
```

A felhasználónak megfelelő alias nevek a leveleket a kiszolgálón tárolják, ahol a felhasználó elolvashatja, illetve az ügyfél a POP vagy az IMAP segítségével letöltheti. A teljes címeknek megfelelő alias nevek a leveleket a teljes címben megadott gazdagépre továbbítják.

A minta *genericstable* legtöbb bejegyzése (*kathy, sara, dave, becky és jay*) „bármilyen levél egyetlen címre" típusú leképzés, és ugyanúgy működnek, ahogy a *win* bejegyzés esetén már láttuk. Sokkal érdekesebb eset az *alana* felhasználónévé. A **vbrew.com** tartományban ugyanis három felhasználó is ezzel a felhasználónévvel rendelkezik: Alana Henson, Alana Darling és Alana Sweet. Az Alana Darling és Alana Henson esetében a *genericstable* kulcsokban megadott teljes címek lehetővé teszik, hogy a *sendmail* „egy az egybe" leképzést végezzen e címeknél. Az Alana Sweet bejegyzésnél alkalmazott kulcs azonban csupán egy felhasználónév. Ez a kulcs bármilyen bejövő címnek megfelel, amely tartalmazza az *alana* felhasználónevet, kivéve az *alana@vpils.vbrew.com* és az *alana@vale.vbrew.com* bemeneti címeket. Olyan rendszereken, amelyek több gazdagépről származó levelezést kezelnek, lehetőség van arra, hogy a bejelentkezési nevek több példányban létezzenek. Mivel a *genericstable* kulcs tartalmazhat teljes e-mail címeket is, az azonos felhasználóneveket megfelelően leképezhetjük.

A minta Linux *sendmail* konfigurációnkban használt utolsó adatbázis az *access*. Az adatbázis olyannyira sokoldalú, hogy érdemes minden levelezőkiszolgáló konfigurációjában elhelyezni.

Az access adatbázis

Az *access* adatbázis segítségével nagy rugalmassággal és több beleszólással állíthatjuk be, hogy mely gazdagépektől vagy felhasználóktól fogadunk el, illetve mely gazdagéphez és felhasználókhoz továbbítunk leveleket. Az *access* adatbázis olyan hatékony konfigurációs eszköz a leveleket továbbító kiszolgálók beállításához, amely a kéretlen levélszemét ellen is nyújt némi védelmet, és a továbbítási folyamat finomabb szabályozását teszi lehetővé, mint a *relay_domains* állomány. A *relay_domains* állománnyal ellentétben az *access* adatbázis nem alapértelmezett része a *sendmail* konfigurációnak. Az *access* adatbázist használva a minta Linux *sendmail* konfigurációhoz adtuk hozzá az *access_db* szolgáltatást:

```
FEATURE(`access_db')dnl
```

Az *access* adatbázis alapgondolata egyszerű. Amikor SMTP kapcsolat jön létre, a *sendmail* összehasonlítja a boríték fejlécéből származó információt az *access* adatbázisban tárolt információval, és ennek alapján dönti el, hogyan kezelje az üzenetet.

Az *access* adatbázis olyan szabályok gyűjteménye, amelyek leírják, hogy adott gazdagépről vagy felhasználótól érkező vagy oda tartó üzeneteket hogyan kezelje a rendszer. Az adatbázis formátuma egyszerű. A táblázat minden egyes sora tartalmaz egy hozzáférési szabályt. Minden szabály bal oldali része egy minta, amelyet összevetünk a levélüzenet borítékának fejléc-információjával. A jobb oldal azt a műveletet írja le, amelyet végrehajtunk, ha a borítékinformáció illeszkedik a mintára.

A bal oldali minta illeszkedhet:

- Egy személyre, akit vagy a teljes e-mail címmel (felhasználó@gazdagép.tartomány), vagy a következő formátumban írt felhasználónévvel adunk meg: *felhasználónév@.*
- Egy gazdagépre, melyet gazdagépnevével vagy IP címével adunk meg.
- Egy tartományra, melyet a tartomány nevével adunk meg.
- Egy hálózatra, melyet egy IP cím hálózati részével adunk meg.

Alapértelmezés szerint a mintának a borítékot küldő címére kell illeszkednie, és ennek megfelelően csak akkor kerül sor bármilyen műveletre, ha a levél a megadott címről érkezik. Ha a *sendmail* konfigurációban megadjuk a *blacklist_recipient* szolgáltatást, a mintát a forrás és a rendeltetési boríték címével is összevethetjük. A bal oldal elé elhelyezett opcionális elemekkel azonban finomabban szabályozhatjuk, hogy a minta illesztésére mikor kerüljön sor. Ha ilyen opcionális elem áll a minta elején, a *sendmail* a minta illesztését bizonyos feltételekhez köti. A három alapvető elem a következő:

`To:`
A művelet csak akkor hajtódik végre, ha a levelet a megadott címre küldik.

`From:`
A művelet csak akkor hajtódik végre, ha a levél a megadott címről érkezik.

`Connect:`
A művelet csak akkor hajtódik végre, ha a megadott cím az SMTP kapcsolat távoli végén lévő rendszer címe.

A hozzáférési szabályok jobb oldalán öt alapvető műveletet határozhatunk meg. Ezek:

`OK`
A levélüzenet elfogadása.

`RELAY`
A levélüzenet elfogadása továbbításra.

`REJECT`
A levél elutasítása egy általános üzenet kíséretében.

`DISCARD`
A levél elvetése a `$#discard` levelező használatával.

`ERROR:dsn:kód szöveg`
Hibaüzenet küldése a megadott DSN kód, a megadott SMTP hibakód és a megadott szöveges üzenet használatával.

Lássunk egy példát az */etc/mail/access* állományra!

```
friends@cybermail.com   REJECT
aol.com                 REJECT
207.46.131.30           REJECT
postmaster@aol.com      OK
linux.org.au            RELAY
example.com             ERROR:5.7.1:550 Relaying denied to spammers
```

A példa minden levelet elutasít, amely a *friends@cybermail.com* címről, az **aol.com** tartomány bármely gazdagépéről vagy a **207.46.131.30** gazdagéptől érkezik. A következő szabály annak ellenére is elfogadja a *postmaster@aol.com* címről érkező leveleket, hogy magára a tartományra elutasító szabály vonatkozik. Az ötödik szabály lehetővé teszi, hogy a **linux.org.au** tartomány bármely gazdagépéről leveleket továbbítsunk. Az utolsó szabály az **example.com** címről érkező leveleket egyedi hibaüzenet kíséretében elutasítja. A hibaüzenet tartalmazza az 5.7.1 kézbesítési állapotra vonatkozó kódot, amely az RFC 1893-ban leírt érvényes kód, és az 550-ös SMTP hibakódot, amely szintén érvényes kód az RFC 821 alapján.

Az *access* adatbázis a bemutatottnál jóval többre képes. Vegyük észre, hogy kifejezetten az „alapvető" elemekről és az „alapvető" műveletekről beszéltünk, mert több más érték is létezik, amelyet fejlett konfigurációkban használhatunk. Amennyiben összetett konfigurációt szeretnénk létrehozni, olvassuk el a fejezet *További ismeretek* című részét.

Az *access* adatbázis a minta konfigurációnkban használt utolsó adatbázis. Létezik néhány más, a minta Linux *sendmail* konfigurációban nem alkalmazott adatbázis is. Ezekről szólunk most.

Egyéb adatbázisok

A rendelkezésre álló *sendmail* adatbázisokból nem mindegyiket használtuk fel minta konfigurációnkban, vagy azért, mert használatuk ellenjavallt, vagy mert elavult eljárásokra vonatkoznak. A következő adatbázisokról van szó:

`define(`confUSERDB_SPEC', `útvonal')`
A `confUSERDB_SPEC` opció hatására a *sendmail* az *aliases* adatbázis alkalmazása után, de még a *.forward* állomány alkalmazása előtt a helyi címekre alkalmazza a *user* adatbázist. Az útvonal argumentum határozza meg, hogy a *sendmail* hol keresi az adatbázist. A *user* adatbázis használata nem terjedt el, mert a *sendmail* fejlesztői ellenjavallják a FAQ 3.3. és 3.4 kérdésekre adott válaszaikban.

`FEATURE(`use_ct_file', `útvonal')`
A *use_ct_file* tulajdonság hatására a *sendmail* a megadott állományból megbízható felhasználók neveit adja a `$=t` osztályhoz. Mivel a `$=t` osztályban megadott felhasz-

nálók más felhasználók nevében is küldhetnek leveleket, biztonsági kockázatot jelentenek. Kevesebb állomány szükséges az esetleges manipulációk elleni védekezéshez, ha a megbízható felhasználókat a makró konfigurációban adjuk meg a confTRUSTED_USERS segítségével; és mivel csak nagyon kevés felhasználóban szükséges valóban megbíznunk, megadásuk a makró konfigurációs állományban nem jelent különösebb nehézséget.

`FEATURE(`domaintable', `specifikáció')`

A *domaintable* szolgáltatás hatására a *sendmail* a tartománytáblázat alapján képezi le az egyik tartománynevet a másikra. Az opcionális adatbázis specifikációval megadhatjuk az adatbázis típusát és elérési útját; alapértelmezés szerint az adatbázis *hash* típusú és az */etc/mail/domaintable* állományra mutat. A *domaintable* megkönnyíti az átállást, ha régi tartománynevek helyett új tartományneveket szeretnénk alkalmazni, mert minden levél esetén lefordítja a régi nevet az újra. Mivel csak ritkán kerülünk ilyen helyzetbe, az adatbázist is csak ritkán használjuk.

`FEATURE(`uucpdomain', `specifikáció')`

Az *uucpdomain* szolgáltatás hatására a *sendmail* az *uucpdomain* adatbázis segítségével képezi le az UUCP hálózati helyneveket internetes tartománynevekre. Az opcionális adatbázis-specifikációval felülírhatjuk az adatbázis alapértelmezett *hash* típusát és az adatbázis alapértelmezett */etc/mail/uucpdomain* útvonalát. Az *uucpdomain* adatbázis az .UUCP pszeudo tartományból származó e-mail címeket a régi UUCP bang címekre (felkiáltójeles formátumra) alakítja. Az adatbázis kulcsa a .UUCP pszeudotartományból származó gazdagépnév. A kulcsra kapott visszatérési érték a bang cím. Valószínűtlen, hogy valaha is szükségünk lesz az adatbázisra, mert a UUCP-t változatlanul alkalmazó hálózati helyek is csak ritkán használnak bang címeket, hiszen a jelenlegi UUCP levelezők az internetes címekhez hasonló e-mail címeket kezelnek.

`FEATURE ('bitdomain', 'specifikáció')`

A *bitdomains* szolgáltatás hatására a *sendmail* a *bitdomain* adatbázis segítségével képezi le a BITNET gazdagépneveket internetes tartománynevekre. A BITNET egy elavult IBM nagyszámítógépes hálózat, amellyel nem fogunk kapcsolatba kerülni, így az adatbázisra sem lesz szükségünk.

További két adatbázis a *mailertable* és a *virtusertable,* amelyeknek – bár a minta konfigurációban nem szerepeltek – jó hasznát vehetjük.

A mailertable

A *mailertable* szolgáltatással a *mailertable* adatbázis támogatását biztosíthatjuk a *sendmail* konfigurációban. A *mailertable* szolgáltatás szintaxisa:

```
FEATURE(`mailertable', `specifikáció')
```

Az opcionális adatbázis-specifikációval felülírhatjuk az adatbázis alapértelmezett *hash* típusát és az adatbázis alapértelmezett */etc/mail/mailertable* útvonalát.

A *mailertable* a tartományneveket képezi le a belső levelezőre, amely a tartományhoz tartozó leveleket kezeli. Egyes levelezőket csak akkor használhatunk, ha a *mailertable*

adatbázisban hivatkozunk rájuk. A `MAILER(`smtp')` parancs például az *esmtp, relay, smtp, smtp8* és a *dsmtp* levelezőket adja a konfigurációhoz. Alapértelmezés szerint a *sendmail* csupán kettőt használ ezek közül. Az *esmtp* levelező feladata a szabványos SMTP levelek küldése, a *relay* levelező pedig a leveleknek egy külső kiszolgálón keresztüli továbbítására szolgál. A maradék három levelezőt a *sendmail* csak akkor használja, ha egy *mailertable* bejegyzés hivatkozik rájuk, illetve ha egyedi újraíró szabályban jelennek meg. (A *mailertable* használata jóval egyszerűbb, mint a saját újraíró szabályok létrehozása.)

Példaként vegyük az *smtp8* levelezőt! Az *smtp8* levelezőt arra tervezték, hogy 8 bites MIME adatokat küldhessen olyan elavult levelezőkiszolgálókra, amelyek támogatják a MIME-ot, de nem értik az Extended SMTP-t. Ha az **example.edu** tartomány ilyen levelezőkiszolgálót használna, a levelek kezelésére a következő bejegyzést helyezhetnénk el a *mailertable* adatbázisban:

```
.example.edu        smtp8:oldserver.example.edu
```

A *mailertable* bejegyzés két mezőből áll. Az első mező a kulcs, amely a kézbesítési cím gazdagéprészét tartalmazza. Ez lehet a teljes minősített gazdagépnév - **emma.example.edu** -, vagy egyszerűen egy tartománynév. A tartománynév megadásához kezdjük a nevet ponttal, mint az előbbi példában. Ha tartománynevet adunk meg, az a tartomány minden gazdagépére illeszkedik.

A második mező a visszatérési érték. Általában tartalmazza a levél kezelésére kijelölt levelező nevét és a kiszolgálót, amelyre a levelet küldeni szeretnénk. A felhasználónevet a kiszolgáló címével együtt is megadhatjuk a *felhasználó@kiszolgáló* formátumban. Emellett a kiválasztott levelező lehet a belső error levelező is. Ha az error levelezőt használjuk, a levelező nevét követő érték nem a kiszolgáló neve, hanem egy hibaüzenet. Lássunk egy példát e három választási lehetőségre!

```
.example.edu            smtp8:oldserver.example.edu
vlite.vbrew.com         esmtp:postmaster@vstout.vbrew.com
vmead.vbrew.com         error:nohost This host is unavailable
```

A *mailertable* adatbázison áthaladó leveleket általában a címzett felhasználóhoz küldjük. A *jane@emma.example.edu* címre címzett levelek például az *smtp8* levelezőn keresztül az **oldserver.example.edu** kiszolgálóra mennek a *jane@emma.exmple.edu* felhasználóhoz. Amikor azonban a második mezőben egy felhasználónevet adunk meg, a rendes viselkedést megváltoztatva a leveleket nem a levelezőkiszolgálóra, hanem egy önálló személyhez irányítjuk. A **vlite.vbrew.com** gazdagép bármely felhasználójához címzett levelek például a *postmaster@vstout.vbrew.com* felhasználóhoz kerülnek. Ott a leveleket feltételezhetően már kézzel dolgozzák fel. És végül, a *mailertable* kezelte leveleket nem feltétlenül kell kézbesítenünk, helyette hibaüzenetet is küldhetünk a feladónak. A **vmead.vbrew.com** gazdagépre címzett levelekre a Gazdagép nem érhető el (This host is unavailable) hibaüzenettel válaszolunk a küldőnek.

A virtusertable

A *sendmail virtusertable* szolgáltatása biztosítja a virtuális felhasználói táblázat támogatását; ebben a táblázatban történik a virtuális e-mail befogadásának (virtual e-mail host-

ing) beállítása. A virtuális e-mail befogadása lehetővé teszi a levelezőkiszolgálónak, hogy több más tartomány nevében leveleket fogadjon és kézbesítsen, mintha több önálló levelező gazdagép volna. A virtuális felhasználói táblázat a *felhasználó@gazdagép* címre szánt bejövő üzeneteket a *másikfelhasználó@másikgazdagép* címre képezi le. Úgy is felfoghatjuk, mint egy levelező fejlett alias szolgáltatását, amely nem csak a rendeltetési felhasználót, de a rendeltetési tartományt is figyelembe veszi működésekor.

A *virtusertable* tulajdonság beállításához vegyük fel a szolgáltatást az *m4* makró konfigurációba:

```
FEATURE(`virtusertable')
```

Alapértelmezés szerint a *virtusertable* forrásállomány az */etc/mail/virtusertable*. Ezt felülírhatjuk egy további argumentummal a makró definíciójában; az elérhető opciókról tájékozódhatunk a részletes *sendmail* referenciában.

A virtuális felhasználói táblázat formátuma roppant egyszerű. Minden egyes sor bal oldalán egy minta áll, amely az eredeti rendeltetési levélcímet reprezentálja; a jobb oldalon álló minta az a levélcím, amelyre a virtuálisan befogadott címet leképezzük. A következő példában négy lehetséges bejegyzést látunk:

```
samiam@bovine.net       colin
sunny@bovine.net        darkhorse@mystery.net
@dairy.org              mail@jhm.org
@artist.org             $1@red.firefly.com
```

A példában három tartomány virtuális befogadását valósítjuk meg; ezek a **bovine.net**, a **dairy.org** és az **artist.org**.

Az első bejegyzés a **bovine.net** virtuális tartomány egyik felhasználójához küldött leveleket a gép egyik helyi felhasználójához irányítja át. A második bejegyzés ugyanezen virtuális tartomány egy másik felhasználójához címzett leveleket egy másik tartományra irányítja át. A harmadik példa a **dairy.org** virtuális tartomány bármely felhasználójához címzett levelet irányítja át egyetlen távoli levélcímre. Végül az utolsó bejegyzés az **artist.org** virtuális tartomány egyik felhasználójához címzett összes levelet egy másik tartomány ugyanazon felhasználójához irányítja át; a *julie@artist.org* címre címzett leveleket például a *julie@red.firefly.com* címre irányítjuk.

A beállítások ellenőrzése

Az elektronikus levelezés alapvető, de kényes szolgáltatás, ha rosszul állítjuk be, a behatolók könnyen kihasználhatják. Ezért nagyon fontos, hogy alaposan ellenőrizzük a beállításokat. Szerencsére a *sendmail* aránylag egyszerű eljárással siet a segítségünkre.

Egy „címteszt" üzemmódot támogat a *sendmail,* amely számos különféle ellenőrzésre kínál lehetőséget. A következő példákban megadunk egy rendeltetési levelezési címet és egy tesztet, amellyel a címet szeretnénk vizsgálni. A *sendmail* ezután feldolgozza a rendeltetési címet, miközben megjeleníti az egyes szabálykészletek kimenetét. A *send-*

mail programot úgy kapcsolhatjuk tesztelő üzemmódba, hogy meghívjuk a `-bt` argumentummal.

A címteszt üzemmód alapértelmezett konfigurációs állománya az */etc/mail/sendmail.cf* állomány. Alternatív konfigurációs állományt a `-C` argumentummal adhatunk meg. Ez fontos, mivel az új konfigurációt érdemes tesztelnünk, mielőtt az */etc/mail/sendmail.cf* útvonalra másoljuk. A fejezet korábbi részében létrehozott minta Linux *sendmail* konfigurációs állomány tesztelését a következő *sendmail* paranccsal végezhetjük el:

```
# /usr/sbin/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
>
```

A > prompt azt jelzi, hogy a *sendmail* készen áll a teszt üzemmód utasításainak fogadására. Címteszt üzemmódban a *sendmail* különféle utasításokat fogad, ezekkel ellenőrizhetjük a konfigurációt és a beállításokat, és megfigyelhetjük, hogyan dolgozza fel a *sendmail* az e-mail címeket. A 12.4. táblázat a teszt üzemmód parancsait foglalja össze.

12.4. táblázat. A sendmail teszt üzemmódjában elérhető parancsok

Parancs	*Feladat*
szabálykészlet [,szabálykészlet . . .] *cím*	A *cím* feldolgozása a megadott, vesszővel elválasztott *szabálykészlet*ekkel.
=S*szabálykészlet*	A *szabálykészlet* tartalmának megjelenítése.
=M	Az összes levelező definíció megjelenítése.
$*v*	A *v* makró értékének megjelenítése.
$=*c*	A *c* osztály értékeinek megjelenítése.
.D*vérték*	A *v* makró beállítása az *érték*re.
.C*cérték*	Az *érték* bejegyzése a *c* osztályba.
-d*érték*	A hibakeresési szintet az *érték*re állítja.
/tryflags *kapcsolók*	A */try* által a címfeldolgozáshoz használt kapcsolók beállítása.
/try *levelező cím*	A *cím* feldolgozása a *levelező*vel.
/parse *cím*	A *cím* levelező/gazdagép/felhasználó kézbesítési hármas megjelenítése.
/canon *gazdagépnév*	A *gazdagépnév* kanonizálása.
/mx *gazdagépnév*	A *gazdagépnév* MX rekordjainak kikeresése.
/map *leképző kulcs*	A *kulcs* kikeresése a *leképző* adatbázisban.
/quit	Kilépés a címteszt üzemmódból.

Egyes parancsok (=S, =M, $*v* és a $=*c*) a *sendmail.cf* állományban definiált aktuális konfigurációs értékeket jelenítik meg, a `/map` parancs pedig a *sendmail* adatbázis állományaiban beállított értékeket mutatja. A `-d` paranccsal megváltoztathatjuk a megjelenített információ mennyiségét; számos hibakeresési szintet beállíthatunk, a *sendmail* rend-

szergazdának azonban csak egy párra van szüksége. A megadható hibakeresési értékekről a *sendmail* részletes referenciájában tájékozódhatunk.

A `.D` és `.C` parancsokkal valós időben állíthatjuk be a makró és az osztály értékeit. Az utasítással kipróbálhatunk eltérő konfigurációs beállításokat, mielőtt a teljes konfigurációt újraépítenénk.

Két parancs a *sendmail* és a DNS közötti kapcsolat megjelenítésére szolgál. A `/canon` egy adott gazdagépnévhez a DNS által visszaküldött kanonikus nevet mutatja. Az `/mx` egy adott gazdagéphez a DNS által visszaküldött levélváltók listáját tartalmazza.

A további parancsok többsége e-mail címeket dolgoz fel a *sendmail* újraíró szabályaival. A `/parse` a kézbesítési címek feldolgozását jeleníti meg, és megmutatja, hogy a címre küldött levelek kézbesítésében mely levelező vesz részt. A `/try` egy adott levelező esetében mutatja a feldolgozást. (A `/tryflags` utasítás határozza meg, hogy a `/try` parancs a küldő vagy a fogadó címét dolgozza fel.) A *`szabálykészlet cím`* parancscsal bármely (esetleg több) tesztelni kívánt szabálykészlet címfeldolgozásába tekinthetünk be.

Mindenekelőtt ellenőrizzük, hogy a *sendmail* képes a leveleket a rendszer helyi felhasználóihoz kézbesíteni. Szeretnénk, hogy e tesztekben a program az összes címet a gép *local* levelezőjére írja át:

```
# /usr/sbin/sendmail -bt -Cvstout.cf
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /parse issac
Cracked address = $g
Parsing envelope recipient address
canonify          input: issac
Canonify2         input: issac
Canonify2       returns: issac
canonify        returns: issac
parse             input: issac
Parse0            input: issac
Parse0          returns: issac
ParseLocal        input: issac
ParseLocal      returns: issac
Parse1            input: issac
Parse1          returns: $# local $: issac
parse           returns: $# local $: issac
2                 input: issac
2               returns: issac
EnvToL            input: issac
EnvToL          returns: issac
final             input: issac
final           returns: issac
mailer local, user issac
```

A kimenetből láthatjuk, hogyan dolgozza fel a *sendmail* a rendszer *isaac* nevű felhasználójához címzett leveleket. Az egyes sorok a szabálykészleteknek átadott információt, illetve a szabálykészlet feldolgozásának kimenetét mutatják. A *sendmail* programmal tudattuk, hogy a kézbesítési címet szeretnénk feldolgozni. Az utolsó sor igazolja, hogy a rendszer az *isaac* felhasználónak címzett leveleket valóban a *local* levelezőhöz irányítja.

Most ellenőrizzük az *isaac@vstout.vbrew.com* SMTP címünkre címzett leveleket! Az eredménynek meg kell egyeznie az előző példával.

```
> /parse isaac@vstout.vbrew.com
Cracked address = $g
Parsing envelope recipient address
canonify          input: isaac   @ vstout . vbrew . com
Canonify2         input: isaac < @ vstout . vbrew . com >
Canonify2       returns: isaac < @ vstout . vbrew . com . >
canonify        returns: isaac < @ vstout . vbrew . com . >
parse             input: isaac < @ vstout . vbrew . com . >
Parse0            input: isaac < @ vstout . vbrew . com . >
Parse0          returns: isaac < @ vstout . vbrew . com . >
ParseLocal        input: isaac < @ vstout . vbrew . com . >
ParseLocal      returns: isaac < @ vstout . vbrew . com . >
Parse1            input: isaac < @ vstout . vbrew . com . >
Parse1          returns: $# local $: isaac
parse           returns: $# local $: isaac
2                 input: isaac
2               returns: isaac
EnvToL            input: isaac
EnvToL          returns: isaac
final             input: isaac
final           returns: isaac
mailer local, user isaac
```

Ezután vizsgáljuk meg, hogy a program a *vbrew.com* tartomány más gazdagépeihez címzett leveleket közvetlenül az adott gazdagépre továbbítja az SMTP levelezővel:

```
> /parse issac@vale.vbrew.com
Cracked address = $g
Parsing envelope recipient address
canonify          input: issac   @ vale . vbrew . com
Canonify2         input: issac < @ vale . vbrew . com >
Canonify2       returns: issac < @ vale . vbrew . com . >
canonify        returns: issac < @ vale . vbrew . com . >
parse             input: issac < @ vale . vbrew . com . >
Parse0            input: issac < @ vale . vbrew . com . >
Parse0          returns: issac < @ vale . vbrew . com . >
ParseLocal        input: issac < @ vale . vbrew . com . >
ParseLocal      returns: issac < @ vale . vbrew . com . >
Parse1            input: issac < @ vale . vbrew . com . >
MailerToTriple    input: < > issac < @ vale . vbrew . com . >
MailerToTriple  returns: issac < @ vale . vbrew . com . >
Parse1          returns: $# esmtp $@ vale . vbrew . com . $: issac <
                                 @ vale . vbrew . com . >
parse           returns: $# esmtp $@ vale . vbrew . com . $: issac <
                                 @ vale . vbrew . com . >
2                 input: issac < @ vale . vbrew . com . >
2               returns: issac < @ vale . vbrew . com . >
EnvToSMTP         input: issac < @ vale . vbrew . com . >
PseudoToReal      input: issac < @ vale . vbrew . com . >
PseudoToReal    returns: issac < @ vale . vbrew . com . >
```

```
MasqSMTP          input: issac < @ vale . vbrew . com . >
MasqSMTP        returns: issac < @ vale . vbrew . com . >
EnvToSMTP       returns: issac < @ vale . vbrew . com . >
final             input: issac < @ vale . vbrew . com . >
final           returns: issac   @ vale . vbrew . com
mailer esmtp, host vale.vbrew.com., user issac@vale.vbrew.com
```

Láthatjuk, hogy a teszt az üzenetet az alapértelmezett SMTP levelezőhöz (esmtp) irányítja, amely a *vale.vbrew.com* gazdagéphez küldi, azon belül pedig a gazdagép *isaac* felhasználójához.

Végül ellenőrizzük a *vstout.cf* konfigurációhoz létrehozott *genericstable* adatbázist! Ellenőrizzük az *alana* felhasználónév leképzését a *vbrew.com* tartomány mindhárom *alana* nevű felhasználója esetén. A következő tesztben nyomon követhetjük, hogyan képezi le a *genericstable* a nevek egyes változatait:

```
# sendmail -bt
ADDRESS TEST MODE (ruleset 3 NOT automatically invoked)
Enter <ruleset> <address>
> /tryflags HS
> /try esmtp alana@vpils.vbrew.com
Trying header sender address alana@vpils.vbrew.com for mailer esmtp
canonify          input: alana @ vpils . vbrew . com
Canonify2         input: alana < @ vpils . vbrew . com >
Canonify2       returns: alana < @ vpils . vbrew . com . >
canonify        returns: alana < @ vpils . vbrew . com . >
1                 input: alana < @ vpils . vbrew . com . >
1               returns: alana < @ vpils . vbrew . com . >
HdrFromSMTP       input: alana < @ vpils . vbrew . com . >
PseudoToReal      input: alana < @ vpils . vbrew . com . >
PseudoToReal    returns: alana < @ vpils . vbrew . com . >
MasqSMTP          input: alana < @ vpils . vbrew . com . >
MasqSMTP        returns: alana < @ vpils . vbrew . com . >
MasqHdr           input: alana < @ vpils . vbrew . com . >
canonify          input: alana . darling @ vbrew . com
Canonify2         input: alana . darling < @ vbrew . com >
Canonify2       returns: alana . darling < @ vbrew . com . >
canonify        returns: alana . darling < @ vbrew . com . >
MasqHdr         returns: alana . darling < @ vbrew . com . >
HdrFromSMTP     returns: alana . darling < @ vbrew . com . >
final             input: alana . darling < @ vbrew . com . >
final           returns: alana . darling @ vbrew . com
Rcode = 0, addr = alana.darling@vbrew.com
> /try esmtp alana@vale.vbrew.com
Trying header sender address alana@vale.vbrew.com for mailer esmtp
canonify          input: alana @ vale . vbrew . com
Canonify2         input: alana < @ vale . vbrew . com >
Canonify2       returns: alana < @ vale . vbrew . com . >
canonify        returns: alana < @ vale . vbrew . com . >
1                 input: alana < @ vale . vbrew . com . >
1               returns: alana < @ vale . vbrew . com . >
HdrFromSMTP       input: alana < @ vale . vbrew . com . >
PseudoToReal      input: alana < @ vale . vbrew . com . >
```

```
PseudoToReal    returns: alana < @ vale . vbrew . com . >
MasqSMTP          input: alana < @ vale . vbrew . com . >
MasqSMTP        returns: alana < @ vale . vbrew . com . >
MasqHdr           input: alana < @ vale . vbrew . com . >
canonify          input: alana . henson @ vbrew . com
Canonify2         input: alana . henson < @ vbrew . com >
Canonify2       returns: alana . henson < @ vbrew . com . >
canonify        returns: alana . henson < @ vbrew . com . >
MasqHdr         returns: alana . henson < @ vbrew . com . >
HdrFromSMTP     returns: alana . henson < @ vbrew . com . >
final             input: alana . henson < @ vbrew . com . >
final           returns: alana . henson @ vbrew . com
Rcode = 0, addr = alana.henson@vbrew.com
> /try esmtp alana@foobar.vbrew.com
Trying header sender address alana@foobar.vbrew.com for mailer esmtp
canonify          input: alana @ foobar . vbrew . com
Canonify2         input: alana < @ foobar . vbrew . com >
Canonify2       returns: alana < @ foobar . vbrew . com . >
canonify        returns: alana < @ foobar . vbrew . com . >
1                 input: alana < @ foobar . vbrew . com . >
1               returns: alana < @ foobar . vbrew . com . >
HdrFromSMTP       input: alana < @ foobar . vbrew . com . >
PseudoToReal      input: alana < @ foobar . vbrew . com . >
PseudoToReal    returns: alana < @ foobar . vbrew . com . >
MasqSMTP          input: alana < @ foobar . vbrew . com . >
MasqSMTP        returns: alana < @ foobar . vbrew . com . >
MasqHdr           input: alana < @ foobar . vbrew . com . >
canonify          input: alana . smiley @ vbrew . com
Canonify2         input: alana . smiley < @ vbrew . com >
Canonify2       returns: alana . smiley < @ vbrew . com . >
canonify        returns: alana . smiley < @ vbrew . com . >
MasqHdr         returns: alana . smiley < @ vbrew . com . >
HdrFromSMTP     returns: alana . smiley < @ vbrew . com . >
final             input: alana . smiley < @ vbrew . com . >
final           returns: alana . smiley @ vbrew . com
Rcode = 0, addr = alana.smiley@vbrew.com
> /quit
```

A teszt `/tryflags` utasítása lehetővé teszi, hogy megadjuk: a fejléc küldőjének címét (HS), a fejléc fogadójának címét (HR), a boríték küldőjének címét (ES) vagy a boríték fogadójának címét (ER) kívánjuk-e feldolgozni. Esetünkben azt szeretnénk látni, hogyan történik a fejlécet küldő címének újraírása. A `/try` paranccsal megadhatjuk, hogy mely levelezőhöz szeretnénk a címet újraírni, ezután pedig magát a címet, amelyet újra szeretnénk írni.

Ez a tesztünk is sikeres volt! A *genericstable* tesztek Alana Darling, Alana Henson és Alana Smiley esetében is működtek.

A sendmail futtatása

A *sendmail* daemon két módon futtatható. Az egyik mód, hogy az *inetd* daemon futtatja; a másik – gyakrabban használt – mód, hogy önálló daemonként futtatjuk. A levelező

programok emellett gyakran mint felhasználói parancsot hívják meg a *sendmail* programot, hogy a helyben előállított leveleket kézbesítésre fogadja.

Ha a *sendmail* programot önálló üzemmódban kívánjuk futtatni, helyezzük indító szkriptbe a *sendmail* parancsot, és a rendszer indításakor automatikusan elindul. A szintaxis általában:

```
/usr/sbin/sendmail -bd -q10m
```

A `-bd` argumentum hatására a *sendmail* daemonként fut, elágazik és a háttérben folytatja futását. A `-q10m` argumentum hatására a *sendmail* minden tíz percben ellenőrzi a várólistát. A várólista ellenőrzésére más időtartamot is megadhatunk.

Ahhoz, hogy a *sendmail* programot az *inetd* hálózati daemonból futtathassuk, a következőhöz hasonló bejegyzésre volna szükségünk:

```
smtp  stream  tcp nowait  nobody  /usr/sbin/sendmail -bs
```

A `-bs` argumentum utasítja a *sendmail* programot, hogy a szabványos be- és kimeneten az SMTP protokollt alkalmazza – az *inetd* daemonhoz ugyanis ez szükséges.

Amikor a *sendmail* programot ily módon hívjuk meg, feldolgozza a várólista átvitelre váró leveleit. Amikor a *sendmail-t* az *inetd* daemonból futtatjuk, létre kell hoznunk egy cron feladatot is, hogy adott időközönként lefuttassa a *runq* utasítást az üzenet-várólista könyvtár (mail spool) időközönkénti kiszolgálására. A megfelelő cron táblázat bejegyzése így nézhet ki:

```
# A mail spool futtatása 15 percenként
0,15,30,45     *     *     *     *     /usr/bin/runq
```

A legtöbb telepítésben a *sendmail* 15 percenként dolgozza fel a várólistát, ahogy *crontab* példánk is mutatja. A példa a *runq* parancsot alkalmazza. A *runq* parancs rendszerint egy szimbolikus csatolás a *sendmail* futtatható állományhoz, és a

```
# sendmail -q
```

kényelmesebb formája.

Tippek és trükkök

Sok mindent tehetünk annak érdekében, hogy a *sendmail* kiszolgáló fenntartását hatékonyabbá tegyük. A *sendmail* csomagban számos karbantartó eszközt találunk; nézzük meg a fontosabbakat!

Az üzenet-várólista kezelése

A levelek a */var/spool/mqueue* könyvtárban várakoznak, mielőtt elküldenénk azokat. A könyvtár neve üzenet-várólista könyvtár (mail spool). A *sendmail* program *mailq* pa-

rancsa lehetővé teszi, hogy a várakozó levélüzeneteket és állapotukat rendezett módon megjelenítsük. Az */usr/bin/mailq* parancs szimbolikus csatolás a *sendmail* végrehajtható állományhoz, és ugyanúgy működik, mint a következő utasítás:

```
# sendmail -bp
```

A *mailq* parancs kimenetében megjelenik az üzenet azonosítója (ID), az időpont, amikor a várólistára került, a küldő és egy üzenet, amely az aktuális állapotról számol be. A következő példában egy levélüzenetet látunk, amely egy hiba miatt megakadt a sorban:

```
$ mailq
                Mail Queue (1 request)
-Q-ID- -Size- --Q-Time--- -------Sender/Recipient-------
RAA00275      124 Wed Dec  9 17:47 root
                (host map: lookup (tao.linux.org.au): deferred)
                                    terry@tao.linux.org.au
```

Az üzenet továbbra is a várakozó levelek között van, mert a rendeltetési gazdagép IP címének meghatározása nem sikerült.

Ha azt szeretnénk, hogy a *sendmail* azonnal feldolgozza a várólistát, adjuk ki az */usr/bin/runq* parancsot. A várólista feldolgozására a háttérben kerül sor. A *runq* parancsnak nincs kimenete, ellenben egy *mailq* paranccsal ellenőrizhetjük, hogy a várólista most már üres.

A várólista feldolgozása távoli gazdagépen

Ha ideiglenes betárcsázó internetkapcsolattal és állandó IP címmel rendelkezünk és egy MX gazdagép gyűjti leveleinket, amíg nem vagyunk a hálózatra kapcsolódva, érdemes tudnunk, hogyan vehetjük rá az MX gazdagépet a várólista feldolgozására röviddel azután, hogy létrehoztuk a kapcsolatot.

A *sendmail* terjesztésben találunk egy rövid *perl* programot, amellyel ezt könnyen megvalósíthatjuk, ha a levelező gazdagépek támogatják. Az *etrn* szkript gyakorlatilag ugyanazt teszi a távoli gazdagépen, mint a *runq* parancs a helyi kiszolgálón. Ha meghívjuk a parancsot:

```
# etrn vstout.vbrew.com
```

a **vstout.vbrew.com** gazdagép feldolgozza a helyi gépünkre szánt várakozó leveleket. A parancsot rendszerint a PPP indítószkriptben helyezzük el, így a hálózati kapcsolat létrejötte után hamarosan lefut.

Levelezési statisztika

A *sendmail* adatokat gyűjt a levelezési forgalom nagyságáról, illetve bizonyos adatokat a gazdagépekről, amelyekre leveleket kézbesített. Az információ megjelenítésére két parancs szolgál: a *mailstats* és a *hoststat*.

A mailstats utasítás

A *mailstats* utasítás a *sendmail* által feldolgozott levelek mennyiségéről kínál adatokat. Először az adatgyűjtés megkezdésének ideje jelenik meg, amelyet egy táblázat követ. A táblázatban egy sor tartozik az egyes beállított levelezőkhöz, és egy sor összegzi a teljes levélforgalmat. Minden sorban nyolc mező található, ezeket a 12.5. táblázatban foglaltuk össze.

12.5. táblázat. A mailstats statisztika mezői

Mező	*Jelentés*
`M`	A levelező (átviteli protokoll) száma.
`Msgsfr`	A levelezőtől fogadott üzenetek száma.
`bytes_from`	A levelezőtől fogadott levelek mérete Kbájtban.
`Msgsto`	A levelezőhöz küldött üzenetek száma.
`bytes_to`	A levelezőhöz küldött levelek mérete Kbájtban.
`Msgsreg`	Az elutasított üzenetek száma.
`Msgsdis`	Az elvetett üzenetek száma.
`Mailer`	A levelező neve.

A 12.7. példában a *mailstats* parancs egy kimenetét láthatjuk.

12.7. példa. A mailstats parancs mintakimenete

```
# /usr/sbin/mailstats
Statistics from Sun Dec 20 22:47:02 1998
 M   msgsfr  bytes_from   msgsto    bytes_to  msgsrej msgsdis  Mailer
 0        0          0K       19        515K        0       0  prog
 3       33        545K        0          0K        0       0  local
 5       88        972K      139       1018K        0       0  esmtp
=  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =  =
 T      121       1517K      158       1533K        0       0
```

Adatgyűjtés csak akkor történik, ha a *sendmail.cf* állományban a `StatusFile` opció be van kapcsolva és a státusállomány létezik. A `StatusFile` opciót az általános Linux konfiguráció definiálja, ezért az ebből létrehozott *vstout.cf* állományunk tartalmazza:

```
$ grep StatusFile vstout.cf
O StatusFile=/etc/mail/statistics
```

Az adatgyűjtést újraindíthatjuk, ha a statisztikát tartalmazó állomány hosszát nullára állítjuk és újraindítjuk a *sendmail* programot.

A hoststat utasítás

A *hoststat* parancs azoknak a gazdagépeknek az állapotáról jelenít meg adatokat, amelyekre a *sendmail* megkísérelt levelet kézbesíteni. A *hoststat* paranccsal egyenértékű a következő utasítás:

```
sendmail -bh
```

A kimenetben minden gazdagép külön sorban jelenik meg. A kimenet tartalmazza az utolsó kézbesítési kísérlet óta eltelt időt és az egyes kísérleteknél kapott állapotüzeneteket is.

A gazdagépek állapotát csak akkor tárolja állandóan a rendszer, ha a `HostStatusDirectory` opcióban megadunk egy könyvtárt az állapotinformációk tárolására. Az opciót az *m4* makró konfiguráció definiálja a `confHOST_STATUS_DIRECTORY` direktívával. Alapértelmezés szerint ez az útvonal nem létezik az állapotkönyvtárhoz, ezért nem marad fenn információ a gazdagépek állapotáról.

A 12.8. példa a *hoststat* parancs kimenetét mutatja. Vegyük észre, hogy a legtöbb eredmény sikeres kézbesítésről számol be. Az **earthlink.net** esetében azonban a kézbesítés sikertelen volt. Az állapotüzenet esetenként segítségünkre lehet a hiba okának meghatározásában. Jelen esetben a kapcsolat átlépte az időkorlátot, valószínűleg azért, mert a gazdagép nem volt bekapcsolva vagy a kézbesítési kísérlet idején nem volt elérhető.

12.8. példa. A hoststat parancs mintakimenete

```
# hoststat
--------- Hostname ------ How long ago ------Results------
mail.telstra.com.au                    04:05:41 250 Message accepted for
scooter.eye-net.com.au                81+08:32:42 250 OK id=0zTGai-0008S9-0
yarrina.connect.com.a                 53+10:46:03 250 LAA09163 Message acce
happy.optus.com.au                    55+03:34:40 250 Mail accepted
mail.zip.com.au                        04:05:33 250 RAA23904 Message acce
kwanon.research.canon.com.au          44+04:39:10 250 ok 911542267 qp 21186
linux.org.au                          83+10:04:11 250 IAA31139 Message acce
albert.aapra.org.au                    00:00:12 250 VAA21968 Message acce
field.medicine.adelaide.edu.au        53+10:46:03 250 ok 910742814 qp 721
copper.fuller.net                     65+12:38:00 250 OAA14470 Message acce
amsat.org                             5+06:49:21 250 UAA07526 Message acce
mail.acm.org                          53+10:46:17 250 TAA25012 Message acce
extmail.bigpond.com                   11+04:06:20 250 ok
earthlink.net                         45+05:41:09 Deferred: Connection time
```

A *purgestat* utasítással törölhetjük a gazdagépekről gyűjtött adatokat. A parancs megfelel a következő parancsnak:

```
# sendmail -bH
```

A statisztika mindaddig egyre nő, amíg nem töröljük. Érdemes a *purgestat* parancsot bizonyos időközönként lefuttatni, így könnyebben kereshetjük meg a legújabb bejegyzéseket, különösen akkor, ha hálózati helyünk forgalmas. A parancsot elhelyezhetjük egy *crontab* állományban, így automatikusan futtathatjuk, de alkalmanként magunk is meghívhatjuk.

További ismeretek

A *sendmail* összetett témakör – túlságosan is összetett ahhoz, hogy egyetlen fejezetben foglalkozzunk vele. A fejezet célja az volt, hogy segítségünkre legyen az első lépések megtételében és az egyszerű kiszolgálóbeállításában. Ha azonban összetett konfigurációval rendelkezünk, vagy a fejlett tulajdonságokat is használni szeretnénk, további ismeretekre lesz szükségünk. Az induláshoz néhány forrás bemutatásával szeretnénk segítséget nyújtani.

- A *sendmaild* terjesztésben több kiváló *README* állományt találhatunk. Indulásnak érdemes a terjesztés telepítésekor a legfelső szintű könyvtárban létrejött *README* állományt elolvasnunk. Az állományban találunk egy listát a további információs állományokról, például a *sendmail/README* és a *cf/README* állományról, amelyek alapvető ismereteket tartalmaznak. (A *sendmail* konfigurációs nyelvével foglalkozó *cf/README* állomány az interneten is elérhető, a *http://www.sendmail.org/m4/readme.html* címen.)
- A *sendmail Installation and Operations Guide* kiváló forrás, amelyet különböző formátumokban is megtalálhatunk a *sendmail* forráskódú terjesztésében: *doc/op/op.me*, illetve *doc/op/op.ps*.
- A *sendmail* webhely hasznos írásokat és on-line dokumentációt kínál. A *Compiling Sendmail* dokumentáció ideális példa. A leírás elolvasható a *http://www.sendmail.org/-compiling.html* címen.
- A *sendmail* oldalon megtalálhatjuk az elérhető *sendmail* könyvek listáját a *http://www.sendmail.org/books.html* címen.
- Részt vehetünk formális *sendmail* képzésen is. Egypár képzési lehetőségről a *http://www.sendmail.org/classes.html* címen tájékozódhatunk.

A források bőséges ismereteket kínálnak, többet, mint amire valaha is szükségünk lesz a *sendmail* programhoz. Fedezzük fel őket!

13

Az IPv6 hálózatok beállítása

Az IPv4 lehetőségei napról napra szűkülnek. Egyes jelzések szerint az internet felhasználóinak száma szerte a világon 2005-ben több mint egymilliárd. Figyelembe véve, hogy a felhasználók jelentős része rendelkezik mobiltelefonnal, egy otthoni számítógéppel és valószínűleg a munkahelyén is egy számítógéppel, az elérhető IP címtér kritikusan leszűkül. Kína nemrégiben minden diákja számára IP címet igényelt, ami összesen mintegy 300 millió címet jelent. Az ilyen teljesíthetetlen kérelmek erre a hiányra világítanak rá. Amikor az IANA eredetileg elkezdte kiadni a címtereket, az internet kicsi és kevéssé ismert kutatói hálózat volt csupán. Kevés igény mutatkozott a címek iránt, és az A osztályú címteret szabadon ki lehetett jelölni. Amikor azonban az internet jelentősége és mérete növekedésnek indult, az elérhető címek száma csökkenni kezdett, így egy új IP cím beszerzése nehezebbé és költségesebbé vált. A NAT és a CIDR erre a hiányra próbál választ adni. A NAT önálló megoldás, amely lehetővé teszi, hogy egy hálózati hely összes felhasználója egyetlen IP címen keresztül kapcsolódjon az internetre. A CIDR a hálózati címblokkok hatékonyabb felosztását biztosítja. Mindkét megoldás hagy azonban kívánnivalót maga után.

Az olyan új elektronikus eszközök megjelenésével, mint például a PDA-k és a mobiltelefonok, amelyek mindegyike saját IP címet igényel, a NAT címblokkok máris nem tűnnek olyan nagynak.

Az IP címek hiányát felismerő kutatók újratervezték az IPv4 protokollt, hogy 128 bites címteret támogasson. A választott 128 bites címtér 340 trillió lehetséges címet tesz lehetővé, és az exponenciális növekedés remélhetőleg megfelelő címzési lehetőséget biztosít majd a közeli (és távoli) jövőre. Valójában ez a szám olyan nagy, hogy a föld minden egyes lakosára egymilliárd cím jut.

Az IPv6 nem csak a címtérlogisztika egyes kérdéseire ad választ, hanem bizonyos konfigurációs és biztonsági kérdésekre is megoldást jelent. Ebben a részben megnézzük, milyen lehetőségekkel rendelkezünk jelenleg a Linux és az IPv6 területén.

Az IPv4 és az ideiglenes megoldások

Kezdetben az IANA a kérelmezőknek egy teljes A osztályú hálózati teret jelölt ki, vagyis a kérelmezők 16,7 millió címet kaptak – több mint elegendőt. Felismerve a hibát, az IANA megkezdte a B osztályú hálózatok kijelölését – ami ismét csak a szükségesnél jóval több címet jelentett az átlagos kérelmezők esetén. Az internet növekedésével hamarosan vi-

lágossá vált, hogy nincs értelme minden kérelmezőnek A vagy B osztályú hálózatot biztosítani. De még a későbbi megoldás, a C osztályú címek kiosztása is pazarlásnak bizonyult, hiszen a legtöbb vállalat nem igényel 254 IP címet. Miután az IANA nem vonhatta vissza a már kiosztott címtereket, a fennmaradó címtérrel kellett okosabban gazdálkodniuk. Az egyik megoldásuk a *Classless Inter-Domain Routing* (CIDR) volt.

A CIDR

A CIDR lehetővé teszi, hogy hálózati blokkokat osszunk ki a jól körülhatárolt A/B/C tartományokon kívül. Annak érdekében, hogy a már létező C osztályú hálózati blokkokból a legtöbbet hozzuk ki, a CIDR lehetővé teszi a rendszergazdáknak, hogy címterüket kisebb egységekre bontsák, amelyeket azután önálló hálózatokhoz jelölhetnek ki. Így könnyebben juttathatunk IP címet több felhasználónak, mert a címteret igény szerint osztjuk fel, és nem kell előre meghatározott méretű terekben gondolkodnunk. Egy C osztályú alhálózattal rendelkező szolgáltató például úgy dönthet, hogy a hálózatot 32 önálló hálózatra osztja fel, a határok jelzését pedig hálózati címekkel és alhálózati maszkokkal oldja meg. Lássunk most egy minta CIDR jelölést!

```
10.10.0.64/29
```

A példában a /29 jelzi az alhálózati maszkot, ami azt jelenti, hogy a cím első 29 bitje az alhálózat. Ezt a **255.255.255.248** jelöléssel is felírhatnánk; a hálózat ezzel hat használható címhez jut.

Noha a CIDR valóban gyors és egyszerű megoldás, valójában nem hoz létre több IP címet, és más hátrányai is vannak. Először is romlik a hatékonyság, mert minden kijelölt hálózat egy üzenetszóró IP címet és egy hálózat IP címet is igényel. Vagyis ha a szolgáltató egy C osztályú hálózatot 32 önálló hálózatra bont, a hálózati és üzenetszóró IP címekre összesen 64 címet kell áldoznunk. Másodszor, az összetett CIDR hálózatokban könnyebb konfigurációs hibát ejteni. Egy hibás alhálózati maszkot használó útválasztó például az általa kiszolgált kis hálózatok teljes leállását okozhatja.

A NAT

A *hálózati címfordítás* (Network Address Translation, NAT) hozott bizonyos eredményeket az IP címtér kérdésben, és nélküle ma már jócskán kifutottunk volna a használható IP címtérből. A NAT segítségével számos gép osztozhat ugyanazon az IP címen. Az eljárás a NAT eszköz mögötti gépek számára bizonyos elkülönülést és biztonságot is jelent, hiszen nehezebb az egyes gépek azonosítása. A NAT-nak hátrányai is vannak – elsősorban az, hogy egyes régebbi protokollok nem képesek az átirányítást kezelni.

A megoldás: IPv6

Az IPv6 a csökkenő IP tér problémájának megoldására született. A jövőlátó tervezők a 128 bites címtér mellett döntöttek, így összesen:

340 282 366 920 938 463 463 374 607 431 768 211 456 ($3{,}4 \times 10^{38}$) címet biztosítottak.

A méretek érzékeltetésére, ez 655 570 793 348 866 943 898 599 ($6{,}5 \times 10^{23}$) címet jelent a Föld felületének minden egyes négyzetméterére számítva. Az IPv4 jelenlegi 32 bites címterével szemben ez a változás határozott növekedést jelent.

Az IPv6 címzés

Az IPv4 és IPv6 közötti különbségek közül először a címek írási módja ötlik a szemünkbe. Egy tipikus IPv6 cím így fest:

```
fe80:0010:0000:0000:0000:0000:0000:0001
```

Minden IP cím nyolc darab, egyenként négy hexadecimális értéket tartalmazó sorozatból áll. A címek önmagukban hosszúak és nehezen kezelhetők, ezért egy rövidítési eljárást fejlesztettek ki hozzájuk. Az egyetlen, csak nullát tartalmazó karakterláncot helyettesíthetjük dupla kettősponttal. Az előbbi példa tehát rövidített formában:

```
fe80:0010::1
```

Az összevonást egy címben azonban csak egyszer végezhetjük el, ellenkező esetben nem tudnánk megmondani, mit távolítottunk el. Vegyük példaként a következő IP címet, amelyben több, egymástól független nulla karakterlánc is van:

```
2001:0000:0000:a080:0000:0000:0000:0001
```

Mivel csak egyetlen nulla sorozatot távolíthatunk el, az IP címet nem rövidíthetjük le így:

```
2001::a080::1
```

Általában a leghosszabb karakterláncot távolítjuk el. Ha a példa leghosszabb sorozatát helyettesítjük, a rövidített IP cím:

```
2001:0000:0000:a080::1
```

Az IPv6 protokollban több különböző típusú cím létezik, ezek a specifikációban elérhető különféle funkciókat definiálják:

Helyi kapcsolati címek (link-local address)
: E címek beállítása az IPv6 protokollkezelő inicializálásakor automatikusan történik, a hálózati kártyánk MAC címe alapján. Az ilyen címeket általában „csak ügyfél" címekként tartjuk számon, és nem képesek kiszolgáló futtatására vagy bejövő kapcsolatok figyelésére. A helyi kapcsolati címek mindig az *FE8x, FE9x, FEAx* vagy *FEBx* sorozattal kezdődnek, ahol az x helyére bármilyen hexadecimális számjegy kerülhet.

Helyi hálózati helycímek (site-local address)
Noha az eredeti specifikáció része és különféle dokumentációkban találkozhatunk is vele, a helyi hálózati helycímek használata nem javasolt, és már nem tekintjük az IPv6 részének.

Globális unicast cím (Global unicast address)
Az ilyen típusú IP cím alkalmas az internetes útválasztásra, és minden gépen ez mutat kifelé. A címet jelenleg a *2xxx* vagy *3xxx* jelölés azonosítja, noha ezt a jövőben szükség szerint bővíthetik.

Az IPv6 előnyei

Bár az IPv6 legnyilvánvalóbb előnye, hogy drámaian megnöveli a címteret, számos más fontos előnye is van. Az IPv6 például egyes területeken teljesítményjavulást hozott. A csomagok feldolgozása hatékonyabb, mert a csomagfejléc opció mezőinek feldolgozása csak akkor történik meg, ha valóban tartalmaznak opciókat. További teljesítményjavulást jelent a csomagfragmentáció kiiktatása. A beépített IPSec a biztonság szempontjából hozott javulást. Mivel a titkosítás és a „letagadhatatlanság" (non-repudiation) a protokoll része, megvalósítása egységesebb. A szolgáltatás minősége (Quality of Service, QoS) szintén olyan előny, amely az IPv6 fejlődésével együtt javul. A funkció lehetővé teszi, hogy a hálózati rendszergazdák a hálózati forgalmat rangsorolják. Ez különösen azokon a hálózatokon fontos, amelyeken olyan szolgáltatások futnak, mint például a hangátvitel az IP protokollon (Voice over IP), mert itt még a kis hálózati zavarok is rontják a szolgáltatás színvonalát. Végül a címek automatikus beállításának fejlődésével könnyebbé válik a menet közbeni hálózatkezelés. További előnyökre számíthatunk a kutatások és a megvalósítás előrehaladtával. Várhatóan a mobil IP eljárások fejlődésétől is hasonlókat remélhetünk, aminek révén lehetővé válhatna, hogy minden eszköz megtarthassa az IP címét, függetlenül aktuális hálózati kapcsolatától.

Az IPv6 beállítása

Az IPv6 támogatása sokat fejlődött az elmúlt időszakban, és ma már szinte minden Linux terjesztés támogatja. Az utóbbi néhány kibocsátásban már a 2.4 kernelnek is a része volt, és a 2.6 kernel is tartalmazza.

A kernel és a rendszer beállítása

Amióta a kernel forrás tartalmazza, az IPv6 támogatás beállítása jóval könnyebbé vált a Linuxon. Nincs már szükség patch programokra, bár bizonyos eszközöket telepítenünk kell; ezekkel nemsokára foglalkozunk.

Ha az IPv6 támogatás nincs a kernelbe építve, esetleg megtalálhatjuk modulként lefordítva. A modul meglétét gyorsan ellenőrizhetjük a következő paranccsal:

```
vlager# modprobe ipv6
vlager#
```

Ha nem kapunk választ, a modul valószínűleg sikeresen betöltődött. A támogatás bekapcsolt állapotát többféleképpen is megvizsgálhatjuk. Érdemes a /proc könyvtárat ellenőriznünk:

```
vlager# ls -l /proc/net/if_inet6
-r-r-r-    1  root    root        0  Jul 1 12:12  /proc/net/if_inet6
```

Ha az *ifconfig* kompatibilis verzióját birtokoljuk, az ellenőrzést azzal is végezhetjük:

```
vlager# ifconfig eth0 |grep inet6
       inet6 addr:  fe80::200:ef99:f3df:32ae/10 Scope:Link
```

Amennyiben a tesztek sikertelenek, valószínűleg ismét le kell fordítanunk a kernelt a támogatás bekapcsolásához. Az IPv6 kernel konfigurációs opciója a *.config* állományban:

```
CONFIG_IPV6=m
```

Ha a „make menuconfig" segítségével végezzük a beállítást, az IPv6 bekapcsolását a 2.4 kernel esetén a „Network Options" részben találjuk. A 2.6 kernel konfigurációban ez a „Network Support/Network Options" részben található. A támogatást belefordíthatjuk a kernelbe, vagy megépíthetjük modulként. Ha az utóbbit tesszük, az interfész konfigurálása előtt ne felejtsük el kiadni a *modprobe* parancsot.

Az interfész beállítása

Azért, hogy az interfészt beállíthassuk az IPv6-hoz, az elterjedt hálózati segédprogramokból az IPv6-os verziókat kell használnunk. Mivel a legtöbb Linux terjesztés napjainkban gyárilag támogatja az IPv6 protokollt, valószínűleg már telepítettük is ezeket az eszközöket. Ha egy régebbi verziót korszerűsítünk vagy Linux rendszerünket az alapoktól magunk építjük, esetleg telepítenünk kell a *net-tools* csomagot, amelyet az interneten több helyen is megtalálunk. A legújabb verzióra a Google vagy FreshMeat keresőben a „net-tools" kifejezésre keresve bukkanhatunk rá.

A verziók kompatibilitásának ellenőrzéséhez végezzünk egy gyors próbát az *ifconfig* vagy a *netstat* segítségével. A próba így nézhet ki:

```
vlager# /sbin/netstat grep inet6
```

Mielőtt tovább mennénk, győződjünk meg róla, hogy rendelkezünk az IPv6 hálózati kapcsolatait vizsgáló különféle eszközökkel, például a *ping6*, a *traceroute6* és a *tracepath6* programokkal. Ezeket az *iputils* csomagban találjuk, és a legtöbb korszerű terjesztésben már szintén telepítettek. Az útvonalunkon ellenőrizhetjük, hogy az eszközök elérhetők-e, és ha nem, el kell végeznünk a telepítésüket. Amennyiben le kell töltenünk őket, elérhetőek a szerző *ftp://ftp.inr.ac.ru/ip-routing* címén.

Ha minden rendben ment, az interfész beállítása automatikusan megtörténik a MAC cím alapján. Vizsgáljuk meg az *ifconfig* paranccsal:

```
vlager# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:07:E9:DF:32:AE
          inet addr:10.10.10.19  Bcast:10.10.10.255  Mask:255.255.255.0
          inet6 addr: fe80::207:e9ff:fedf:32ae/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2272821 errors:0 dropped:0 overruns:0 frame:73
          TX packets:478473 errors:0 dropped:0 overruns:0 carrier:0
          collisions:4033 txqueuelen:100
          RX bytes:516238958 (492.3 Mb)  TX bytes:54220361 (51.7 Mb)
          Interrupt:20 Base address:0x2000
 vlager#
```

A kimenet harmadik sora a vlager helyi kapcsolati címét mutatja. Könnyen megismerjük, mert minden `fe80` cím helyi kapcsolat típusú IP cím. Ha biztonsági megfontolásból nem szívesen használjuk MAC címünket fő IP címként, vagy éppenséggel egy kiszolgálót állítunk be és egyszerűbb IP címet szeretnénk, a következő példa alapján megadhatjuk saját IP címünket:

```
vlager# ifconfig eth0 inet6 add 2001:02A0::1/64
vlager#
```

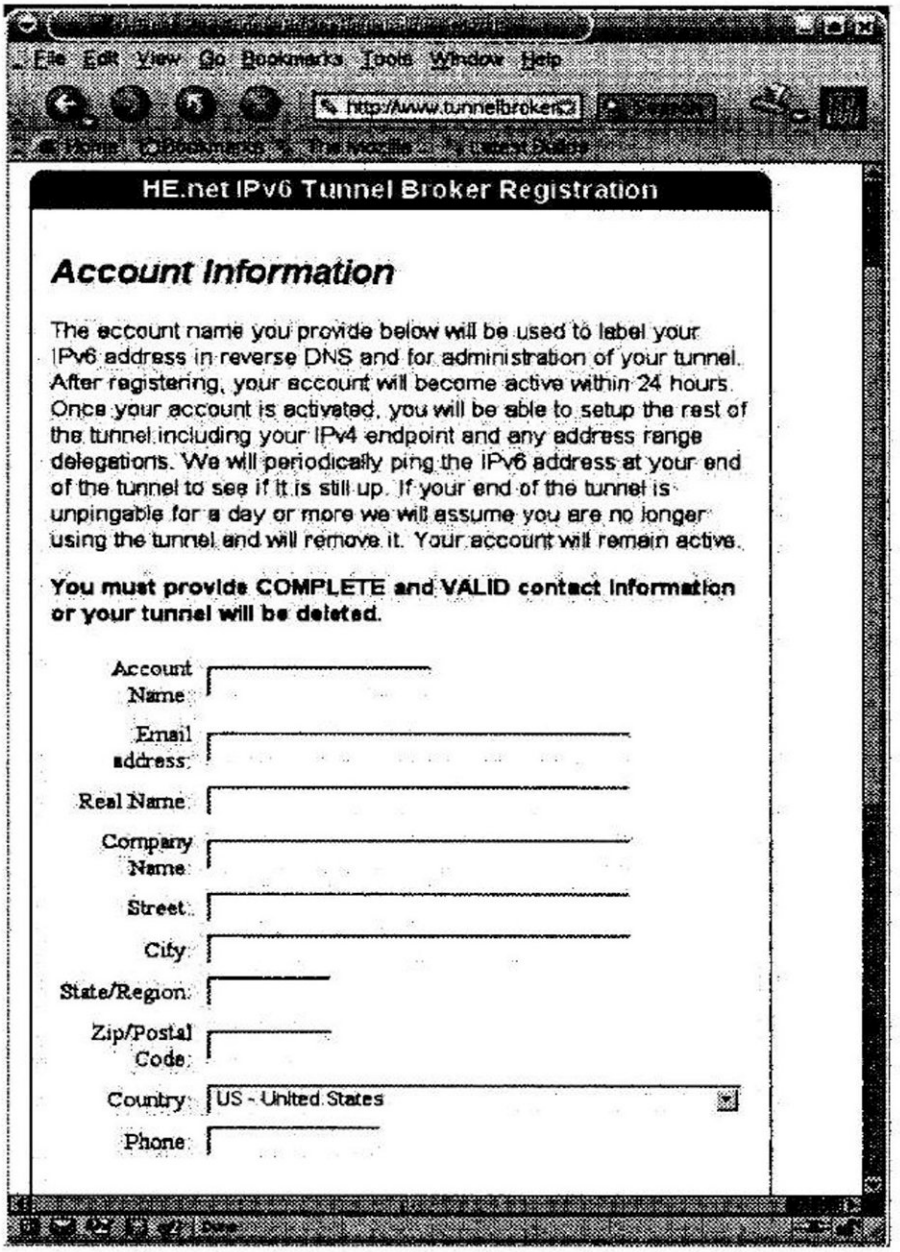
Ennél a pontnál azonban még nem feltétlenül rendelkezünk egy globális típusú címmel, mint mi az előbbi példában. Ezért IP címünk lehet helyi kapcsolat vagy helyi hálózati hely típusú cím. Az internetes útválasztást nem igénylő forgalomhoz ezek tökéletesen megfelelnek, ha azonban szándékunkban áll a külvilághoz is kapcsolódni, akkor rendelkeznünk kell közvetlen kapcsolattal az IPv6 gerincvezetékhez, vagy IPv6 csatornával egy *csatornaközvetítőn* keresztül. Ez utóbbiról szól a következő rész.

IPv6 kapcsolat létrehozása csatornaközvetítőn keresztül

Ahhoz, hogy az IPv6 szines világához csatlakozzunk, szükségünk van egy útvonalra, amelyen keresztül csatlakozhatunk. A legtöbb felhasználó számára jelenleg egy csatorna jelenti az egyetlen lehetőséget, mivel kevés hálózati hely rendelkezik közvetlen IPv6 kapcsolattal. Ha az IPv6 forgalmat közvetlenül IPv4 hálózatokon keresztül próbáljuk meg irányítani, nem jutunk messzire, mert valószínűleg már a következő útválasztó sem tudja majd, mihez is kezdjen a furcsának tűnő forgalommal. A legtöbb felhasználó számára a legegyszerűbb útvonal egy csatorna létrehozása egy csatornaközvetítőn (tunnel broker) keresztül. Az interneten számos különféle közvetítő létezik, akik saját IPv6 címtartományt biztosíthatnak számunkra. Az egyik leggyorsabb és legismertebb csatornaközvetítő a Hurricane Electric (13.1. ábra), amely automatizált IPv6 csatornakérő formulával rendelkezik. Csupán azt várják el tőlünk, hogy legyen egy „pingelhető" IPv4 címünk, amely folyamatosan kapcsolódik az internetre. Ezt az IPv4 címet tekinti majd a csatornaforrásának.

A csatorna kiépítése

Mihelyt megkaptuk IPv6 címterünket, hozzákezdhetünk a csatorna kiépítéséhez. Ehhez a kiegészítő Link becsomagoló (encapsulation) interfészre lesz szükségünk, amely az IPv6 modul telepítésével jön létre.



13.1. ábra. Csatorna beszerzése a he.net oldalról

A csatorna kiépítéséhez be kell állítanunk a sit0 és sit1 interfészeket. A sit interfészek virtuális adaptereknek tekintjük, mert nem képviselnek közvetlenül hardvert a rendszerünkben. A szoftver szempontjából azonban a kezelésük majdnem pontosan úgy történik, mint bármely más interfészé. A sit interfészekre adatokat irányítunk és az útválasztásban felhasználjuk őket. A sit virtuális interfészek lehetővé teszik, hogy IPv4 címeinket IPv6 címekre képezzük le, majd létrehozzunk a gépünkön egy IPv6 interfészt. Először bekapcsoljuk a sit0 interfészt és hozzárendeljük az IPv4 címünket. Például a csatorna közvetítő IPv4 végpontja a **10.10.0.8**, a **vlager** IPv6 csatornájának végponti IP címe pedig a `2001:FEFE:0F00::4B`.

```
vlager# ifconfig sit0 up
vlager# ifconfig sit0 inet6 tunnel ::10.10.0.8
```

Ebben a lépésben bekapcsoltuk a sit0 interfészt és a csatornaközvetítő IPv4 címéhez kötöttük. A következő lépésben IPv6 címünket a sit1 interfészhez rendeljük. Ezt a következő utasításokkal tehetjük meg:

```
vlager# ifconfig sit1 up
vlager# ifconfig sit1 inet6 add 2001:FEFE:0F00::4B/127
```

A csatorna most már működőképes. A teszteléséhez azonban IPv6 forgalmat kell a sit1 interfészre irányítanunk. Ezt könnyen megtehetjük a *route* paranccsal:

```
vlager$ route -A inet6 add ::/0 dev sit1
```

A parancs utasítja az operációs rendszert, hogy minden IPv6 forgalmat a sit1 eszközre irányítson. Miután az útvonal a helyére került, ellenőrizhetjük az IPv6 működését. A kapcsolatot a *ping6* paranccsal vizsgálhatjuk. A példában az újonnan létrehozott csatorna távoli végének IPv6 címét pingeljük:

```
vlager# ping6 2001:470:1f00:ffff::3a
PING 2001:470:1f00:ffff::3a(2001:470:1f00:ffff::3a) 56 data bytes
64 bytes from 2001:470:1f00:ffff::3a: icmp_seq=1 ttl=64 time=26.2 ms
64 bytes from 2001:470:1f00:ffff::3a: icmp_seq=2 ttl=64 time=102 ms
64 bytes from 2001:470:1f00:ffff::3a: icmp_seq=3 ttl=64 time=143 ms
64 bytes from 2001:470:1f00:ffff::3a: icmp_seq=4 ttl=64 time=130 ms
Ctrl-c
-- 2001:470:1f00:ffff::3a ping statistics --
4 packets transmitted, 4 received, 0% packet loss, time 3013ms
rtt min/avg/max/mdev = 26.295/100.590/143.019/45.339 ms
vlager#
```

Jelenleg van egy rendszerünk, amelyet a nyilvános IPv6 hálózatra állítottunk. A teljes IPv6 világot láthatjuk magunk körül, mint ahogy az is láthat bennünket. Nagyon fontos, hogy tudjuk, pontosan mely szolgáltatások figyelnek a hálózatra, és hogy a szolgáltatások hibáit hibajavító programokkal megszüntessük, mert különben nem lehet használni azokat. Az IPv6 netfilter támogatás fejlesztés alatt áll, egyelőre nem biztos, hogy megbízhatunk a stabilitásában.

Az IPv6 kezelésére képes alkalmazások

Jelenleg már szép számmal léteznek olyan, az IPv4 hálózatokon használt alkalmazások, amelyek képesek az IPv6 kezelésére is. A legnépszerűbbek között találjuk az Apache kiszolgálót és az OpenSSH-t. Ebben a részben arról szólunk, hogyan állíthatjuk be az IPv6 támogatást ezekben az alkalmazásokban.

Az Apache webkiszolgáló

Noha az Apache v1.3 verzió széles körben elterjedt a stabilitása miatt, a forráskód módosítása nélkül nem támogatja az IPv6-ot. Ha mindenképpen a v1.3 verziót szeretnénk alkalmazni IPv6 támogatással, találhatunk hozzá patch programokat, de ezekhez nem

kapunk támogatást, és valószínűleg nem is tesztelték őket. Rengeteg vita folyt arról, hogy a stabil v1.3 hivatalosan támogassa-e az IPv6-ot, de a fejlesztők végül úgy döntöttek, hogy nem nyúlnak a v1.3 verzióhoz, és az Apache IPv6 támogatásának fejlesztésére a 2.0 verziót használják. Az Apache 2.0 és az újabb verziók módosítás nélkül támogatják az IPv6-ot, ezért a következő részben ezekkel foglalkozunk.

Az Apache v2.0.x beállítása az IPv6 támogatásához

Az Apache beállítása az IPv6 támogatásához meglehetősen egyszerű feladat. A megépítéshez nincs szükség különleges opciókra, és akár forrásból, akár RPM-ből építhetjük. A fordításkor beállítható egyik opció, az *-enable-v4-mapped* azonban érdekes lehet az IPv6 felhasználók számára. Az előre megépített csomagokban általában ez az alapértelmezett beállítás, és lehetővé teszi, hogy egy általános `Listen` direktívával alakítsunk ki vonalat:

```
Listen 80
```

A direktíva a webkiszolgáló-folyamatot az összes elérhető IP címhez köti. Az IPv6 rendszerek adminisztrátorai a szolgáltatást rossz hatékonyságúnak és elégtelennek találhatják, mert szükségtelen foglalatokat (socket) nyit meg a nagyszámú alapértelmezett IPv6 címekhez. Éppen ezért a fordításnál megadhatjuk a *-disable-v4-mapped* direktívát, explicit módon beállítva a figyelő interfészeket. A szolgáltatás kikapcsolásával ismét lehetővé válik, hogy az interfészek valamennyi portot figyeljék, ekkor azonban külön be kell állítanunk őket.

Amikor a kiszolgálót kívánságunknak megfelelően lefordítottuk és telepítettük, a figyelő bekapcsolásához módosítanunk kell a konfigurációs állományt. Ez a lépés nagyon hasonlít az Apache IPv4 konfigurálásához. Azért, hogy a **vlager** IPv6 IP webfigyelőjét bekapcsoljuk, a következőképpen változtatjuk meg az *apache.conf* állományt:

```
Listen [fec0:ffff::2]:80
```

Amikor az Apache-t elindítjuk, az megnyit egy figyelőt a megadott IP 80-as portján. A hatást a *netstat* paranccsal ellenőrizhetjük:

```
vlager# netstat -aunt
Active Internet connections (servers and established)
Proto Recv-Q Send-Q  Local Address          Foreign Address    State
tcp        0      0  10.10.0.4:22          0.0.0.0:*          LISTEN
tcp        0      0  fec0:ffff::2:80       :::*               LISTEN

vlager#
```

A táblázat második bejegyzése az IPv6 apache figyelő, amely pontosan olyan formában jelenik meg, mint az IPv4 címek.

Ha azt szeretnénk, hogy az Apache kiszolgáló minden elérhető IPv6 címre figyeljen, akkor némileg eltérő konfigurációs opcióval választunk:

```
Listen [::]:80
```

A direktíva nagyon hasonlít az IPv4 esetében ugyanezen célra használt 0.0.0.0 címre. Ha a 80-as porton kívül más portokra is szeretnénk figyelőket kapcsolni, cseréljük ki a már létező portszámot, vagy hozzunk létre további `Listen` vonalakat. Az Apache kiszolgálóról részletesebb leírást olvashatunk a 14. fejezetben.

Az OpenSSH

Az OpenSSH projekt a kezdetektől kompatibilis az IPv6-tal, és a program ilyen jellegű támogatását ma már érettnek minősíthetjük. Beállítása is egyszerű. A fordításkor nincs szükség kiegészítő opciók megadására, így bináris csomagból bonyodalmak nélkül telepíthetjük.

Ebben a részben felételezzük, hogy már rendelkezünk az IPv4 alatt működő OpenSSH-val és tudjuk, hová telepítettük a konfigurációs állományokat. Esetünkben a konfigurációs állományok az */etc/ssh* útvonalon vannak. Egy IPv6 figyelő megadásához a következő sort helyezzük el az *sshd_config* állományban:

```
ListenAddress fec0:ffff::2
Port 1022
```

Amikor az OpenSSH-t a -6 parancssori opcióval újraindítjuk, máris figyeli az IPv6 címünket az 1022-es porton.

Az OpenSSH ügyfélből éppen ilyen egyszerűen érhetjük el az IPv6 gazdagépeket, csupán a -6 parancssori opciót kell megadnunk:

```
othermachine$ ssh -6 fec0:ffff::2 -p 1022
bob@fec0:ffff::2's password:
bob@vlager $
```

Az OpenSSH beállítása az IPv6 használatára csupán ennyiből áll. Most rendelkezünk egy kiszolgálóval, amelyen egy OpenSSH IPv6 figyelő fut, és használhatjuk az *ssh* parancsot, hogy az IPv6 hálózat más gazdagépeivel kapcsolatba lépjünk. Ha hibát tapasztalunk, olvassuk el a következő, *Hibakeresés* című részt.

Hibakeresés

Mivel az IPv6 hálózatkezelés gyakran még felderítetlen terület, könnyen előfordul, hogy valami meghibásodik. Az IPv6 kezelésében az egyik leggyakoribb hiba a címek jelöléséből fakad. Hibát okozhat, hogy az alrészek elválasztására nem pontot, hanem kettőspontot használunk, mert sok rendszergazda keze szinte magától a pont felé indul. A jelöléssel kapcsolatos másik probléma akkor merül fel, amikor a címeket rövidítve írjuk. Ahogy már említettük, a kihagyott nulla sorozatok helyére két kettőspontot teszünk. Ha elfeledkezünk a kettőspontokról, a gép hibaüzenetet küld a hiányos IP címről. Lássunk néhány példát hibás IPv6 jelölésekre:

```
fe80.ffff.0207.3bfe.0ddd.bbfe.02
3ffe:0001:fefe:5
2001:fdff::0901::1
```

Súlyosabb hiba esetén, ha egyáltalán nem látjuk az IPv6 tárt, nézzük át a kernel konfigurációját. Ha az IPv6 támogatást közvetlenül a kernelbe fordítottuk, ellenőrizzük a rendszernaplót, hogy kapunk-e hibaüzenetet a betöltéskor. Sikeres IPv6 telepítés esetén a rendszer indításakor ilyen üzenetet kapunk:

```
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 4096 buckets, 32Kbytes
TCP: Hash tables configured (established 32768 bind 65536)
NET4 Unix domain sockets 1.0/SMP for Linux NET4.0.
IPv6 v0.8 for NET4.0
IPv6 over IPv4 tunneling driver
```

A kód a hálózati tár inicializálásáról számol be, az utolsó két sor pedig kimondottan az IPv6-ra vonatkozik. Ha ez a két sor nem jelenik meg, érdemes fontolóra vennünk az IPv6 megépítését modulként.

Ha az IPv6 támogatást modulként valósítottuk meg, ellenőrizzük, hogy beállítottuk az automatikus betöltést. Erre annyi módszer létezik, ahány Linux terjesztés, ezért a részleteket a terjesztésünk dokumentációjában olvashatjuk el.

Ha a modul szabályosan betöltődik, az *lsmod* parancs hatására meg kell jelennie:

```
vlager# lsmod |grep ipv6
Module                  Size  Used by    Not tainted
ipv6                  162132  -1
```

A modul betöltésekor a következő üzenetnek kell megjelennie a rendszernaplóban:

```
Jul  7 16:13:43 deathstar kernel: IPv6 v0.8 for NET4.0
Jul  7 16:13:43 deathstar kernel: IPv6 over IPv4 tunneling driver
```

Ha biztosan tudjuk, hogy az IPv6 tár telepítése hibátlan és a helyi LAN-on forgalmat is képesek vagyunk bonyolítani, de nem tudunk az IPv6 csatornán adatokat küldeni, ellenőrizzük az IPv4 kapcsolatot. Ehhez mindenekelőtt vizsgáljuk meg alaposan a sit0 konfigurációban megadott IPv4 csatornacímeket. Ha a beállítások helyesek, teszteljük az IPv4 távoli végpontját. Ha nem tudunk IPv4 forgalmat irányítani a csatorna IP végpontjára, akkor IPv6 forgalmat sem leszünk képesek küldeni.

A kapcsolati problémák további oka lehet egy helytelenül beállított tűzfal. Ha úgy döntöttünk, hogy az IPv6-hoz a Netfiltert használjuk, ellenőrizzük a tűzfalszabályokat: próbáljunk meg adatokat küldeni egyszer úgy, hogy a szabályok be vannak kapcsolva, egyszer pedig úgy, hogy nincsenek. Előfordulhat, hogy a Netfilter hibás és az IPv6 esetén nem működik megfelelően, így egyes konfigurációk nem működőképesek.

14

Az Apache webkiszolgáló beállítása

A Linux rendszerek egyik legelterjedtebb szoftvercsomagja jelenleg az Apache webkiszolgáló. A fejlesztők kis csoportjaként 1995-ben induló Apache Szoftver Alapítványt 1999-ben jegyezték be az Apache HTTP kiszolgálók fejlesztésére és támogatására. Az Apache HTTP kiszolgáló, amely ma már több mint 25 millió internetes webkiszolgálón fut, elsősorban rugalmasságáról és kiváló teljesítményéről ismert. A fejezetben megvizsgáljuk egy Apache HTTP kiszolgáló megépítésének és beállításának alapjait, valamint szólunk néhány olyan beállítási lehetőségről, amelyekkel működését biztonságosabbá tehetjük és teljesítményét javíthatjuk. A fejezetben az Apache 1.3-as verziójával foglalkozunk, amely jelenleg a legelterjedtebb és legjobban támogatott verzió.

Az Apache HTTPD kiszolgáló – bevezetés

Az Apache önmagában csak egyszerű webkiszolgáló. Amikor tervezték, célját a weboldalak kiszolgálásában határozták meg. Noha a kereskedelemben kapható egyes webkiszolgálókat számos különböző szolgáltatással ruháznak fel, az ilyen összetett termékek általában jelentős számú biztonsági rést tartalmaznak. Az Apache HTTPD kiszolgáló egyszerűsége és modularitása biztonságosabb terméket eredményez, és a beszámolók – különösen ha az Apache-t más webkiszolgálókkal hasonlítjuk össze – stabil és megbízható szoftvert tárnak elénk.

Ez persze nem jelenti azt, hogy az Apache kiszolgálók nem képesek dinamikus tartalmat biztosítani a felhasználóknak. Rengeteg Apache modul áll rendelkezésünkre, ezeket integrálva szinte végtelen számú új szolgáltatást hozhatunk létre. A kiegészítő termékek, például a PHP és a *mod_perl* segítségével hatékony webes alkalmazásokat készíthetünk, és dinamikus webtartalmat állíthatunk elő. A fejezetben azonban inkább a tulajdonképpeni Apache kiszolgáló beállítására fordítjuk a figyelmünket. Megnézzük, hogyan építhetünk és hozhatunk működőképes állapotba egy Apache HTTPD webkiszolgálót; milyen opciókat kell megadnunk ahhoz, hogy stabil és biztonságos webkiszolgálóhoz jussunk.

Az Apache beállítása és megépítése

Ha Linux terjesztésünkben egyelőre nincs jelen az Apache, legegyszerűbben a számos Apache tükröoldal egyikéről szerezhetjük be. Lelőhelylistát az Apache Szoftver Alapít-

vány fő oldalán, a *http://www.apache.org* címen találunk. Jelenleg az Apache HTTPD két fő verziója érhető el, az 1.3 és a 2.0. A 2.0 új verzió, jelenleg is aktív fejlesztés alatt áll, és új szolgáltatásokat vonultat föl, ellenben valószínűbb, hogy belső hibákat és támadási felületeket tartalmaz. A fejezetben az 1.3-as verzió legfrissebb kiadásával foglalkozunk, mert ez már bizonyította megbízhatóságát és stabilitását. Ennek ellenére a két változat számos konfigurációs opciója hasonló.

A szoftver beszerzése és fordítása

Az Apache programot tetszés szerint beszerezhetjük forrás vagy csomag formátumban. Amennyiben csomagból telepítjük, nem lesz részünk az első beállítás rugalmasságában, amit a forrásból való építés nyújtani tud. A csomagokban a gyakoribb opciókat általában előre beépítik a bináris állományokba. Ha különleges szolgáltatásokat vagy opciókat szeretnénk használni, illetve ha a kiszolgáló legegyszerűbb változatát szeretnénk megépíteni, érdemes az építést forrásból végezni.

Az Apache forrásból történő megépítése hasonló a más Linux forrásokéhoz, és a „configure-make-make install" eljárást követi. Az Apache számos beállítási lehetőségét a forrás konfigurálásakor kell megadnunk. Ilyenkor adjuk meg például azokat a modulokat, amelyeket szeretnénk megépíteni, illetve amelyeket szeretnénk kiiktatni. A modulok nagyszerű lehetőséget biztosítanak a webkiszolgáló szolgáltatásainak telepítéséhez és eltávolításához, azonkívül széles körű tulajdonságkészletet kínálnak – például a teljesítmény, a hitelesítés és a biztonság területén. A 14.1. táblázat az Apache dokumentációban leírt elérhető modulok közül ismertet egypárat.

14.1. táblázat. Apache modulok

Típus	*Alapértelmezés szerint ki- vagy bekapcsolt*	*Feladat*
A környezet létrehozása		
mod_env	bekapcsolt	A környezeti változók beállítása a CGI/SSI szkriptek számára.
mod_setenvif	bekapcsolt	A környezeti változók beállítása a HTTP fejlécek alapján.
mod_unique_id	kikapcsolt	Egyedi azonosítók előállítása a lekérésekhez.
A tartalom típusának kérdéseit		
mod_mime	bekapcsolt	A tartalom típusának/kódolásának meghatározása (beállított).
mod_mime_magic	kikapcsolt	A tartalom típusának/kódolásának meghatározása (automatikus).
mod_negotiation	bekapcsolt	Tartalomválasztás a HTTP Accept* fejlécek alapján.
URL leképzés		
mod_alias	bekapcsolt	Egyszerű URL fordítás és átirányítás.
mod_rewrite	kikapcsolt	Fejlett URL fordítás és átirányítás.

14.1. táblázat. Apache modulok (folytatás)

Típus	*Alapértelmezés szerint ki- vagy bekapcsolt*	*Feladat*
mod_userdir	bekapcsolt	Az erőforráskönyvtárak kiválasztása felhasználónév. szerint.
mod_spelling	kikapcsolt	Elgépelt URL címek helyesbítése.
Könyvtárkezelés		
mod_dir	bekapcsolt	Könyvtár és könyvtári alapértelmezett állomány kezelése.
mod_autoindex	bekapcsolt	Automatizált könyvtárindex állomány készítés.
Hozzáférés-szabályozás		
mod_access	bekapcsolt	Hozzáférés-szabályozás (felhasználó, gazdagép és hálózat alapján).
mod_auth	bekapcsolt	HTTP elemi hitelesítés (felhasználó és jelszó alapján).
mod_auth_dbm	kikapcsolt	HTTP elemi hitelesítés a UNIX NDBM állományokkal.
mod_auth_db	kikapcsolt	HTTP elemi hitelesítés a Berkeley DB állományokkal.
mod_auth_anon	kikapcsolt	HTTP elemi hitelesítés a névtelen felhasználók számára.
mod_digest	kikapcsolt	Digest hitelesítés.
HTTP válasz		
mod_headers	kikapcsolt	Tetszőleges HTTP válaszfejlécek (beállított).
mod_cern_meta	kikapcsolt	Tetszőleges HTTP válaszfejlécek (CERN-szerű állományok).
mod_expires	kikapcsolt	Expires HTTP válaszok.
mod_asis	bekapcsolt	Nyers HTTP válaszok.
Szkriptkezelés		
mod_include	bekapcsolt	Server Side Includes (SSI) támogatás.
mod_cgi	bekapcsolt	Common Gateway Interface (CGI) támogatás.
mod_actions	bekapcsolt	CGI szkripteket belső „kezelőként" jelöl ki.
Belső tartalomkezelők		
mod_status	bekapcsolt	Tartalomkezelő; a kiszolgáló futásidejű állapotáról nyújt információt.
mod_info	kikapcsolt	Tartalomkezelő; átfogó információt ad a kiszolgáló beállításairól.
A lekérések naplózása		
mod_log_config	bekapcsolt	A lekérések testre szabható naplózása.
mod_log_agent	kikapcsolt	Különleges HTTP User-Agent naplózás (ellenjavallt).
mod_log_referer	kikapcsolt	Különleges HTTP Referrer naplózás (ellenjavallt).
mod_usertrack	kikapcsolt	A felhasználó linkekre történő kattintásainak naplózása.

14.1. táblázat. Apache modulok (folytatás)

Típus	*Alapértelmezés szerint ki- vagy bekapcsolt*	*Feladat*
Egyéb		
mod_imapn	bekapcsolt	Kiszolgálóoldali képtérkép támogatás.
mod_proxy	kikapcsolt	Gyorsítótáras proxy modul (HTTP, HTTPS, FTP).
mod_so	kikapcsolt	Dynamic Shared Object (DSO) betöltése a kiszolgáló indulásakor.
Kísérleti		
mod_mmap_static	kikapcsolt	A gyakran kiszolgált oldalak gyorsítótárba helyezése az *mmap()* segítségével.
Fejlesztői		
mod_example	kikapcsolt	Apache API bemutató (csak fejlesztőknek).

Miután eldöntöttük, mely opciókat kívánjuk használni, a következőképpen adhatjuk hozzá ezeket a konfigurációs szkripthez. Egy modul bekapcsolásához írjuk be:

```
vlager# ./configure -enable-module=module_name
```

Egy alapértelmezett modul kikapcsolásához írjuk be:

```
vlager# ./configure -disable-module=module_name
```

Az alapértelmezett modulokat csak akkor kapcsoljuk be vagy ki, ha pontosan tudjuk, mi a feladatuk. Egyes modulok be- vagy kikapcsolása hátrányosan befolyásolhatja a teljesítményt, illetve a biztonságot. A modulokról részletesen olvashatunk az Apache webhelyén.

A konfigurációt követően a teljes csomagot lefordítjuk. Mint sok más Linux program esetében, a fordítást *make* paranccsal végezzük. Ezután a *make install* paranccsal telepíthetjük az Apache szoftvert a konfigurálásnál megadott, a `-prefix=` opció által kijelölt könyvtárba.

A konfigurációs állomány beállításai

Miután az Apache szoftvert a választott könyvtárba telepítettük, hozzákezdhetünk a kiszolgáló beállításához. Az Apache kiszolgáló korábbi változatai több konfigurációs állományt is használtak, ma már azonban csak a *httpd.conf* állomány szükséges. Ennek ellenére jól jöhet, ha több konfigurációs állományt használunk (így például egyszerűbbé tehetjük a frissítéseket). Az `include` opció lehetővé teszi, hogy a fő *httpd.conf* állományból további konfigurációs állományokat olvassunk be.

Az Apache alapértelmezett konfigurációs állománnyal érkezik, amelyben a leggyakoribb opciókat előre beállították. Ha szeretnénk a kiszolgálónkat minél előbb elindítani,

megtehetjük az alapértelmezett konfigurációs állománnyal. Annak ellenére, hogy a konfiguráció működőképes, ezt a lépést sok rendszergazda nem tartja elfogadhatónak. Akik így vélekednek és a konfiguráció finombeállítása mellett döntenek, rendszerint a kiszolgáló IP címét és a port információt állítják be elsőként.

A címek és portok kötése

A `Listen` és a `BindAddress` lehet az első két opció, amelyeket megváltoztatunk:

```
# Listen: az Apache-t specifikus IP címekhez és/vagy portokhoz köthetjük
# az alapértelmezett értékek helyett. Lásd még a <VirtualHost>
# direktívát.
#
#Listen 3000
Listen 172.16.0.4:80
```

A konfiguráció módosítását követően az Apache kiszolgáló kizárólag a kijelölt interfészen és porton figyel. A `BindAddress` opcióval megadhatjuk azt az IP címet is, amelyhez a szervert kötni szeretnénk. Az előbbi példától eltérően az opcióval csak az IP címet határozzuk meg, a portot nem:

```
# BindAddress: Segítségével virtuális gazdagépeket támogathatunk.
# A direktíva megmondja a kiszolgálónak, hogy melyik IP címre figyeljen.
# Tartalmazhatja a "*" kifejezést, egy IP címet vagy egy teljes
# minősített internet tartománynevet.
# Lásd még a <VirtualHost> és a Listen direktívákat.
#
BindAddress 172.16.0.4
```

A naplózás és az útvonal-beállítási lehetőségei

Az Apache megépítésekor megadhattuk a telepítési könyvtárt. Amennyiben megadtuk, a telepítő automatikusan beállította a kiszolgálógyökér dokumentumai és a naplóállományok útvonalát. Ha ezeket szeretnénk megváltoztatni, a következő opciókkal tehetjük:

`ServerRoot`
: A kiszolgáló fő konfigurációs és naplóállományainak helye.

`DocumentRoot`
: A HTML dokumentumok, illetve más webtartalom helye.

Alapértelmezés szerint az Apache a fő kiszolgálógyökér útvonala alá helyezi a naplóállományokat. Ha rendszerünkön ettől eltérő helyen gyűjtjük a naplókat és szeretnénk megváltoztatni a naplóállományok útvonalát, a következő opciókat szükséges módosítanunk.

`CustomLog`
A hozzáférési naplóállomány helye.

`ErrorLog`
A hibanapló helye.

Az Apache naplózásának beállításakor más hasznos lehetőségeket is megadhatunk:

`HostNameLookups`
Meghatározza, hogy az Apache a naplózott IP címekhez kikeresse-e a neveket. Érdemes a beállítást kikapcsolva hagyni, mert a naplózás lassul, ha a kiszolgáló az összes nevet megpróbálja kikeresni.

`LogLevel`
Meghatározza, hogy az Apache mennyi információt írjon a naplóállományokba. Alapértelmezés szerint az értéke a `warn`, de beállítható a `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` és `emerg` értékekre is. A növekvő naplózási szintek egyre kevesebb információt jelentenek.

`LogFormat`
Az opcióval a rendszergazdák megválaszthatják a naplók formátumát. A dátum, idő, IP cím és a hasonló elemek tetszőleges formába rendezhetők. Rendszerint nem változtatjuk meg az alapértelmezett értéket.

Kiszolgálóazonosító karakterláncok

Alapértelmezésben az Apache nagyon barátságos és a kérelmező felhasználóknak rengeteget elárul magáról, többek között a verzióját, a virtuális gazdagépnevét, a rendszergazda nevét stb. A biztonságra törekvő rendszergazdáknak ajánlatos kikapcsolni ezt a szolgáltatást, hiszen a támadók az információt felhasználva jóval gyorsabban kiismerhetik gépünket.

Noha az eljárás nem jelent teljes biztonságot webhelyünk számára, feltarthatja az automatizált letapogató eszközöket használó esetleges támadókat. A következő két konfigurációs opcióval korlátozhatjuk a kiszolgáló által kiadott adatok mennyiségét:

`ServerSignature`
Ha az opciót bekapcsoljuk, a kiszolgáló által előállított oldalakhoz fűz egy sort a kiszolgáló, amelyben a verzióról olvasható információ.

`ServerTokens`
Ha az opciónak a `Prod` értéket adjuk, az Apache soha nem fedi fel a verziószámát.

A teljesítmény beállítása

Az egyes hálózati helyek különböző teljesítménybeállításokat igényelnek. Az Apache alapértelmezett beállításai sok hálózati helynek a megfelelő teljesítményt biztosítják.

A forgalmasabb helyeken azonban érdemes módosítanunk a konfigurációt, hogy növeljük a teljesítményt. A következő beállításokkal a kiszolgáló teljesítményét hangolhatjuk. Az Apache teljesítményének állításáról további ismereteket az Apache Szoftver Alapítvány webhelyén találunk.

Timeout
Ennyi másodperc után következik be a fogadási és küldési kérések időtúllépése.

KeepAlive
Kapcsoljuk be az opciót, ha állandó kapcsolatokat szeretnénk. Értékei az on (be) és az off (ki).

MaxKeepAliveRequests
Az opcióval megadhatjuk, hány állandó kapcsolati kérést engedélyezzen a kiszolgáló állandó kapcsolatok esetén. A magasabb értékek javíthatják a teljesítményt.

KeepAliveTimout
Ennyi másodpercig vár az Apache új kérésre az aktuálisan kapcsolt munkafolyamatból.

Min/MaxSpareServers
Ezekkel az opciókkal tartalék kiszolgálókat hozhatunk létre, amelyeket az Apache akkor használhat, amikor elfoglalt. A nagyobb hálózati helyeknek érdemes az alapértelmezett értékeknél nagyobbat megadni. Minden egyes tartalék kiszolgáló további memóriát igényel.

StartServers
Azt állítja be, hogy az Apache az első indulásakor hány kiszolgálót indít el.

MaxClients
A rendszergazda ezzel az opcióval korlátozhatja egy kiszolgálóügyfél munkafolyamatainak számát. Az Apache dokumentációja figyelmeztet, hogy az értékét ne állítsuk túl kicsire, mert hátrányosan befolyásolhatja az elérhetőséget.

Az Apache indítása és leállítása az apachectl segítségével

Amikor a kiszolgálót beállítottuk és készen állunk a futtatására, az Apache szoftverrel kapott *apachectl* eszközzel végezhetjük a kiszolgáló biztonságos indítását és kikapcsolását. Az *apachectl* eszköz rendelkezésre álló lehetőségei a következők:

`start`
Elindítja a szabványos HTTP kiszolgálót.

`startssl`
A normál kiszolgáló mellett az SSL kiszolgálókat is elindítja.

`stop`
Kikapcsolja az Apache kiszolgálót.

`restart`
HUP jelet küld a futó kiszolgálónak.

`fullstatus`
A webkiszolgáló állapotáról tesz részletes jelentést, de a *mod_status* modult igényli.

`status`
Az előbbi állapotjelző képernyő rövidebb változatát jeleníti meg. A *mod_status* modult igényli.

`graceful`
SIGUSR1 jelet küld az Apache kiszolgálónak.

`configtest`
Hibák után kutat a konfigurációs állományban.

Noha nem kötelező az Apache kiszolgálót az *apachectl* eszközzel indítanunk, mégis érdemes, és ez egyben az indítás legegyszerűbb módja. Az *apachectl* a kiszolgáló kikapcsolását is gyorsabban és hatékonyabban végzi.

A VirtualHost beállítási lehetőség

Az Apache egyik leghatékonyabb szolgáltatása az a képessége, hogy egyetlen gépen több webkiszolgálót is képes futtatni. Ehhez a VirtualHost szolgáltatást vesszük igénybe, amelyet a *httpd.conf* állományban érhetünk el. A virtuális gazdagépek két típusát állíthatjuk be: a nevesített virtuális gazdagépet és az IP virtuális gazdagépet. A nevesített virtuális gazdagép esetén több TLD-t is használhatunk egyetlen IP címen, az IP virtuális gazdagépek esetén viszont IP címenként egy virtuális gazdagép használható. Ebben a részben mindkettőre mutatunk példát, és megnézünk néhány gyakori konfigurációs lehetőséget is.

Az IP alapú virtuális gazdagépek

Azok számára, akik csak egyetlen hálózati helyet tartanak fenn, vagy több IP címmel is rendelkeznek a hálózati helyeik futtatásához, az IP alapú virtuális gazdagépek jelentik a legjobb konfigurációs választást. Tegyük fel, hogy a Virtuális Sörgyár úgy dönt, létrehoz egy hálózati helyet a Virtuális Borászat részére. Az új webhely létrehozásához szükséges, a *httpd.conf* állományhoz fűzött minimális konfigurációt mutatja be a következő példa.

```
Listen www.virtualvineyard.com:80
.
```

```
.
<VirtualHost www.virtualvineyard.com>
ServerAdmin webmaster@vbrew.com
DocumentRoot /home/www/virtualvineyard.com
ServerName www.virtualvineyard.com
ErrorLog /var/www/logs/vvineyard.error_log
TransferLog /var/www/logs/vvineyard.access_log
</VirtualHost>
```

Ellenőrizzük, hogy az */etc/hosts* állomány tartalmazza a *www.virtualvineyard.com* bejegyzést. Erre azért van szükség, mert az Apache kiszolgálónak indulásakor ki kell keresnie egy IP címet ehhez a tartományhoz. Rábízhatnánk ezt a DNS kiszolgálóra is, de ha a DNS kiszolgáló valamilyen okból nem érhető el, amikor a webkiszolgáló újraindul, a webkiszolgáló kudarcot vall. Egy másik megoldás, hogy a kiszolgálónk IP címét megadjuk a konfiguráció elején, a `<VirtualHost>` címkében. Első hallásra ez talán hatékonyabb módszernek tűnhet, de ha meg szeretnénk változtatni a webkiszolgáló IP címét, az Apache konfigurációs állományát is meg kellene változtatnunk.

A példa konfigurációs opciói mellett a fejezetben korábban tárgyalt opciók bármelyikét megadhatjuk a VirtualHost csoportban, így maximális rugalmasságot érhetünk el valamennyi önálló webkiszolgálónk esetén.

A név alapú virtuális gazdagépek

A név alapú virtuális gazdagépek beállítása nagyon hasonlít az előző példára azzal a különbséggel, hogy egyetlen IP címen több tartományt is létrehozhatunk. A szolgáltatásnak két hátránya van. Az egyik – és talán a legnagyobb hátulütője –, hogy az SSL csak egyetlen IP címmel használható. Ez nem az Apache esetében jelent gondot, hanem inkább az SSL esetén, amikor tanúsítványokat szeretnénk használni. A másik potenciális hátránya, hogy egyes régebbi, például a HTTP 1.1 specifikációt nem támogató webböngészőkkel nem működik. Ennek oka, hogy a név alapú virtuális gazdagépkezelés a ügyféltől várja el, hogy a HTTP kérés fejlécében informálja a kiszolgálót a meglátogatni kívánt webhelyről. Az elmúlt években kiadott böngészők szinte mindegyike megvalósítja a HTTP 1.1 előírást, így a legtöbb rendszergazda számára ez nem jelent gondot.

Visszatérve a minta konfigurációhoz, a fejezet korábbi példáját használjuk fel, ezúttal azonban a Virtuális Sörgyár csak egyetlen nyilvános IP címmel rendelkezik. Mindenekelőtt közöljük az Apache kiszolgálóval, hogy virtuális gazdagépkezelést szeretnénk kialakítani, majd a példában látható módon megadjuk a hálózati helyünk adatait:

```
NameVirtualHost 172.16.0.199
<VirtualHost 172.16.0.199>
ServerName www.vbrew.com
DocumentRoot /home/www/vbrew.com
</VirtualHost>
<VirtualHost 172.16.0.199>
ServerName www.virtualvineyard.com
DocumentRoot /home/www/vvineyard.com
</VirtualHost>
```

Az érthetőség kedvéért a kiegészítő opciókat elhagytuk, de szükség szerint a korábban ismertetett opciók bármelyikét megadhatjuk.

Az Apache és az OpenSSL

Miután beállítottuk és teszteltük Apache webkiszolgálónk konfigurációját, érdemes egy SSL oldalt beállítanunk. Az SSL használatára számos okunk lehet a web alapú e-mail ügyfelek védelmétől a biztonságos elektronikus kereskedelmi tranzakciók biztosításáig. Az Apache világában az SSL támogatására az Apache-SSL és a *mod_ssl* szolgál. Ebben a részben a régebbi és elterjedtebb *mod_ssl* modullal foglalkozunk.

Mint bármely más SSL alapú alkalmazásnál, tanúsítványokra van szükségünk. Az ügyfél és a kiszolgáló közötti bizalmi kapcsolat a tanúsítványokra épül. Ha üzleti célokra tartunk fenn webhelyet, érdemesebb egy harmadik fél, például a Verisign vagy a Thawte által aláírt tanúsítványt beszereznünk.* Mivel ezek a tanúsítványok költségesek, ha nem üzleti célú webhelyet tartunk fenn, készítsünk inkább saját tanúsítványt. A megoldás hátránya, hogy amikor az ügyfelek a webhelyünket felkeresik, hibaüzenetet kapnak, amely figyelmezteti őket, hogy tanúsítványunk nem megbízható, mert nem írta alá egy harmadik fél. Így át kell „kattingatniuk magukat" a hibaüzeneten, és el kell dönteniük, megbíznak-e a tanúsítványunkban. Ebben a fejezetben a saját tanúsítvány létrehozása mellett döntő rendszergazdák számára mutatunk be minta konfigurációkat; (megjegyezzük, hogy a *cacert.org* ingyenes tanúsítványokat kínál az önálló felhasználóknak).

Az SSL tanúsítvány előállítása

Az SSL munkafolyamatokhoz először egy tanúsítványt kell készítenünk. Mindenekelőtt győződjünk meg róla, hogy az OpenSSL telepítve van a rendszerünkön. A programot megtaláljuk forrás és bináris csomag formátumban is a *http://www.openssl.org* címen. Sok Linux terjesztés telepíti a csomagot, így előfordulhat, hogy nem is kell ezzel foglalkoznunk. Miután az OpenSSL-t telepítettük, illetve meggyőződtünk róla, hogy telepítve van, kezdjünk hozzá a szükséges SSL tanúsítvány létrehozásához.

Első lépésként készítsünk egy tanúsítvány-aláírási kérelmet. Ehhez meg kell adnunk egy ideiglenes PEM jelszót és egy pár adatot a webhelyünkről:

```
vlager# openssl req -config openssl.cnf -new -out vbrew.csr
Using configuration from openssl.cnf
Generating a 1024 bit RSA private key
A konfigurálás az openssl.cnf állomány alapján történik.
Az 1024 bites RSA saját kulcs létrehozása.
.....++++++
....++++++
writing new private key to 'privkey.pem'
az új saját kulcs beírása a 'privkey.pem' állományba
```

* Működnek tanúsítványt aláíró hazai cégek is, mint például a NetLock – *a lektor.*

```
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```
Adja meg a PEM jelszót:
A jelszó ellenőrzése - Adja meg a PEM jelszót:
```
----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```
A következőkben a tanúsítványkérelemhez szükséges adatokat kell megadnia.
A megadott információ az úgynevezett Megkülönböztetett Név (Distinguished Name, DN).
A számos mező közül egyeseket üresen hagyhat.
Más mezők rendelkeznek egy alapértelmezett értékkel,
ezeket a '.' beírásával hagyhatjuk üresen.
```
----
Country Name (2 letter code) [AU]:US
```
Országnév (kétbetűs kód)
```
State or Province Name (full name) [Some-State]:California
```
Az állam vagy tartomány neve (teljes név):
```
Locality Name (eg, city) [ ]:Berkeley
```
A helység neve (például város):
```
Organization Name (eg, company) [Internet Widgits Pty Ltd]:www.vbrew.com
```
A szervezet neve (például cég):
```
Organizational Unit Name (eg, section) [ ]:
```
A szervezeti egység neve (például részleg):
```
Common Name (eg, YOUR name) [ ]:www.vbrew.com
```
Az általánosan használt név (például az ÖN neve):
```
Email Address [ ]:webmaster@vbrew.com
```
E-mail cím:
```
Please enter the following 'extra' attributes
to be sent with your certificate request
```
Kérjük adja meg a következő 'kiegészítő' jellemzőket,
melyek a tanúsítványkérelemhez csatolva kerülnek továbbításra.
```
A challenge password [ ]:
```
Figyelemfelkeltő jelszó:
```
An optional company name [ ]:
```
Opcionális cégnév:

A következő lépésben eltávolítjuk a saját kulcs PEM jelszavát a tanúsítványról. Így lehetővé tesszük, hogy a kiszolgáló újrainduljon anélkül, hogy kérné a jelszót. A különösen óvatos rendszergazdák ezt a műveletet átléphetik, de ha a kiszolgáló összeomlik, manuálisan kell újraindítaniuk.

```
vlager # openssl rsa -in privkey.pem -out vbrew.key
read RSA key
Enter PEM pass phrase:
writing RSA key
```

Miután eltávolítottuk a jelszót, aláírjuk a tanúsítvány állományt. Ezt az x509 opcióval és az OpenSSL segítségével tehetjük meg:

```
apache ssl # openssl x509 -in vbrew.csr -out vbrew.cert -req -signkey
vbrew.key -days 365
Signature ok
subject=/C=US/ST=California/L=Berkeley/O=www.vbrew.com/CN=www.vbrew.com/Ema
il=webmaster@vbrew.com
Getting Private key
```

Amikor elkészültünk, megkezdhetjük a tanúsítvány használatát. Másoljuk a tanúsítvány állományokat az Apache könyvtárba, hogy a webkiszolgáló elérhesse azokat.

A mod_ssl modul fordítása az Apache kiszolgálóhoz

Ha az Apache kiszolgálót a fejezet korábbi példájához hasonlóan forrásból fordítottuk, a *mod_ssl* használatához egy patch programmal módosítjuk az Apache forrását, majd újrafordítjuk. Ha az Apache kiszolgálót a Linux terjesztésünk részére készült bináris csomagból fordítottuk, valószínű, hogy a modul már benne található. Azért, hogy lássuk, szükséges-e újrafordítanunk a kiszolgálót, a következő paranccsal ellenőrizzük, mely modulok találhatók meg benne:

```
vlager # /var/www/bin/httpd -l
Compiled-in modules:
  http_core.c
  mod_env.c
  mod_log_config.c
  mod_mime.c
  mod_negotiation.c
  mod_status.c
  mod_include.c
  mod_autoindex.c
  mod_dir.c
  mod_cgi.c
  mod_asis.c
  mod_imap.c
  mod_actions.c
  mod_userdir.c
  mod_alias.c
  mod_access.c
  mod_auth.c
  mod_setenvif.c
```

Esetünkben a *mod_ssl* modul nem található, ezért letöltjük és belefordítjuk Apache kiszolgálónkba. Szerencsére ez közel sem olyan bonyolult, mint amilyennek hangzik. A *mod_ssl* forrása letölthető a *http://www.modssl.org* címről. Csomagoljuk ki az OpenSSL forrásával együtt. Az egyszerűség kedvéért mindhárom forrásfát egyazon könyvtár alá helyeztük. Amikor mindent kicsomagoltunk, folytathatjuk a műveletet. Először a *mod_ssl* megépítésének beállításait adjuk meg:

```
vlager # ./configure --with-apache=../apache_1.3.28 --with-openssl=../openssl-0.9.6i
Configuring mod_ssl/2.8.15 for Apache/1.3.28
```

```
 + Apache location: ../apache_1.3.28 (Version 1.3.28)
 + Auxiliary patch tool: ./etc/patch/patch (local)
 + Applying packages to Apache source tree:
   o Extended API (EAPI)
   o Distribution Documents
   o SSL Module Source
   o SSL Support
   o SSL Configuration Additions
   o SSL Module Documentation
   o Addons
Done: source extension and patches successfully applied.
```

Feltéve, hogy az OpenSSL-t forrásból építettük és összhangban van az Apache forráskönyvtárral, így állíthatjuk be és építhetjük meg:

```
vlager # cd ../apache_1.3.28
vlager # SSL_BASE=../openssl-0.9.6i ./configure
                        -prefix=/var/www —enable-module=ssl
Configuring for Apache, Version 1.3.28
 + using installation path layout: Apache (config.layout)
Creating Makefile
Creating Configuration.apaci in src
Creating Makefile in src
 + configured for Linux platform
 + setting C pre-processor to gcc -E
 + using "tr [a-z] [A-Z]" to uppercase
 + checking for system header files
 + adding selected modules
    o ssl_module uses ConfigStart/End
      + SSL interface: mod_ssl/2.8.15
      + SSL interface build type: OBJ
      + SSL interface compatibility: enabled
      + SSL interface experimental code: disabled
      + SSL interface conservative code: disabled
      + SSL interface vendor extensions: disabled
      + SSL interface plugin: Built-in SDBM
      + SSL library path: /root/openssl-0.9.6i
      + SSL library version: OpenSSL 0.9.6i Feb 19 2003
      + SSL library type: source tree only (stand-alone)
 + enabling Extended API (EAPI)
 + using system Expat
 + checking sizeof various data types
 + doing sanity check on compiler and options
Creating Makefile in src/support
Creating Makefile in src/regex
Creating Makefile in src/os/unix
Creating Makefile in src/ap
Creating Makefile in src/main
Creating Makefile in src/modules/standard
Creating Makefile in src/modules/ssl
```

Miután elkészültünk a forrás beállításával, a *make install* utasítással újraépítjük az Apache kiszolgálót. Az előbbi *httpd -l* parancsot is lefuttathatjuk még egyszer, így ellenőrizve, hogy a *mod_ssl* modul valóban bekerült a kiszolgálóba.

A konfigurációs állomány módosításai

Csupán néhány apró változtatásra van szükségünk. Az Apache kiszolgálón az SSL-t a már bemutatott VirtualHost direktívával kapcsolhatjuk be a legegyszerűbben. Mindenekelőtt azonban a VirtualHost részen kívül, a konfigurációs állomány végén helyezzük el a következő SSL direktívákat:

```
SSLRandomSeed startup builtin
SSLSessionCache None
```

Az SSL motor bekapcsolásához most építsük meg VirtualHost konfigurációnkat. Helyezzük a következő sorokat is a *httpd.conf* állományba:

```
<VirtualHost www.vbrew.com:443>
SSLEngine On
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:!SSLv2:+EXP:+eNULL
SSLCertificateFile conf/ssl/vbrew.cert
SSLCertificateKeyFile conf/ssl/vbrew.key
</VirtualHost>
```

A megadott sorok bekapcsolják az SSLEngine motort és beállítják a rejtjelező készletet. Megadhatjuk, hogy melyiket szeretnénk engedélyezni, és melyiket nem. A „!" jel a kimondottan tiltani kívánt bejegyzések, a „+" jel az engedélyezettek mellé kerül. Ha a tanúsítványokat eltérő könyvtárban tároljuk, módosítsuk az SSLCertificateFile és KeyFile bejegyzéseket. A *mod_ssl* modulhoz elérhető további opciókról a dokumentációban tájékozódhatunk, amely megtalálható a *mod_ssl* webhelyén.

Hibakeresés

Mivel az Apache konfiguráció rendkívül összetett lehet, könnyen előfordulhatnak benne hibák. Ebben a részben a gyakori hibákkal és megoldásaikkal foglalkozunk.

A konfigurációs állomány ellenőrzése az apachectl segítségével

A rendszergazdák szerencséjére az Apache kiszolgálóhoz kapunk egy, a konfiguráció ellenőrzését szolgáló programot, amellyel megvizsgálhatjuk a konfiguráció módosításait még azelőtt, hogy a működő kiszolgálót leállítanánk. Ha a program hibát talál, arról különböző információkat közöl. Nézzük meg a következő példát:

```
vlager # ../bin/apachectl configtest
Syntax error on line 985 of /var/www/conf/httpd.conf:
Invalid command 'SSLEgine', perhaps mis-spelled or defined by a module not
included in the server configuration
```

A konfigurációt tesztelő eszköz hibát talált a 985. sorban, és úgy tűnik, az SSLEngine direktívát helytelenül írtuk. Az eszköz felismeri a szintaktikai hibákat, ami természetesen nagy segítség. Érdemes a programot mindig lefuttatnunk, mielőtt a kiszolgálókat leállítjuk és újraindítjuk.

A `configtest` azonban nem oldhatja meg az összes gondunkat. Egy IP cím felcserélt számjegyei, az elírt tartománynevek vagy a megjegyzésbe tett, de egyébként szükséges direktívák átcsúsznak a teszten és hibát okoznak a kiszolgáló működésében.

Az oldal nem található

Rendkívül gyakori hiba és számos oka lehet. Az Apache kiszolgáló ilyen hibát jelez, ha nem talál vagy nem képes olvasni egy oldalt. Ha ilyen hibajelzést kapunk, először ellenőrizzük az útvonalakat. Ne feledjük, az Apache kiszolgálóval egy virtuális könyvtári környezetben dolgozunk. Ha e szerkezeten kívül eső állományokra is hivatkozunk, a kiszolgáló valószínűleg nem képes azokat kiszolgálni. Emellett vizsgáljuk meg az állományok engedélyeit, hogy a webkiszolgáló-folyamatot birtokló felhasználó jogosult-e az olvasásukra. A *root* vagy bármely más felhasználó birtokában lévő és 700-as módra (olvasásra/írásra/végrehajtásra jogosult felhasználó) állított állományok hibát okozhatnak a kiszolgálóban, mert nem tudja azokat olvasni.

Az útvonal- és tartományneveket könnyen elgépelhetjük. Bár a configtest némelyiket felismerheti, valószínűleg nem találja meg mindet. Egyetlen elírás is tönkreteheti a teljes webhelyet. Ha hiba fordul elő, nézzünk át mindent kétszer is.

SSL hibák

Ha SSL kiszolgálónk nem működik, több dolog is tönkremehetett. Ha a kiszolgáló nem szolgálja ki az oldalakat, nézzünk bele az *error_log* állományba, ahol rengeteg segítséget kaphatunk. Tegyük fel, hogy a minta kiszolgálónk nem szolgálja ki az SSL oldalakat, a nem titkosított oldalakat viszont igen. Az *error_log* állományban a következőt látjuk:

```
[Wed Aug  6 14:11:33 2003] [error] [client 10.10.0.158] Invalid method in request
\x80L\x01\x03
```

Az ilyen típusú hiba meglehetősen gyakori. Az érvénytelen kérelem (invalid request) arra utal, hogy az ügyfél megkísérli az SSL munkafolyamat egyeztetését, a kiszolgáló azonban valamilyen okból csak nem titkosított oldalakat kínál az SSL porton. Ezt úgy is ellenőrizhetjük, hogy a böngészőt a 443-as portra állítjuk és normál HTTP munkafolyamatot kezdeményezünk. A hiba oka, hogy a kiszolgálót nem utasítottuk az SSLEngine bekapcsolására, illetve a kiszolgáló nem kapcsolta azt be.

A hiba kijavításához nézzük meg, létezik-e a következő sor a *httpd.conf* állományban:

```
SSLEngine On
```

Vizsgáljuk meg az SSL kiszolgálóhoz létrehozott VirtualHost bejegyzést is. Ha gond van az IP címmel vagy a DNS névvel, amelyen a kiszolgálót létre kívántuk hozni, ilyen jellegű hibáról érkezik jelzés. Nézzük meg a konfigurációs állományunk egy részletét:

```
<VirtualHost www.vbrew.cmo:443>
SSLEngine On
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:!SSLv2:+EXP:+eNULL
SSLCertificateFile conf/ssl/vbrew.cert
SSLCertificateKeyFile conf/ssl/vbrew.key
</VirtualHost>
```

A VirtualHost bejegyzés egyetlen elírásának hatására a kiszolgáló a .com helyett a .cmo felső szintű tartományban próbálja a nevet megtalálni. Természetesen az Apache nem ismeri fel a hibát, de pontosan azt teszi, amire utasítottuk.

Más SSL vonatkozású problémák valószínűleg a kulcsok helyére és a jogosultságokra vezethetők vissza. Győződjünk meg arról, hogy a kiszolgáló ismeri a kulcsok helyét, és akiknek szükséges, azok képesek a kulcsokat olvasni. Ügyeljünk arra is, hogy ha a tanúsítvány magunk írtuk alá, egyes ügyfelek a beállításuknak megfelelően esetleg nem fogadják el a tanúsítványt, ami hibához vezet. Ilyen esetben vagy változtassuk meg az ügyfél-munkaállomások konfigurációját, vagy vásároljunk olyan tanúsítványt, amelyet egy harmadik fél írt alá.

15

IMAP

Az *Internet Message Access Protocol* (IMAP) elnevezésű protokollt a mobil e-mail hozzáférés érdekében hozták létre. A cégek alkalmazottai gyakran különféle helyeken (az irodában, otthon, egy hotelszobában és így tovább) olvassák el leveleiket, és igényt tartanak olyan szolgáltatásokra, hogy például először a levelek fejléceit tölthessék le, és azután csak az őket érdeklő leveleket. Az IMAP előtt az internet fő levélkézbesítő protokollja a POP volt, amely kezdetlegesebb, csak kézbesítésre alkalmas szolgáltatást nyújtott.

Az IMAP lehetővé teszi, hogy az úton lévő felhasználók bárhonnan elérjék elektronikus leveleiket és tetszés szerint letöltsék vagy a kiszolgálón hagyják azokat. A POP ellenben nem működik megfelelően, ha a felhasználók más gépről kívánnak e-mailjeikhez hozzáférni; a felhasználók levelei végül a különböző levelező ügyfeleken szétszórva kötnek ki. Az IMAP biztosítja a felhasználóknak, hogy több e-mail levelesládát is kezelhessenek a távolból, és a régi leveleiket eltárolják, keressenek köztük vagy archiválják azokat.

Az IMAP – bevezetés

Az RFC 3501 specifikációban részletesen leírt IMAP protokollt úgy tervezték, hogy megbízható, mobil levélkézbesítő és levélelérési eljárást biztosítson. Az RFC dokumentációból megismerhetjük a protokoll részleteit, működését a hálózati rétegen, illetve számos beállítási lehetőségét.

Az IMAP és a POP

A POP és IMAP protokollokat gyakran egy kalap alá veszik és összehasonlítják, ami egy kicsit igazságtalan, hiszen sok dologban különböznek. A POP protokollt eredetileg egyszerű levélkézbesítő eljárásnak dolgozták ki, és erre tökéletesen megfelel. A felhasználók a kiszolgálóra kapcsolódnak és letöltik üzeneteiket, melyeket aztán – jó esetben – törölnek a kiszolgálóról. Az IMAP teljesen eltérő megközelítést alkalmaz. Az üzenetek őrzőjeként lép fel, és olyan keretrendszert biztosít, amelyben a felhasználók hatékonyan kezelhetik a tárolt üzeneteket. Noha a rendszergazdák és a felhasználók beállíthatják a POP protokollt úgy, hogy az üzeneteket a kiszolgálón tárolja, ennek rossz hatékonyságára hamar fény derül, mert a POP ügyfél a levelek lekérésekor az összes régi üzenetet is letölti. A helyzet gyorsan kaotikussá válhat, ha az ügyfél rendszeresen nagyobb mennyiségű levelet kap. A hordozhatóságot nem igénylő vagy csak kevés levelet fogadó

felhasználók számára a POP megfelelő választás lehet, de a hatékonyabb szolgáltatásokat igénylők számára az IMAP jelenti a megoldást.

Melyik IMAP?

Ha úgy látjuk, hogy az IMAP protokollt nekünk találták ki, alapvetően két lehetőségünk van. Az IMAP két fő változata a Cyrus IMAP és a University of Washington IMAP kiszolgálója. Mindkettő megfelel az IMAP RFC leírásának, és mindkettőnek vannak rendkívül előnyös és hátrányos tulajdonságai. A két változat eltérő formátumú levelesládát használ, ezért nem működnek együtt. Az egyik legfontosabb különbség közöttük, hogy a Cyrus IMAP nem használja az */etc/passwd* állományt a levelezési fiókokat tartalmazó adatbázisához, így a rendszergazdáknak nem szükséges külön levelezőfelhasználókat adniuk a rendszer jelszóállományához. A rendszergazda szempontjából ez biztonságosabb megoldás, mert a fiókok létrehozása rendszerbiztonsági kockázatként is felfogható. Az UW IMAP egyszerű beállítása és telepítése azonban gyakran vonzóbbnak tűnik. A fejezetben elsősorban e két legnépszerűbb IMAP kiszolgálóra fordítjuk figyelmünket: az UW IMAP-ra népszerűsége és egyszerű telepíthetősége, és a Cyrus IMAP-ra a kiegészítő biztonsági szolgáltatásai miatt.

Az IMAP ügyfél beszerzése

Az UW IMAP, ahogy neve is sugallja, a Washington Egyetemről szerezhető be. A *http://www.washington.edu/imap/* címen található weboldalon különféle dokumentációkat és javaslatokat találunk a kiépítéshez, valamint egy hivatkozást a szoftvereiket tároló FTP helyhez. Több különböző verzió is elérhető különféle formákban. Az egyszerűség kedvéért az UW IMAP csapata elhelyezett egy közvetlen hivatkozást is a legfrissebb verzióhoz: *ftp://ftp.cac.washington.edu/mail/imap.tar.Z*.

Az UW-IMAP telepítése

Miután a kiszolgálószoftvert letöltöttük és kicsomagoltuk, kezdődhet a telepítés. Az UW-IMAP a nagy, hordozhatóságot biztosító adatbázisa miatt nem támogatja a GNU automake eljárást, vagyis nincs *configure* szkriptje. Ehelyett a felhasználó által megadott paraméterekre támaszkodó *Makefile* parancsot használhatjuk. A támogatott operációs rendszerek száma nagy, és több Linux terjesztést is találunk közöttük. Íme egy lista néhány támogatott Linux terjesztésről:

```
# ldb   Debian Linux
# lnx   Linux a hagyományos jelszavakkal és a crypt( ) függvénnyel a C
#       könyvtárban (lásd lnp, sl4, sl5, és slx)
# lnp   Linux Pluggable Authentication Module (PAM) támogatással
# lrh   RedHat Linux 7.2
# lsu   SuSE Linux
# sl4   Linux, a crypt( ) függvényhez az -lshadow paranccsal
# sl5   Linux árnyék jelszavakkal, további könyvtárak nélkül
# slx   Linux, a crypt( ) függvényhez az -lcrypt paranccsal
```

Az `lrh` verzió valószínűleg az újabb Red Hat verziókon is fut. Ha saját disztribúciónkat nem találjuk a listán, próbálkozzunk olyannal, amelynek általános leírása megfelelőnek tűnik. Az `lnp` a legtöbb új Linux verzióhoz megfelelő választás lehet.

Ha nem telepítettük az OpenSSL-t, módosítanunk kell a Makefile állományt. Keressük meg az SSL beállítását szolgáló részben a következő sort:

```
SSLTYPE=nopwd
```

A nopwd opciót állítsuk a none értékre, így tájékoztathatjuk az IMAP-ot, hogy nem használunk OpenSSL-t.

Ha telepítettük az OpenSSL-t, de a telepítő mindig hibát jelez, annak valószínűleg az az oka, hogy rossz helyen keresi az OpenSSL-t. Alapértelmezés szerint a *Makefile* előre beállított útvonalon keres, amely a megépítési folyamatban megválasztott értékektől függ. Ha például az IMAP megépítéséhez az `lnp` opciót adtuk meg, az SSL-t az */usr/ssl* könyvtárban keresi. Ha azonban Gentoo Linuxot futtatunk, az SSL könyvtár az */usr* lesz; keressük meg tehát a Makefile SSLPATH opcióját és javítsuk ki az útvonalat. Ugyanez vonatkozik az SSLCERTS opcióra, amely a *Makefile* azonos részén helyezkedik el.

Miután sikeresen lefordítottuk az IMAP kiszolgálót, telepítsük az *inetd.conf* állományba (illetve a *xinetd* állományba, ha azt használjuk). Az *inetd.conf* állományhoz a következő sort adjuk:

```
imap    stream  tcp     nowait  root    /path/to/imapd    imapd
```

Az útvonalat természetesen változtassuk meg úgy, hogy arra a helyre mutasson, ahová az *imapd* bináris állományt telepítettük.

A legtöbb korszerű Linux rendszer meglehetősen jól felépített */etc/services* állománnyal rendelkezik, de azért ellenőrizzük, hogy az IMAP is ott van-e benne. Ehhez keressük meg, illetve ha nem léteznek, helyezzük el a következő sorokat:

```
imap            143/tcp
imaps           993/tcp
```

Amikor elkészültünk, teszteljük a telepítést a *netstat* programmal. Ha a telepítés sikeres, egy figyelőt látunk a 143-as TCP porton:

```
vlager# netstat -aunt
 Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:143             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
```

Mint bármely más szolgáltatás esetében, most is szükség lehet a tűzfal beállítására, hogy engedélyezze az új kapcsolatokat.

Az IMAP beállítása

Az UW IMAP egyik legkellemesebb tulajdonsága, hogy ha már telepítettük, szinte mindig teljesen működőképes. Az alapértelmezett opciókat, beleértve a szabványos */etc/passwd* hitelesítés használatát és a Unix levelesláda formátumot a legtöbb rendszergazda elfogadhatja. Ha nagyobb rugalmasságra vagy több szolgáltatásra van szükségünk, az UW IMAP olyan fejlett konfigurációs beállításokat is lehetővé tesz, mint például a név-

telen bejelentkezés, az IMAP figyelmeztető üzenetek, az eltérő levelesláda formátumok és a megosztott levelesládák használata, amelyről a következő részben szólunk.

Az UW IMAP magas szintű konfigurációs beállításai

Igényeinknek megfelelően számos további beállítási lehetőséget adhatunk meg UW IMAP kiszolgálónk számára. Hasznos beállítás lehet például a névtelen bejelentkezés támogatása. Az eljárással úgy biztosíthatunk adatokat a felhasználóknak, hogy nem szükséges külön fiókot létrehoznunk a részükre. Az eljárást az egyetemeken régóta használják az információ elosztására, illetve csak olvasásra jogosító hozzáférés biztosítására a különféle vitafórumokhoz. A szolgáltatás bekapcsolásához csupán egy *anonymous.newsgroups* nevű állományt kell elhelyeznünk az */etc* könyvtárunkban, és a névtelen felhasználók máris hozzáférhetnek a közös levelesládákhoz.

Szintén hasznos lehet, ha az IMAP felhasználóknak figyelmeztető üzenetet küldhetünk. A szolgáltatást bekapcsolva a leveleik olvasásához bejelentkező felhasználóknak küldhetünk üzenetet. Mivel az üzenet minden alkalommal megjelenik, amikor a felhasználók megnézik levelezésüket, csak vészhelyzetben használjuk. Ne tartalmazzon hirdetést, és ne ide helyezzük a felelősségvállalásról szóló jogi nyilatkozatunkat se. A figyelmeztető üzenet létrehozásához készítsünk egy *imapd.alert* állományt. Ennek tartalma az üzenet szövege.

Eltérő levelesláda formátumok

Az UW IMAP kiszolgálón használt alapértelmezett levelesládára azért esett a választás, mert ez biztosítja a legnagyobb rugalmasságot és kompatibilitást. Noha e két jellemző nagyon fontos, a teljesítménnyel fizetünk érte. Az UW IMAP támogatja az mbx formátumot is, amely hatékonyabb a megosztott levelesládák esetén, mert támogatja az egyidejű olvasást és írást.

Az IMAP beállítása az OpenSSL használatára

Az IMAP számos kényelmi lehetőséget biztosít a levelezésüket kezelő felhasználók számára, de hiányzik belőle egy nagyon lényeges szolgáltatás - a titkosítás. Ezért fejlesztették ki az IMAP-SSL-t, amelyet ha telepítünk, a kompatibilis ügyfélszoftverrel rendelkező IMAP felhasználók az IMAP összes szolgáltatását igénybe vehetik anélkül, hogy az esetleges hallgatózók miatt aggódnának. Ahhoz, hogy az IMAP mellé SSL támogatást is telepítsünk, először győződjünk meg róla, hogy az IMAP kiszolgáló telepítése megfelelő és a kiszolgáló működik. Szükségünk van továbbá egy működő OpenSSL telepítésre is. A legtöbb Linux terjesztésben megtalálható az OpenSSL, de ha valamiért terjesztésünk nem tartalmazná, az OpenSSL megépítéséről szóló leírást az Apache kiszolgálóval foglalkozó fejezetben találhatjuk meg.

A konfiguráció első lépése a digitális tanúsítványok létrehozása az IMAP kiszolgáló számára. Ezt megtehetjük az OpenSSL parancssori segédprogramjával. A következő példa egy minta tanúsítvány létrehozását szemlélteti:

```
vlager# cd /path/to/ssl/certs
vlager# openssl req -new -x509 -nodes -out imapd.pem -keyout imapd.pem -days 365
Using configuration from /etc/ssl/openssl.cnf
```

```
Generating a 1024 bit RSA private key
A konfigurálás az /etc/ssl/openssl.cnf állomány alapján történik.
Az 1024 bites RSA saját kulcs létrehozása.
.............++++++
.....................................................++++++
writing new private key to 'imapd.pem'
az új saját kulcs beírása a 'privkey.pem' állományba
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
A következőkben a tanúsítványkérelemhez szükséges adatokat kell megadnia.
A megadott információ az úgynevezett Megkülönböztetett Név (Distinguished Name, DN).
A számos mező közül egyeseket üresen hagyhat.
Más mezők rendelkeznek egy alapértelmezett értékkel,
ezeket a '.' beírásával hagyhatjuk üresen.
-----
Country Name (2 letter code) [AU]:
Országnév (kétbetűs kód)
.
.
Common Name (eg, YOUR name) [  ]: mail.virtualbrewery.com
Az általánosan használt név (például az ÖN neve)
Email Address [  ]:
E-mail cím
vlager # ls -l
total 4
-rw-r--r--    1 root     root         1925 Nov 17 19:08 imapd.pem
vlager #
```

A tanúsítvány létrehozásakor ne felejtsük el megadni a levelezőkiszolgálónk tartománynevét a Common Name mezőben! Ha nem vagy rosszul adjuk meg, a legjobb esetben hibaüzenetet kapunk, amikor az ügyfelek megpróbálnak kapcsolódni, de rossz esetben a kiszolgáló is leállhat.

Az IMAP kiszolgálót az OpenSSL használatához valószínűleg újra kell fordítanunk és konfigurálnunk. Szerencsére ez elég egyszerű. Ha Red Hat, SuSE vagy bármely más, a listában szereplő terjesztést használunk, a parancssorban adjuk meg a terjesztés nevét; a következő parancssori opciók egyébként a legtöbb más terjesztés esetén is működnek:

```
vlager#  make lnp PASSWDTYPE=pam SSLTYPE=nopwd
```

Ha az OpenSSL-lel kapcsolatban hibaüzenetet kapunk, az útvonalak beállításával lehet gond. A beállítást a *Makefile* `SSLDIR`, `SSLLIB` és `SSLINCLUDE` opcióival végezhetjük el. A legtöbb esetben erre nem lesz szükség.

A fordítást követően másoljuk az új IMAP kiszolgálót az építési könyvtárból a rendszer egyéb daemon állományait tartalmazó könyvtárába. Mivel az IMAP-SSL a szabványos IMAP-tól eltérő portot használ, az *inetd.conf* állományt is módosítanunk kell.

```
imaps stream tcp nowait root /path/to/imapd imapd
```

Ha a *xinetd* állománnyal dolgozunk, a következő minta alapján hozzunk létre egy állományt az */etc/xinetd.d* könyvtárunkban:

```
service imaps
{
        socket_type             = stream
        wait                    = no
        user                    = root
        server                  = /path/to/imapd
        log_on_success          += DURATION USERID
        log_on_failure          += USERID
        disable                 = no
}
```

Ellenőrizzük, hogy az */etc/services* állomány tartalmaz egy *imaps* bejegyzést:

```
vlager # cat /etc/services |grep imaps
imaps           993/tcp                         # IMAP over SSL
imaps           993/udp                         # IMAP over SSL
vlager #
```

Most már tetszőleges számú ügyfélről tesztelhetjük a kiszolgálónkat. Győződjünk meg róla, hogy az ügyfelek konfigurációjában jeleztük, hogy SSL-t fogunk használni. A kapcsolódáskor az egyes ügyfeleken üzenetet kaphatunk, amely rákérdez, hogy meg kívánunk-e bízni a tanúsítványban. Az üzenet csak akkor jelenik meg, ha saját tanúsítványt készítettünk, ahogy az előbbi példában tettük. Bizonyos esetekben, például ha a kiszolgálót üzleti célra használjuk, érdemes ennek elkerülésére egy tanúsítványt vásárolnunk.

A Cyrus IMAP

Az IMAP rendszergazdák másik lehetősége a CMU Cyrus terméke. A Cyrus az általános funkcionalitás tekintetében hasonlít az UW IMAP-ra – a felhasználó szemszögéből csekély az eltérés. A különbségek nagy része az adminisztrációban mutatkozik, ugyanakkor az előnyeit is itt tapasztalhatjuk.

A Cyrus IMAP beszerzése

A Cyrus szoftvert több forrásból is beszerezhetjük, de a legújabb forráskódokat tartalmazó és legmegbízhatóbb forrás a CMU központi Cyrus elosztó webhelye a *http://asg.web.cmu.edu/cyrus/download/* címen. Itt a legfrissebb kibocsátások mellett a régebbi verziókat is letölthetjük. A régi verziók előnyösek lehetnek az olyan webhelyeken, ahol nem szívesen használják a szoftverek legfrissebb verzióit.

A Cyrus kiszolgáló telepítéséhez töltsük le és csomagoljuk ki a legújabb változatot. Ehhez le kell töltenünk az IMAP és a SASL csomagokat.

A SASL a Cyrus IMAP hitelesítési eljárását biztosítja, ezt kell először telepítenünk és beállítanunk. A megépítéséhez a szabványos „configure-make" sorrendet használjuk.

```
vlager# cd cyrus-sasl-2.1.15
vlager# ./configure
loading cache ./config.cache
checking host system type... i686-pc-linux-gnu
.
creating saslauthd.h
Configuration Complete. Type 'make' to build.
vlager# make
make  all-recursive
make[1]: Entering directory `/tmp/cyrus-sasl-2.1.15'
```

Feltéve, hogy a fordítás hiba nélkül zajlott és a make install parancsot is sikeresen futtattuk, hozzákezdhetünk magának a Cyrus IMAP kiszolgálónak a beállításához és telepítéséhez.

A Cyrus IMAP forrás kicsomagolását követően a következő paranccsal kezdhetjük meg a beállítást:

```
vlager# ./configure --with-auth=unix
```

A parancs előkészíti a Cyrus IMAP kiszolgálót a Unix */passwd/shadow* állományok használatára a hitelesítéshez. Ezen a ponton adhatjuk meg a Kerberost is, ha a hitelesítést azzal szeretnénk végezni.

Ezután létrehozzuk a szükséges függő állományokat, majd megépítjük és telepítjük a csomagot:

```
vlager# make depend
 . . .
vlager# make all CFLAGS=-O
 . . .
vlager# make install
 . . .
```

Amikor az előkészítő lépéseket sikeresen végrehajtottuk, a Cyrus IMAP kiszolgáló készen áll a beállításra.

A Cyrus IMAP beállítása

Hozzunk létre egy felhasználót a Cyrus kiszolgálóhoz. A felhasználó neve utaljon a kiszolgálóra, és legyen része a levelező csoportnak.

A felhasználó létrehozása után állítsuk be a Cyrus kiszolgálót. A kiszolgáló fő konfigurációs állománya az */etc/imapd.conf* állomány. Ellenőrizzük, hogy tartalma hasonlít-e a következő mintára. Előfordulhat, hogy a sorok egyikét-másikát nekünk kell beírnunk.

```
configdirectory:        /var/imap
partition-default:      /var/spool/imap
sievedir:              /var/imap/sieve
# Don't use an everyday user as admin.
admins:                cyrus root
```

```
hashimapspool:          yes
allowanonymouslogin:    no
allowplaintext:         no
```

Hibakeresés a Cyrus IMAP kiszolgálón

A Cyrus IMAP megépítése nem egyszerű feladat, mert nagyon kényes az állományokra és elhelyezkedésükre. Ha a beállítási folyamattal gondjaink vannak, vegyük alaposan szemügyre, mi okozza a hibát. Ha például a Cyrus-SASL építésekor hiba jelentkezik, amely a berkeley_db szekció nem definiált hivatkozásairól panaszkodik, valószínűleg nem telepítettük a BerkeleyDB-t, vagy olyan helyre telepítettük, ahol a szkript nem keresi. A telepített BerkeleyDB útvonalát a parancssorból beállíthatjuk, amikor a konfigurációs szkriptet futtatjuk. Sok hibát megoldhatunk, ha először pontosan megkeressük a hiányosság okát, majd kijavítjuk a hibát.

A Cyrus IMAP megépítésekor felmerülő másik gyakori hiba a *com_err.h* állomány helyével kapcsolatos. A Cyrus IMAP az állományt az */usr/include* könyvtárban keresi, noha gyakran az */usr/include/et* könyvtárban található. Ilyenkor a telepítés folytatásához másoljuk az állományt az */usr/include* könyvtárba.

16

Samba

A Microsoft Windows gépek jelenléte a hálózati környezetben gyakran megkerülhetetlen a rendszergazdák számára, és sokszor az is rendkívül fontos, hogy e gépek együttműködjenek más számítógépekkel. Szerencsére az elmúlt tíz évben a fejlesztők egy kis csoportja keményen dolgozott azon, hogy létrehozzon egy sokoldalú, Windows–Unix kapcsolatot biztosító csomagot: a Samba szoftvert. Ami azt illeti, a Samba olyan sikeres és jól használható, hogy a rendszergazdák teljes egészében helyettesíthetik vele a Windows kiszolgálókat, miközben ugyanazon szolgáltatások mellett nagyobb stabilitást érhetnek el.

A Samba – bevezetés

A Samba, amelyet jelenleg is aktívan fejlesztenek, hogy kompatibilis maradhasson a szüntelenül változó Microsoft szoftverrel, olyan keretrendszert biztosít, amelynek segítségével a Linux számítógépek hozzáférhetnek a Windows hálózati erőforrásokhoz, például a megosztott illesztőprogramokhoz és nyomtatókhoz. Amellett, hogy lehetővé teszi e szolgáltatások elérését, a Samba arra is lehetőséget teremt, hogy a Linux ugyanezen szolgáltatásokat kínálja a Windows számítógépeknek. A Sambával tökéletesen helyettesíthetjük a Windows alapú állománykiszolgálót, a Windows nyomtatókiszolgálót, valamint - a fejlett beállítási lehetőségekkel - akár a Primary Domain Controller (PDC) tartományvezérlőt is. A Samba legújabb változatai ráadásul az Active Directory kompatibilitást is biztosítják. A Samba nyílt forráskódjának köszönhetően a fejlesztés tovább folyik, és a Windows architektúra változásával új szolgáltatások jelennek meg. A Sambáról többet Jay Ts, Robert Eckstein és David Collier-Brown *Using Samba* című könyvének második kiadásában olvashatunk (kiadója az O'Reilly).

SMB, CIFS és Samba

A Samba hátterében a *Server Message Blocks* (SMB) technológia áll, amelyet eredetileg a nyolcvanas évek elején fejlesztett ki Dr. Barry Feigenbaum, aki akkor az IBM-nél dolgozott. Kezdetben az IBM aktívan részt vett a fejlesztésben, de a Microsoft hamarosan átvette a kezdeményezést, még több munkát fektetve bele. Pár évvel később a Microsoft átnevezte az SMB protokollt a ma ismert *Common Internet File System* (CIFS) névre. A két elnevezést gyakran szinonimaként használjuk.

A CIFS működéséről kevés pontos, hivatalos dokumentáció létezik. A legtöbb más hálózati protokolltól eltérően nem létezik hozzá hivatalos RFC dokumentáció. Noha a Microsoft a kilencvenes években benyújtotta a specifikációt az IETF-hez, a sok pontatlanság és következetlenség miatt az végül érvényét vesztette. A Microsoft újabb kísérletei a dokumentálásra sem sokat segítettek a Samba fejlesztői csoportjának, részben a licenccel kapcsolatos korlátozások, részben a friss információk hiánya miatt.

A Samba beszerzése

A Samba beszerzésére több lehetőség is kínálkozik. Sok terjesztés ma már külön kérés nélkül is telepíti. Ha saját disztribúciónkkal is ez a helyzet, nem szükséges forrásból megépítenünk. A Red Hat, Mandrake és SuSE felhasználók a csomagot RPM-ből telepíthetik, amelyet a Samba tükröroldalakról tölthetnek le. A Gentoo-felhasználók egyszerűen az *emerge samba* paranccsal telepíthetik a csomagot, míg a Debian-felhasználók ugyanezt az *apt-get* utasítással tehetik meg. Más terjesztések felhasználói, illetve azok, akik inkább ezt választják, a Samba csomagot forrásból telepíthetik. Gyakran ez a megoldás biztosítja a legnagyobb rugalmasságot, mert ilyenkor elérhetjük a fordításnál megadható beállítási lehetőségeket is.

Megépítés forrásból

Ha forrásból szeretnénk a programot megépíteni, szerezzük be a legújabb archivált forrást a központi *http://www.samba.org* oldalon felsorolt Samba tükröroldalak valamelyikéről. Miután letöltöttük, csomagoljuk ki egy könyvtárba, és a kapott konfigurációs állomány segítségével építsük meg a bináris állományokat.

```
vlager#  tar xzvf samba-current.tgz
vlager#  cd samba-3.0.0
vlager#  cd source
vlager# ./configure
 . . .
 . . .
vlager# make
 . . .
vlager# make install
```

A szoftver fordítása és telepítése után megválaszthatjuk, hogyan szeretnénk futtatni: *inetd* szolgáltatásként vagy daemonként. Mindkét megoldás jó, de ha a Sambát az *inetd-ből* futtatjuk, frissítenünk kell az */etc/services* állományunkat; ellenőrizzük, hogy a Samba protokollt leíró következő sorok megtalálhatók-e az állományban:

```
netbios-ns      137/tcp                 # NETBIOS név szolgáltatás
netbios-ns      137/udp
netbios-dgm     138/tcp                 # NETBIOS datagram szolgáltatás
netbios-dgm     138/udp
netbios-ssn     139/tcp                 # NETBIOS munkafolyamat szolgáltatás
netbios-ssn     139/udp
microsoft-ds    445/tcp         Microsoft-DS
microsoft-ds    445/udp         Microsoft-DS
```

Miután a sorokat az állományba szúrtuk, illetve a meglétüket ellenőriztük, a Sambát az *inetd.conf*, illetve a *xinetd* állományunkhoz adhatjuk. Vizsgáljuk meg azt is, hogy tűzfalunk engedélyezi-e a szükséges portokat.

Amennyiben a Sambát daemon folyamatként szeretnénk futtatni, bejegyzést kell készítenünk az *rc* indítószkriptben. Ezt a különböző Linux terjesztésekben eltérő módon tehetjük meg, ezért ha nem vagyunk biztosak az eljárásban, olvassuk el a terjesztésünkkel kapott dokumentációt.

Ismerkedés a Sambával

A Samba fordítását, illetve telepítését követően a beállítás esetenként időrabló feladatával nézünk szembe. A Samba végtelen rugalmasságából következik, hogy nagyon sok beállítási lehetőséget kínál. Szerencsére az állománykiszolgáló szolgáltatás beállítása könnyen érthető. Ebben a részben néhány alapvető beállítási lehetőséggel ismerkedünk meg, és megnézzük, hogyan hozhatunk létre osztott állománykönyvtárakat. További információt a *Using Samba* című kötetben találunk (kiadója az O'Reilly).

Alapvető beállítási lehetőségek

A Samba beállítását a legegyszerűbben a minimális konfiguráció létrehozásával kezdhetjük el. Ezért első lépésként csak egy munkacsoportot hozunk létre, megadjuk a kiszolgálónk nevét, és egyszerű állománymegosztást biztosítunk:

```
[global]

workgroup = Brewery
netbios name = vlager

[share]

path = /home/files
comment = néhány recept a házi sörfőzéshez
```

A Samba konfigurációt a *testparm* utasítással ellenőrizhetjük. A parancs feldolgozza a konfigurációs állományokat és felhívja a figyelmet az elgépelésekre és a helytelen beállításokra. Ennél a pontnál még nem valószínű, hogy hibát találunk, de nem árt, ha megismerkedünk az eszköz használatával.

Ha minden megfelelően működik, elindíthatjuk, illetve újraindíthatjuk a Samba kiszolgálót, és megpróbálhatjuk megnézni az új állománymegosztást. Az *mbclient* program, amely a Samba csomag része, az állománymegosztás megtekintésére szolgál. A példában a **vlager (10.10.0.5)** gépen most létrehozott megosztásokat vizsgáljuk:

```
client#  smbclient -L 10.10.0.5
Password:

        Sharename       Type      Comment
        ---------       ----      -------
        share           Disk      néhány recept a házi sörfőzéshez
client#
```

Miután a vizsgálatot is sikeresen elvégeztük, beállíthatjuk a fejlettebb konfigurációs lehetőségeket, amelyek a Sambát jóval hasznosabbá teszik számunkra.

A Samba felhasználói fiókjainak beállítása

A megadott minimális konfiguráció tökéletes, ha egyszerű nyitott állománymegosztásra van szükségünk; a hozzáférés szabályozását azonban még nem oldottuk meg. Az előbbi konfigurációban bárki igénybe veheti a megosztást, ami egy hálózaton általában nem kívánatos. Ezért a Samba hitelesítési szolgáltatással is rendelkezik.

A korábbi példánál maradva, a következő módon kapcsolhatjuk be a hitelesítést:

```
security = user
encrypt passwords = yes
smb passwd file = /etc/samba/private/smbpasswd
username map = /etc/samba/smbusers
```

Az első sor a felhasználói szintű biztonságot kapcsolja be. Ez azt jelenti, hogy a felhasználói állományt a Samba kiszolgálón kezeljük. A második sor jelzi, hogy a Samba jelszóállományát titkosítani kívánjuk. A harmadik és negyedik sorok a jelszóállomány és a felhasználói állomány útvonalát tartalmazzák. A Linux alatt nem szükséges önálló felhasználói állománnyal rendelkeznünk, de erre is van lehetőségünk.

Ezután felhasználókat adunk a rendszerhez. A Samba az *smbpasswd* állományt biztosítja a felhasználói fiókok kezelésére. Erre nem mindig van szükség, hiszen a szoftver az */etc/passwd* állományban is képes a felhasználókra hivatkozni. Ha azonban egyes felhasználóink Microsoft környezetből férnek hozzá az új Samba kiszolgálónkon tárolt állományokhoz, rendelkeznünk kell az *smbpasswd* állománnyal is.

A felhasználók létrehozása egyszerű, és az *smbpasswd* eszközzel végezhető:

```
vlager# smbpasswd -a larry
New SMB password:
Retype new SMB password:
vlager#
```

A felhasználó létrehozása után az *smbmount* segítségével tesztelhetjük a fiókot, akár windowsos, akár linuxos számítógépről. Miután az Explorer címsorában megadjuk a `\\<server.ip>` címet, a Windows felhasználók a 16.1. ábrán bemutatott párbeszédablakot láthatják.

A Linux és más Unix rendszereken az *smbmount* meghívása így történik:

```
vlager#  mount -t smbfs -o username=larry //server.ip /mnt/samba
```

Ha minden jól ment, most már elérhetjük a megosztott könyvtárt az */mnt/samba* útvonalon.

16.1. ábra. A Windows hálózati bejelentkezési párbeszédablaka

A Samba további beállítási lehetőségei

Mindeddig a Samba kiszolgáló elemi beállítási lehetőségeivel foglalkoztunk. A konfiguráció tökéletesen megfelel azoknak, akik csupán szeretnének gyorsan beszerezni valamit a hálózaton keresztül. Azért, hogy többet is kihozzunk a Sambából, a további lehetőségekkel is megismerkedünk.

A hozzáférés szabályozása

A Samba kiszolgáló további biztonsági beállításokat is kínál, amelyek nagy hálózatokon tehetnek jó szolgálatot. Az IP címen alapuló hozzáférés-szabályozás a már ismerős parancsokkal érhető el:

```
hosts allow = 10.10.
hosts deny = any
```

A beállítás kizárólag a **10.10** hálózatból engedélyez kapcsolatokat. A Samba IP alapú hozzáférésszabályozása a *tcpwrappers* logikáját követi, és IP címeket vagy tartományokat engedélyezhetünk vagy tilthatunk a segítségével. Az opció valóban kényelmes, mert globálisan és a megosztás szintjén egyaránt használható, vagyis készíthetünk IP címlistát aszerint, hogy engedélyezni vagy tiltani szeretnénk a hozzáférést a kiszolgáló valamennyi megosztásához, de le is bonthatjuk a folyamatot, és azt is beállíthatjuk, hogy bizonyos IP címek csak bizonyos osztott könyvtárakhoz férjenek hozzá. Ilyen beállítási lehetőséget még maga a Windows sem kínál!

A Samba kiszolgáló rugalmas abban a tekintetben is, hogy melyik interfészre kapcsolódik. Alapértelmezés szerint az összes elérhető interfészhez kapcsolódik, beleértve a visszacsatoló interfészt is. A nem kívánt hozzáférések elkerüléséhez – például azért,

hogy a Samba ne hozzon létre kötést egy kétkapcsolatos számítógép külső interfészéhez - explicit módon megadhatjuk a kötni kívánt interfészt:

```
bind interfaces only = True
interfaces = eth1 10.10.0.4
```

Így a Samba kizárólag a megadott interfészre figyel a megadott IP címen. Biztonsági szempontból helyesebb, ha a hozzáférést az alkalmazások szintjén korlátozzuk, és nem a tűzfalra bízzuk a védelmünket.

A Samba egy másik hozzáférés-szabályozási lehetősége az osztott könyvtárak megjelölése böngészhetőként. Amikor egy könyvtár böngészhető, azonnal láthatóvá válik és a felhasználók tanulmányozhatják a tartalmát. A szolgáltatás beállítására szolgáló parancs:

```
browsable = yes|no
```

Ha azt szeretnénk, hogy a felhasználók bizonyos állományokhoz hozzáférjenek - például úgy, hogy egy állományra mutató URI címet küldünk nekik -, de ne lássák a könyvtár összes állományát, állítsuk az opciót a `no` értékre. Az ilyen típusú URI valószínűleg már ismerős; a *||vlager.vbrew.com|recipe|secret.txt* cím például Windows rendszeren, míg az *smb://vlager.vbrew.com/recipe/secret.txt* cím Unix rendszeren működik. A felhasználó mindkét esetben csak a *secret.txt* állományhoz férhet hozzá. Az opciót általában a `yes` értékre állítjuk, mert a Samba rendszerint jobban működik, amikor a felhasználók böngészhetik a fájlrendszert.

Hasonló módon az osztott könyvtárakat abból a szempontból is megjelölhetjük, hogy azok nyilvánosan elérhetők-e vagy sem. Ha egy mappa nem nyilvános, a felhasználók még egy adott állományra mutató URI címmel sem érhetik el az állományt. Fontos megjegyeznünk azonban, hogy a nyilvánosként megjelölt könyvtárakat a Samba fiókkal rendelkező összes felhasználó megtekintheti.

```
public = yes|no
```

Miután a felhasználóknak jogot adtunk a betekintésre, a Samba rendszergazdája eldöntheti, hogy az osztott könyvtárak írhatók legyenek-e. Ehhez a writable parancsot használhatja:

```
writable = yes|no
```

Ha az opciónak a `no` értéket adjuk, a könyvtárba semmit nem írhatunk.
Ha nyitottabb Samba kiszolgálót tervezünk, például egy szabadon elérhető, böngészhető dokumentációs kiszolgálót, a következő direktívával bekapcsolhatjuk a Samba vendégfiókját:

```
guest ok = yes|no
```

Végül ejtsünk szót a Samba egyik leghasznosabb tulajdonságáról, amelynek segítségével a hozzáférést a megosztott könyvtárakhoz felhasználói hozzáférési listával szabályozhatjuk. Ezt legegyszerűbben a `valid users` opcióval valósíthatjuk meg:

```
valid users = sharon paul charlie pat
```

Az eljárást tovább egyszerűsíthetjük, ha az */etc/groupállomány* egy már definiált csoportját adjuk meg:

```
valid users = @brewers
```

A Samba felismeri, hogy az at (@) jelet követő *brewers* érték a csoport neve. Amikor beállítottuk a kívánt hozzáférés-szabályozást, valószínűleg kíváncsiak vagyunk, működik-e a konfiguráció. A Sambában lejátszódó folyamatokat a kiváló naplózási szolgáltatásán keresztül követhetjük nyomon.

A Samba naplózó szolgáltatása

A Samba naplózó szolgáltatásának használata egyszerű. A szolgáltatás lehetőségeit számos kiegészítő kapcsolóval bővíthetjük. A legegyszerűbb naplózást a következő parancscsal kapcsolhatjuk be:

```
log file = /var/log/samba.log
```

Az utasítás a Samba valamennyi műveletéről egyszerű bejegyzéseket helyez a megadott állományba. Esetenként azonban a naplóállományok ormótlanra duzzadhatnak, ha sok gép veszi igénybe a kiszolgálót. Azért, hogy a naplókat könnyebben olvashassuk, a Samba képes minden kapcsolódó gazdagéphez külön naplót készíteni. Ehhez csupán a következőt sort kell megadnunk:

```
log file = /var/log/samba.log.%m
```

A Linuxon a legtöbb naplózó szolgáltatásban beállíthatjuk, hogy milyen mennyiségű információra tartunk igényt. Ez a Samba esetén is így van. A Samba naplózási szintje egy 0-tól 10-ig terjedő skálán adható meg, ahol a nagyobb értékek részletesebb bejegyzéseket eredményeznek. Az átlagos felhasználó számára a Samba dokumentációja a 2-es szintet javasolja; ezen a szinten elegendő információt kapunk a hibakereséshez, mégsem esünk túlzásba. A 3-as és magasabb szinteket a Samba programozóinak tervezték, nem pedig hétköznapi használatra. A Samba konfigurációs állományában így adhatjuk meg a naplózás szintjét:

```
log level = 2
```

Ha kiszolgálónk sokat dolgozik, a naplóállományok is gyorsan növekednek. Ezért a Samba lehetővé teszi, hogy meghatározzuk a naplóállomány maximális méretét:

```
max log size = 75
```

A példában a naplóállomány maximális méretét 75 KB-ra állítottuk. Amikor a naplóállomány eléri ezt a méretet, a Samba automatikusan átnevezi, egy *old* kiterjesztést fűzve az állomány nevéhez, majd egy új naplóállományt hoz létre. Amikor az új naplóállomány

is eléri a 75 KB-os méretet, a Samba felülírja a régi *.old* állományt. Amennyiben szeretnénk megtartani a régi naplókat is, egy szkripttel automatizálhatjuk a Samba naplóinak archiválását.

Naplózás a syslog segítségével

A saját naplózó szolgáltatása mellett a Samba – ha a `--with-syslog` opcióval fordítjuk – a rendszernaplót is igénybe veszi. Olyan helyzetekben, amikor a rendszergazdák automatizált naplófigyelő eszközöket használnak, például a *swatch* eszközt, ez a megoldás célszerűbb lehet. Ha azt szeretnénk, hogy a Samba használja a rendszernaplót, helyezzük el a következő sort a konfigurációs állományában:

```
syslog = 2
```

Ekkor a Samba valamennyi 2-es szintű naplóbejegyzése a rendszer naplóállományába is bekerül. Ha ez megfelel nekünk és kizárólag a rendszernaplót szeretnénk használni, helyezzük el a következő sort az *smb.conf* állományban:

```
syslog only = yes
```

Nyomtatás a Sambával

A Samba teljes értékű Windows nyomtatókiszolgáló. A felhasználók kapcsolódhatnak rá, nyomtathatnak, de még azt is lehetővé teszi, hogy szükség szerint illesztőprogramokat töltsenek le a nyomtatókhoz. A Windows nyomtatási feladatokat kezelő rendszerének bonyolultsága miatt a Samba csapatának az együttműködés kialakításakor ez jelentette az egyik legnagyobb kihívást. A Sambát sokféleképpen beállíthatjuk a nyomtatásra, de a két legelterjedtebb beállítás a hagyományos BSD és az újabb CUPS nyomtatás.

BSD nyomtatás

A régebbi, hagyományos nyomtatási eljárás a BSD nyomtatási rendszer, amely az RFC 1179 keretrendszeren alapul. Olyan parancsokat használ, mint például az *lpr*, amely a legtöbb Unix rendszergazda számára ismerős. Ebben a környezetben a Samba jól működik és könnyen beállítható. A BSD nyomtatás bekapcsolásához szükséges alapvető konfiguráció így fest:

```
[global]
printing = bsd
load printers = yes

[printers]
path = /var/spool/samba
printable = yes
public = yes
writable = no
```

Ha arra gondolunk, hogy ez így nagyon egyszerű, igazunk van. Köszönhetjük annak, hogy a Samba számos feltételezéssel él az alapértelmezett konfigurációjában. A konfiguráció *tényleges* kinézetét megtekinthetjük a *testparm* programmal:

```
ticktock samba # testparm -s -v |egrep "(lp|print|port|driver|spool|\[)"
Processing section "[printers]"
[global]
        smb ports = 445 139
        nt pipe support = Yes
        nt status support = Yes
        lpq cache time = 10
        load printers = Yes
        printcap name = /etc/printcap
        disable spoolss = No
        enumports command =
        addprinter command =
        deleteprinter command =
        show add printer wizard = Yes
        os2 driver map =
        wins support = No
        printer admin =
        nt acl support = Yes
        min print space = 0
        max reported print jobs = 0
        max print jobs = 1000
        printable = No
        printing = bsd
        print command = lpr -r -P'%p' %s
        lpq command = lpq -P'%p'
        lprm command = lprm -P'%p' %j
        lppause command =
        lpresume command =
        printer name =
        use client driver = No
[printers]
        path = /var/spool/samba
        printable = Yes
```

Ha az alapértelmezett opciók rendszerünkhöz nem megfelelőek, most átállíthatjuk azokat. Vizsgáljuk meg például, hogy az *lp* parancsok az alapértelmezett útvonalon vannak-e. Az *lp* parancssorhoz is bármit hozzáfűzhetünk, ha úgy gondoljuk, szükségünk van rá.

Bemutatunk egy minta */etc/printcap* állományt is. Mivel minden rendszer különböző, a konfigurációs állomány nem feltétlenül felel meg mindenkinek, de megismerhetjük belőle a konfiguráció formátumát. Részletes információt a *printcap* súgóoldalakon találunk.

```
# /etc/printcap: printer capability database.

/lp|Generic dot-matrix printer entry
        :lp=/dev/lp1
        :sd=/var/spool/lpd/lp
        :af=/var/log/lp-acct
        :lf=/var/log/lp-errs
        :pl#66
        :pw#80
        :pc#150
        :mx#0
        :sh
```

Nyomtatás a CUPS segítségével

A *Common Unix Printing System* (CUPS) a legtöbb Linux terjesztésben fokozatosan átveszi a BSD nyomtatás helyét. A status quo változásának több oka is van, ezekre most nem térünk ki. A Samba környezet bemutatása azonban nem volna teljes, ha nem szólnánk a CUPS nyomtatásról is.

A CUPS megismerését kezdjük egy egyszerű, CUPS-hoz beállított *smb.conf* állománnyal:

```
[global]
load printers = yes
printing = cups
printcap name = cups

[printers]
comment = Sörgyári nyomtatók
path = /var/spool/samba
browsable = no
public = yes
guest ok = yes
writable = no
printable = yes
printer admin = root, @wheel
```

Ha a CUPS működőképes a rendszerünkön, ez minden, ami ahhoz szükséges, hogy a Samba az egyszerű beállításokkal használni tudja a CUPS-t.

A global részben megadjuk, hogy az */etc/printcap* állomány a `load printers` opcióval töltődjön be. Az opció automatikusan elemzi a printcap állományt és elindítja az abban beállított nyomtatókat. Az opciót nem minden rendszergazda használja szívesen. Részint ugyan megkönnyíti a konfigurációt, részint azonban némileg korlátozza a rendszergazda beleszólási lehetőségét abba, hogy mi látható majd a felhasználók számára. Ha az opciót kikapcsoljuk, az egyes megosztásokat egyenként kell megadnunk a Samba konfigurációs állományában.

A printing opció, amelyet korábban a `bsd` értékre állítottunk, most a `cups` értéket tartalmazza. Az érthetőség kedvéért a printcap is a *cups* nevet kapta.

A következő, printers rész szintén hasonlít a korábbi példára; most azonban egy nyomtató adminisztrációs csoportot is beállítottunk. Ez az opció adminisztrációs jogokat biztosít a csoportnak a nyomtatás CUPS-szal kapcsolatos beállításaihoz.

Noha nem feltétlenül szükséges, további opciókkal pontosabban testre szabhatjuk a cups nyomtatást. A CUPS és a Samba részleteiről és fejlett beállítási lehetőségeiről a Samba webhelyén, a *http://www.samba.org,* illetve a CUPS webhelyén, a *http://www.cups.org* címen tájékozódhatunk.

A SWAT használata

A *Samba Web Administration Tool* (SWAT) célja, hogy leegyszerűsítse a Samba adminisztrációját és beállítását. Azoknak, akik a GUI felületet részesítik előnyben, ideális megoldást jelent, mert a Samba fejlesztői csapata írta, és az összes lehetséges beállítást tartalmazza. Más grafikus felületű programok is léteznek, de azok régebbiek.

A SWAT futtatása

A SWAT alapvetően egy webkiszolgáló és adminisztrációs eszköz egyben. Azért, hogy megfelelően működjön, adjuk hozzá *inetd.conf* vagy *xinetd* konfigurációnkhoz. A *xinetd* esetén konfigurációnk valahogy így festhet:

```
service swat
{
        port            = 901
        socket_type      = stream
        wait            = no
        user            = root
        server          = /usr/sbin/swat
        log_on_failure += USERID
        disable         = no
}
```

A SWAT portot is meg kell adnunk az */etc/services* állományban, ha még nem létezne:

```
swat            901/tcp                         # Samba konfigurációs eszköz
```

A *xinetd* folyamatot újraindítva máris használhatjuk a szolgáltatást; ehhez irányítsuk webböngészőnket a Samba kiszolgáló IP címére, a 901-es porton.

A SWAT és az SSL

Észrevehettük, hogy a SWAT nem használ SSL-t, amivel nincs is baj, ha csak a helyi gépről használjuk. Ha azonban hálózaton keresztül szeretnénk igénybe venni, érdemes titkosítást alkalmaznunk. Noha a SWAT nem támogatja közvetlenül a titkosítást, a népszerű *stunnel* SSL eszközzel változtathatunk rajta.

A beállításhoz Markus Krieger egyszerű eljárást fejlesztett ki. Az eljáráshoz szükségünk van a telepített OpenSSL-re és az *stunnel* programra. Az *stunnel* dokumentációját és forrását megtaláljuk a *http://www.stunnel.org* címen. Ha már mindkettőt telepítettük és működőképesek, hozzunk létre egy saját kulcsot:

```
vlager# /usr/bin/openssl req -new -x509 -days 730 -nodes -config
/path/to/stunnel/stunnel.cnf -out /etc/stunnel/stunnel.pem -keyout
/etc/stunnel/stunnel.pem
```

A kulcsok létrehozása után távolítsuk el az eredeti SWAT-ot az *inetd.conf* állományunkból, és ha szükséges, küldjünk egy SIGHUP jelet a daemonnak. A szabványos SWAT daemont többet nem hívjuk meg az *inetd* segítségével, ezért eltávolíthatjuk.

A *stunnel* most már futásra kész. A programot elindíthatjuk a parancssorból, vagy készíthetünk egy kis szkriptet az automatikus indításhoz. A *stunnel* programot *root*-ként így indíthatjuk el:

```
vlager# stunnel -p /etc/stunnel/stunnel.pem -d 901 -l
/path/to/samba/bin/swat swat
```

Hibakeresés a Samba kiszolgálón

Ha megépítettük a Samba kiszolgálót és elsőre minden rendben ment, a szerencsés kevesek közé számíthatjuk magunkat. Mindenki másnak azonban jól jöhet néhány tipp, hogyan kereshetjük meg és javíthatjuk ki a leggyakoribb hibákat.

Gondok a konfigurációs állomány körül

A Samba hibakeresésénél fontos tudnunk, hogy az alapértelmezett opciók mindig előnyt élveznek. Ez azt jelenti, hogy ha egy alapértelmezett konfigurációs opció beállított állapotban van, az értéke nem változik, ha egyszerűen megjegyzésbe tesszük. Megpróbálhatjuk például a load printers opciót megjegyzésbe tenni és úgy kikapcsolni:

```
[printers]
path = /var/spool/samba
#load printers = yes
```

A *testparm* azonban megmutatja, hogy az opció értéke még mindig a `yes`:

```
ticktock samba # testparm -s -v |grep "load printers"
          load printers = Yes
```

A Samba esetén mindig explicit módon kell beállítanunk az opciókat, mert ha csak megjegyzésbe tesszük, a siker nem garantált. Ha valami nem úgy működik, ahogy vártuk, először ezt nézzük meg újra!

Bár magától értetődőnek tűnik, azért ellenőrizzük, hogy az *smbd* és az *nmbd* folyamatok léteznek. Előfordulhat, hogy valamilyen hatására észrevétlenül leállnak, és nem is tudjuk, hogy már nem futnak.

Gondok a felhasználói fiókok körül

A Samba leggyakoribb bejelentkezési hibái a root felhasználóval esnek meg, de a rendszer bármely felhasználója esetén előfordulhatnak. Ne feledjük, hogy minden Samba felhasználóhoz önálló jelszó szükséges, mert a Samba nem használja a Linux */etc/shadow* hash táblát. Ha például *root* felhasználóként megpróbálunk egy Samba állománymegosztáshoz hozzáférni, a rendszer *root* jelszava nem működik, kivéve ha a *root* fiókhoz ugyanazt a *root* jelszót rendeltük (ezt viszont nem tanácsoljuk). A hiba kijavításához önálló fiókot hozunk létre:

```
vlager# smbpasswd -a root
New SMB password:
Retype new SMB password:
Added user root.
```

17

OpenLDAP

Az OpenLDAP szabadon elérhető, nyílt forráskódú LDAP megvalósítás, amelyet különféle platformokra fordíthatunk le. Linuxon jelenleg a legelterjedtebb és legtámogatottabb ingyenes LDAP termék. Teljesítménye és szolgáltatásai tekintetében felveszi a versenyt sok más, a kereskedelemben kapható termékkel, de azoknál rugalmasabb, mert a forráskódja is elérhető és igény szerint testre szabható. Ebben a részben az OpenLDAP kiszolgáló lehetséges alkalmazásai mellett szó lesz a telepítéséről és a beállításáról is.

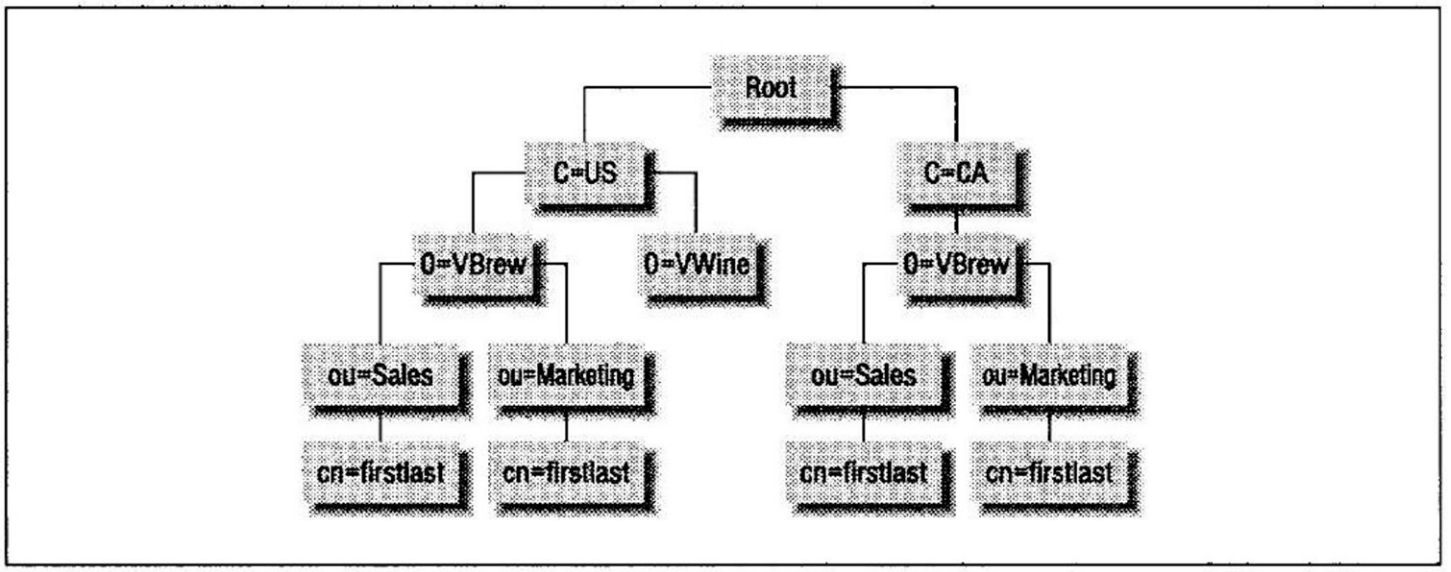
Az LDAP alapjai

Mielőtt tovább lépnénk, érdemes az LDAP-t röviden bemutatnunk. A *Lightweight Directory Access Protocol* (LDAP) olyan könyvtárszolgáltatás, amelynek segítségével szinte bármit tárolhatunk. Ebből a szemszögből nagyon hasonló egy adatbázishoz, noha arra tervezték, hogy kisebb adatmennyiségeket tároljunk benne, és a rekordok között hatékonyan kereshessünk. Az LDAP felhasználására jó példa a PKI környezet. Az ilyen jellegű környezet csupán minimális információt tárol és elérése nagyon gyors.

Az LDAP szerkezetét talán úgy szemléltethetjük legegyszerűbben, ha egy fához hasonlítjuk. Minden LDAP könyvtár gyökér bejegyzésből indul. Ebből a bejegyzésből további bejegyzések ágaznak el, amelyek mindegyike egy-egy információrészt képes tárolni. A 17.1. ábrán egy minta LDAP fát láthatunk.

Az LDAP és a szabályos adatbázisok közötti további alapvető különbség az LDAP együttműködő képessége. Az LDAP előre létrehozott sémákat vagy adategységeket használ, ezekből alakulnak ki az adott fák. Az X.500 szerkezetet az RFC 2253 vázolja, és a következő bejegyzéseket tartalmazza:

```
String  X.500 AttributeType
------  --------------------
CN      commonName
L       localityName
ST      stateOrProvinceName
O       organizationName
OU      organizationalUnitName
C       countryName
STREET  streetAddress
DC      domainComponent
UID     userid
```



17.1. ábra. Minta LDAP faszerkezet

Hasznos séma még az inetOrgPerson. A séma személyeket reprezentál egy szervezeti struktúrában, és például telefonszámokat, címeket, felhasználói azonosítókat vagy akár az alkalmazottak fényképeit tartalmazza értékként.

Az adatok elnevezésének szabályai

Az LDAP bejegyzéseket a könyvtárban *relatív megkülönböztetett névként* (Relative Distinguished Name, RDN) tároljuk, az egyes bejegyzésekre viszont *megkülönböztetett nevükkel* (Distinguished Name, DN) utalunk. A Bob Jones felhasználó RDN-je például lehet a

```
cn=BobJones
```

A felhasználó DN-je pedig így festhet:

```
c=us,st=California,o=VirtualBrewery,ou=Engineering,cn=BobJones
```

Az LDAP természetesen ennél jóval összetettebb, a példa mégis alkalmas arra, hogy segítségével bemutassuk az OpenLDAP telepítéséhez és működtetéséhez szükséges ismereteket. Az LDAP részletesebb leírását megtaláljuk az RFC 2251, „The Lightweight Directory Access Protocol (v3)" specifikációban.

Az OpenLDAP beszerzése

Az OpenLDAP projektnek jelenleg a *http://www.openldap.org* ad otthont. A webhelyről beszerezhetjük az összes aktuális stabil verziót, valamint az „Issue Tracking" („hibakövető") motort arra az esetre, ha hibával találkozunk és beszámolót szeretnénk róla küldeni a fejlesztőknek.

Noha a béta verziók letöltése és használata mindig csábító az ígéretes új szolgáltatások miatt, ha nem tesztelési célokra való kiszolgálóra szánjuk, akkor használjunk inkább közismerten stabil verziókat.

Miután letöltöttük és kibontottuk a forrás archívumát, általában érdemes vetni egy pillantást a benne található README állományokra. Az állományok olvasására szánt öt perc bőségesen megtérül, ha később a telepítéskor gondjaink akadnak.

A szükséges tartozékok

Mint oly sok más szoftvercsomag, az OpenLDAP működése is függ más szoftverektől. Az OpenLDAP futtatásához telepítenünk és konfigurálnunk kell az OpenSSL legújabb változatát. Ha még nem rendelkezünk vele, a telepítési útmutatóval együtt letölthetjük a *http://www.openssl.org* oldalról.

Az OpenLDAP-hoz szükségünk lesz a Cyrus SASL-re. Az *Egyszerű Hitelesítési és Biztonsági Réteg* (Simple Authentication and Security Layer, SASL) neve is jelzi, hogy a SASL egyszerűen használható biztonsági keretrendszert kínál. Sok Linux terjesztés a csomagot alapértelmezésben telepíti; ha azonban magunknak kell ezt megtennünk, a csomagot megtalálhatjuk a *http://asg.web.cmu.edu/sasl/sasl-library.html* címen, de megkereshetjük csomagkereső motorral is, például az RPMfind eszközzel.

Az OpenLDAP nem igényli, de lehetőséget ad a Kerberos támogatására. Ha rendszerünkben Kerberost használunk, az OpenLDAP kiszolgálóra telepítsük. Ha nem használunk Kerberost, nem sok értelme volna üzembe helyezni, különösen az OpenLDAP kiszolgálóhoz. A Kerberosról hatalmas mennyiségű dokumentációt találhatunk, ezek ismeretében pedig eldönthetjük, szükségünk van-e rá.

Az OpenLDAP további választható komponense a backend adatbázis (BDB). A BDB használatához szükségünk van például a Sleepycat Software BerkeleyDB csomagjára vagy egy hasonlóra. A csomagot széles körben használják és sok Linux terjesztésben alapértelmezett. Ha rendszerünkön nem található, letölthetjük a *http://www.sleepycat.com* oldalról. Választhatunk más BDB-t is, például a MySQL-t. A választás joga a miénk, de az átlagos felhasználónak az alapértelmezett OpenLDAP adatbázis is megfelel.

Az OpenLDAP fordítása

Ha már kiválasztottuk a beállítani kívánt opciókat, az OpenLDAP fordítása egyszerű feladat. Az elérhető opciók listáját a *configure* programmal kérhetjük le. Érdemes beállítani például a *-with-tls* opciót, amely az OpenLDAP SSL támogatását kapcsolja be – erről a fejezet egy későbbi részén szólunk.

```
vlager# ./configure -with-tls
Copyright 1998-2003 The OpenLDAP Foundation,  All Rights Reserved.
       Restrictions apply, see COPYRIGHT and LICENSE files.
Configuring OpenLDAP 2.1.22-Release ...
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking build system type... i686-pc-linux-gnu
checking for a BSD compatible install... /bin/install -c
checking whether build environment is sane... yes
```

```
checking for mawk... no
checking for gawk... gawk
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
checking for working makeinfo... found
checking for gnutar... no
checking for gtar... no
.
.
vlager#
```

A *makefile* beállítása után megkíséreljük lefordítani a csomagot. Ezt a legtöbb szoftver esetében a *make* parancsot futtatva végeznénk. Az OpenLDAP megépítésekor azonban érdemes először a *make depend* utasítást kiadnunk. Ha elfeledkeznénk róla, a konfigurációs szkript is figyelmeztet rá.

A függő állományok létrehozása után most már nyugodtan kiadhatjuk a make parancsot és várhatjuk, hogy megépüljön a szoftver. Amikor a folyamat befejeződött, ellenőrizzük, hogy az építés hibátlanul zajlott. A *make test* egy sor ellenőrzést hajt végre és tájékoztat bennünket az esetleges hibákról.

Feltéve, hogy minden rendben volt, root felhasználóként a make install opcióval automatikusan telepíthetjük a szoftvert. Alapértelmezés szerint az OpenLDAP az */usr/local/etc/openldap* útvonalra helyezi a konfigurációs állományait. Egyes rendszereken a következetesség érdekében az állományokat az */etc/openldapútvonalra* helyezhetjük. Ezt az opciót a *./configure* parancssorában adhatjuk meg.

Az OpenLDAP kiszolgáló beállítása

Ha figyelemmel kísértük a telepítést, észrevehettük, hogy két program jött létre: az *slapd* és a *slurpd*. Az OpenLDAP telepítés ezt a két daemont használja.

Az OpenLDAP kiszolgáló beállításának megértéséhez vessünk egy pillantást a konfigurációs állományokra. A **vlager** minta gazdagépen a konfigurációs szkripteket az */usr/local/etc/openldap* könyvtárba helyeztük. A *slapd.conf* állomány egyes részeit saját rendszerünkhöz kell igazítanunk. A következő értékeket szükséges a rendszerünkhöz állítanunk:

```
include         /usr/local/etc/openldap/schema/core.schema
include         /usr/local/etc/openldap/schema/cosine.schema
include         /usr/local/etc/openldap/schema/inetorgperson.schema

database        ldbm
suffix          "o=vbrew"
suffix          "dc=ldap,dc=vbrew,dc=com"
rootdn          "cn=JaneAdmin,o=vbrew"
rootpw          secret
directory       /usr/local/var/openldap-vbrew
defaultaccess   read
schemacheck     on
lastmod         on
```

Változtassuk meg a `rootpw` értékét, a `secret` helyett egy nekünk megfelelő jelszót adva meg. Ezt a jelszót használjuk az LDAP könyvtár megváltoztatásához.

A módosítások után most már futtathatjuk az OpenLDAP kiszolgálót; erről a következő részben olvashatunk.

Az OpenLDAP futtatása

Az önálló OpenLDAP kiszolgáló neve *slapd*, és ha az alapértelmezett útvonalakon nem változtattunk, akkor az */usr/local/libexec/* útvonalra telepítettük. A program egyszerűen a bejövő kapcsolatokat figyeli az LDAP porton (TCP 389), és a kérelmeket dolgozza fel. Mivel a folyamat egy fenntartott porton fut, root jogosultságokkal kell indítanunk. Ezt így tehetjük meg:

```
vlager# su root -c /usr/local/libexec/slapd
```

Ellenőrizzük, hogy a szolgáltatás valóban elindult: futtassuk a *netstat* programot, és nézzük meg, mi figyeli a 389-es portot:

```
vlager# netstat -aunt | grep 389
tcp        0      0 0.0.0.0:389             0.0.0.0:*           LISTEN
```

Amikor idáig eljutottunk, ellenőrizzük, hogy maga a szolgáltatás működik-e. Ehhez küldjünk egy lekérdezést az *ldapsearch* utasítással:

```
vlager# ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
version: 2

#
# filter: (objectclass=*)
# requesting: namingContexts
#

#
dn:
namingContexts: dc=vbrew,dc=com

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
vlager#
```

A lekérdezés a joker karakter miatt mindenre illeszkedik az adatbázisban, így a keresés az összes tárolt elemmel tér vissza. Ha a konfigurációval sikeresen végeztünk, a `dc` mezőben saját tartománynevünket láthatjuk viszont.

Bejegyzések megadása a könyvtárban

Miután LDAP kiszolgálónk már működik, érdemes volna néhány bejegyzést adnunk hozzá. Az OpenLDAP szoftverrel együtt kapjuk az *ldapadd* programot is, amely rekordokat képes beszúrni az LDAP adatbázisba. A program kizárólag az LDAP Data Interchange (LDIF) állományokból fogad el beszúrásokat, ezért először ezt az állományt hozzuk létre. Az LDIF állományok formátumáról az RFC 2849 specifikációban olvashatunk részletesen. Az LDIF állományok létrehozása szerencsére roppant egyszerű. Nézzünk meg most egy mintaállományt:

```
# A Virtuális Sörgyár szervezete
dn: dc=vbrew,dc=com
objectClass: dcObject
objectClass: organization
dc: example
o: VirtualBrew Corporation
description: The Virtual Brewery Corporation

# A könyvtármenedzser szervezeti feladata
dn: cn=Manager,dc=vbrew,dc=com
objectClass: organizationalRole
cn: Manager
description: Directory Manager

dn: dc=ldap,dc=vbrew,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
dc: vbrew
o: vbrew
description: Virtual Brewing Company LDAP Domain

dn: o=vbrew
objectClass: top
objectClass: organization
o: vbrew
description: Virtual Brewery

dn: cn=JaneAdmin,o=vbrew
objectClass: organizationalRole
cn: JaneAdmin
description: Linux System Admin Guru

dn: ou=Marketing,o=vbrew
ou: Marketing
objectClass: top
objectClass: organizationalUnit
description: The Marketing Department

dn: ou=Engineering,o=vbrew
ou: Engineering
objectClass: top
objectClass: organizationalUnit
description: Engineering team
```

```
dn: ou=Brewers,o=vbrew
ou: Brewers
objectClass: top
objectClass: orginazationalUnit
description: Brewing team

dn: cn=Joe Slick,ou=Marketing,o=vbrew
cn: Joe Slick
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
mail: jslick@vbrew.com
firstname: Joe
lasname: Slick
ou: Marketing
uid: 1001
postalAddress: 10 Westwood Lane
l: Chicago
st: IL
zipcode: 12394
phoneNumber: 312-555-1212

dn: cn=Mary Smith,ou=Engineering,o=vbrew
cn: Mary Smith
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
mail: msmith@vbrew.com
firstname: Mary
lasname: Smith
ou: Engineering
uid: 1002
postalAddress: 123 4th Street
l: San Francisco
st: CA
zipcode: 12312
phoneNumber: 415-555-1212

dn: cn=Bill Peris,ou=Brewing,o=vbrew
cn: Bill Peris
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
mail: per@vbrew.com
firstname: Bill
lasname: Peris
ou: Brewing
uid: 1003
postalAddress: 8181 Binary Blvd
l: New York
st: NY
zipcode: 12344
phoneNumber: 212-555-1212
```

Miután elkészültünk az LDIF állománnyal, a következő paranccsal adhatjuk az LDAP könyvtárhoz:

```
vlager# ldapadd -x -D "cn=Manager,dc=vbrew,dc=com" -W -f goo.ldif
Enter LDAP Password:
adding new entry "dc=vbrew,dc=com"

adding new entry "cn=Manager,dc=vbrew,dc=com"

vlager#
```

Most már a korábban bemutatott *ldapsearch* utasítással kereshetünk új könyvtárunk bejegyzései között.

Az OpenLDAP használata

Egy LDAP kiszolgálót rengeteg feladatra használhatunk – túl sokra ahhoz, hogy itt mind felsoroljuk. A Linux hálózati adminisztrátorok számára azonban az egyik leghasznosabb szolgáltatása a hitelesítés. A sok különböző számítógép mellett a náluk is több felhasználó jelszavainak és egyéb hitelesítéshez használt adatainak kezelése a rendszergazda számára kimerítő feladat. Az OpenLDAP könyvtár segítségével központilag kezelhetjük a felhasználói fiókokat egy több rendszerből álló csoportban, miközben gyorsan és hatékonyan kapcsolhatjuk be és ki a felhasználói fiókokat – ami egyébként a több rendszerrel dolgozó adminisztrátor számára mindennapos és fárasztó munka volna.

A szolgáltatás beállításához telepítsük az LDAP NSS és PAM könyvtárakat. Linux rendszeren az NSS és a PAM kezeli a hitelesítést és mondja meg a rendszernek, hol ellenőrizheti a felhasználókat. Két csomagot telepítünk, a *pam_ldap* és az *nss_ldap* csomagokat, amelyeket a *http://www.padl.com/OSS* oldal szoftverekkel foglalkozó részéről tölthetünk le. A legtöbb terjesztésben megtalálhatjuk a csomagokat, melyeket gyakran egyetlen állományban helyeznek el *libnss-ldap* néven. Ha a telepítést forrásból végezzük, a szoftver megépítése nem jelenthet gondot, és a szabályos *configure, make install* eljárással végezhető. A könyvtárakat nem csak az LDAP kiszolgálóra telepítjük, de az ügyfélgépekre is.

A könyvtárak telepítését követően beállíthatjuk az *slapd* OpenLDAP folyamatot. Hasonló módosításokra lesz szükségünk a *slapd.conf* állományban, mint korábban. A már beállított séma szekció megfelelő, az adatbázis szekcióban azonban néhány új definíciót helyezünk el:

```
#######################################################################
# ldbm database definitions
#######################################################################

database        ldbm
suffix          "o=vbrew,dc=com"
rootdn          "uid=root,ou=Engineering,o=vbrew,dc=com"
rootpw          secret
directory       /usr/local/etc/openldap/data
# A fenntartani kívánt indexek
```

```
index   objectClass,uid,uidNumber,gidNumber     eq
index   cn,mail,surname,givenname              eq,subinitial
```

A szekció létrehozza a könyvtár-definíciókat a felhasználói adatokat tároló szerkezethez.

A hozzáférés-szabályozási listák megadása

Mivel az ilyen típusú könyvtárt a névtelen felhasználók nem írhatják, érdemes valamilyen fajta hozzáférési listát (access control list, ACL) is készítenünk. Szerencsére az OpenLDAP leegyszerűsíti számunkra az ilyen jellegű szabályozást.

```
# Egyszerű hozzáférésszabályozási lista
#

access to dn=".*,ou=Engineering,o=vbrew,dc=com"
   attr=userPassword
 by self write
 by dn="uid=root,ou=Engineering,o=vbrew,dc=com" write
 by * auth

access to dn=".*,o=vbrew,dc=com"
 by self write
 by dn="uid=root,ou=Engineering,o=vbrew,dc=com" write
 by * read

access to dn=".*,o=vbrew,dc=com"
 by * read

defaultaccess read
```

A lista egyszerű „zárat" helyez a könyvtárra, és megnehezíti, hogy a felhasználók beleírjanak. A konfigurációt elkészítve biztonságosan elindíthatjuk (vagy újraindíthatjuk) az OpenLDAP daemont.

Átállás az LDAP hitelesítésre

A létrehozott üres könyvtár most már az adatok bevitelére és lekérdezésére várakozik. Az olyan rendszergazdák számára, akik több száz vagy több ezer felhasználói fiókot kezelnek számos számítógépen, a következő lépés, az adatbázis feltöltése a hitelesítésre szolgáló adatokkal első hallásra rémálomnak tűnhet. Szerencsére az OpenLDAP migrációs eszközei, amelyeket a *http://www.padl.com/OSS* címről szerezhetünk be, leegyszerűsítik az LDAP adatbázis feltöltését a már létező */etc/passwd* állományok alapján. A terjesztések vagy csomagok valószínűleg az */usr/share* könyvtárba helyezik ezeket az állományokat, de a részleteket ellenőrizzük csomagunk dokumentációjában.

Azért, hogy a szkriptek megfelelően működjenek, egyikükön apró változtatásokat végzünk: a *migrage_common.ph* szkriptet rendszerünkhöz igazítjuk. Keressük meg benne a következő sorokat:

```
#Default DNS domain
$DEFAULT_MAIL_DOMAIN = "padl.com";
```

```
#Default base
$DEFAULT_BASE = "dc=padl,dc=com";
```

és helyettesítsük az értékeket a rendszerünknek megfelelőkkel. Példánkban ez így festene:

```
$DEFAULT_MAIL_DOMAIN = "vbrew.com";
$DEFAULT_BASE = "o=vbrew,dc=com";
```

Győződjünk meg arról, hogy az OpenLDAP kiszolgálónk figyel, és hogy elmentettük a migrációs beállítási állományon végzett módosításokat. Amikor elkészültünk, futtathatjuk a *migrate_all_online.sh* szkriptet, amely elkezdi bemásolni az */etc/passwd* bejegyzéseket az LDAP könyvtárunkba.

```
vlager# ./migrate_all_online.sh
Enter the Name of the X.500 naming context you wish to import into:
                  [o=vbrew, dc=com]
Enter the name of your LDAP server [ldap]: vlager
Enter the manager DN: [cn=manager,o=vbrew,dc=com] cn=root,o=vbrew,dc=com
Enter the credentials to bind with: elszó
Importing into o=vbrew,dc=com...
Creating naming context entries...
Migrating aliases...
Migrating groups...
.
.
vlager#
```

Ezen a ponton láthatjuk, hogy az */etc/passwd* valamennyi bejegyzése automatikusan bekerült az LDAP könyvtárba. Az *ldapsearch* eszközzel megvizsgálhatunk pár bejegyzést. Most, hogy a könyvtárszolgáltatás üzemkész és feltöltöttük a felhasználókra vonatkozó bejegyzésekkel, állítsuk be az ügyfeleket az LDAP kiszolgáló lekérdezésére – erről szól a következő rész.

Az LDAP beállítása az ügyfeleken

A Linux terjesztések alapértelmezett beállításuk szerint az */etc/passwd* állományban keresik a hitelesítéshez szükséges adatokat. Az *nss-ldap* és a PAM könyvtárak telepítése után, amiről korábban már szó volt, ezt az alapértelmezett beállítást könnyedén megváltoztathatjuk. Először az */etc/nsswitch.conf* állományt módosítjuk. Csupán arra kell utasítanunk a rendszert, hogy a hitelesítési információkat az LDAP-tól kérje le.

```
passwd:     files ldap
group:      files ldap
shadow:     files ldap
```

Észrevehettük, hogy a konfigurációban meghagytuk a file bejegyzést. Érdemes így eljárnunk, mert ezzel az olyan felhasználók, mint például a *root* akkor is rendelkeznek hozzáféréssel, ha történik valami az LDAP kiszolgálóval. Ha kitöröljük a sort és az LDAP leáll, egyetlen rendszerünkbe sem tudunk bejutni! Természetesen ilyen helyzetben jól jön,

ha több kiszolgálónk is van. Az LDAP kiszolgálók közötti másolás lehetséges és nem is bonyolult feladat. A tartalék OpenLDAP kiszolgálók építéséről az *OpenLDAP HOWTO* leírásban olvashatunk; a leírás elérhető a Linux Dokumentációs Projekt webhelyén.

Egyes Linux terjesztések (például a Debian) az ügyfélkonfigurációt az */etc/openldap.conf* állományban tárolják. Legyünk óvatosak, ne keverjük össze a kiszolgáló konfigurációs állományaival az */etc/openldap* könyvtárban!

A következő módosítandó állomány az *openldap.conf*. A többi konfigurációs állományhoz hasonlóan ennek elhelyezkedése is eltérő az egyes terjesztésekben. Az állomány nagyon egyszerű és csupán pár beállítható opciót tartalmaz. Úgy módosítjuk őket, hogy megfeleljenek az LDAP kiszolgálónk URI címének és az alapvető LDAP adatoknak.

```
URI  ldap://vlager.vbrew.com
BASE o=vbrew,dc=com
```

Most kíséreljünk meg végrehajtani egy LDAP lekérdezést az egyik ügyfélgépünkről.

```
client$ ldapsearch -x 'uid=bob'
version: 2

#
# filter: uid=bob
# requesting: ALL
#

# bob,Engineering,vbrew,com
dn: uid=bob,ou=Engineering,o=vbrew,c=com
uid: bob
cn: bob
sn: bob
mail: bob@vbrew.com
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: account
objectClass: posixAccount
objectClass: top
objectClass: shadowAccount
shadowMax: 99999
shadowWarning: 7
loginShell: /bin/bash
uidNumber: 1003
gidNumber: 1003
homeDirectory: /home/bob
gecos: bob

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
client$
```

Ha a lekérdezés eredménye hasonlít az előbbi kimenetre, a lekérdezés működik. OpenLDAP kiszolgálónkat feltöltöttük adatokkal, amelyeket az ügyfélgépek képesek lekérdezni; most már valóban elkezdhetik a munkát.

SSL támogatás az OpenLDAP kiszolgálón

Mivel az LDAP tervezésénél fogva biztonságosan és hatékonyan képes kis adatmennyiségek kiszolgálására, alapértelmezés szerint egyszerű szöveget tartalmaz. Noha bizonyos célokra ez teljesen megfelel, előfordulhat, hogy szeretnénk az adatfolyamokat titkosítani. Így a könyvtár lekérdezéseit könnyebben elrejthetjük a kíváncsi szemek elől, és a támadók is nehezebben juthatnak információhoz a hálózatunkról. Ha az LDAP-t bármiféle hitelesítésre használjuk, erősen javasoljuk a titkosítást.

Amennyiben az OpenLDAP fordításakor megadtuk a `-with-tls` opciót, kiszolgálónk készen áll az SSL támogatására. Ha nem, a folytatáshoz ismét meg kell építenünk az OpenLDAP-t.

Az OpenLDAP SSL támogatásához csupán pár egyszerű módosítást kell végeznünk az OpenLDAP konfigurációs állományában. Adjuk meg, hogy melyik SSL rejtjelezést használja az OpenLDAP, illetve hogy hol találhatja az SSL tanúsítványokat és kulcsállományokat, amelyeket később hozunk majd létre.

```
TLSCipherSuite HIGH:MEDIUM:+SSLv3
TLSCertificateFile /etc/ssl/certs/slapd.pem
TLSCertificateKeyFile /etc/ssl/certs/slapd.pem
```

Esetünkben arról volt szó, hogy az OpenSSL tanúsítványokat az */etc/ssl* alapértelmezett könyvtárban szeretnénk tárolni. Ez az egyes terjesztésekben eltérő lehet, azonban az SSL tanúsítványokat tetszés szerinti könyvtárban is elhelyezhetjük. Megadtuk továbbá a használni kívánt rejtjelezés típusát. Vegyük észre, hogy számos lehetőséggel rendelkezünk, ezek közül kedvünk szerint választhatunk. Mivel a kiszolgáló az indulásakor most már várja a tanúsítványokat, létre kell hoznunk ezeket. A tanúsítványok létrehozásának menetéről a könyv korábbi részeiben, illetve az OpenSSL dokumentációban olvashatunk.

Fontos megértenünk, hogy az eljárással saját magunk írjuk alá a tanúsítványokat, szemben azokkal, amelyeket a tanúsítványszolgáltatóktól vásárolhatunk. Ha egy LDAP ügyfél ellenőrzi a tanúsítványunk érvényességét, valószínűleg hibát jelez; hibát jeleznek például a webböngészők, ha olyan tanúsítvánnyal találkoznak, amelyet nem egy harmadik fél írt alá. Az LDAP ügyfelek többsége azonban nem ellenőrzi a tanúsítványok érvényességét, így feltehetően ezzel a kérdéssel nem kell sokat foglalkoznunk. Amennyiben egy harmadik fél által aláírt tanúsítványra van szükségünk, számos szolgáltatótól beszerezhetjük azt.

A kiszolgáló újraindítása előtt győződjünk meg róla, hogy az LDAP kiszolgálónk képes a tanúsítványokat olvasni. A rendszer más felhasználói számára azonban ezek ne legyenek elérhetők és írhatók. Mielőtt továbblépünk, távolítsuk el az ideiglenes állományokat is, például az előző lépésben létrehozottakat. Ezután biztonságosan újraindíthatjuk a kiszolgálót.

Az SSL elérhetőségének vizsgálata

Az SSL támogatás elérhetőségének vizsgálatára sok gyors eljárás létezik. A legegyszerűbb, ha az előző példához hasonlóan a *netstat* programmal megnézzük, figyel-e a kiszolgáló:

```
vlager# netstat -aunt | grep 636
tcp        0      0 0.0.0.0:636             0.0.0.0:*               LISTEN
```

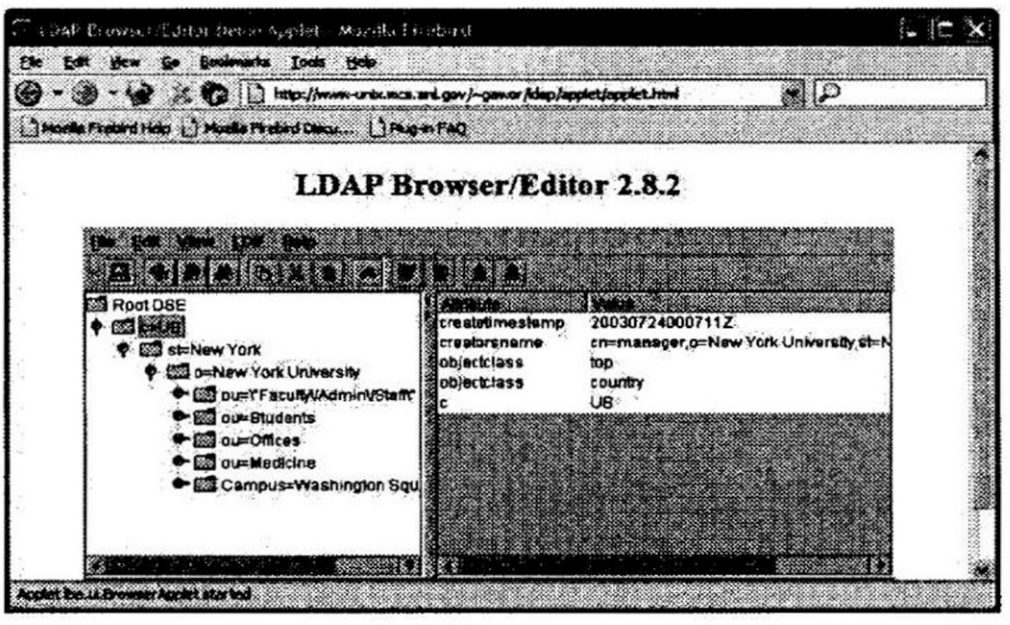
Első lépésnek megfelel, ha látjuk, hogy a kiszolgáló figyel. Ez azt jelenti, hogy megértette az SSL futtatására vonatkozó kérést és figyeli a megfelelő portot. Most megnézhetjük, hogy a folyamat valójában működik-e. Ezt is könnyen megtehetjük, ha utasítjuk, hogy mutassa meg a digitális tanúsítványokat. Az OpenSSL csomaggal érkező ügyfelet használhatjuk erre:

```
vlager# openssl s_client -connect vlager:636 -showcerts
```

A parancs kimenetében láthatjuk a létrehozott digitális tanúsítványok teljes technikai leírását. Ha a parancsból egyáltalán nem kapunk kimenetet, ellenőrizzük, hogy elindították-e a szolgáltatást, és hogy valami figyel-e a 636-os porton. Ha továbbra sem kapunk választ, olvassuk el a fejezet hibakeresésről szóló részét.

LDAP GUI böngészők

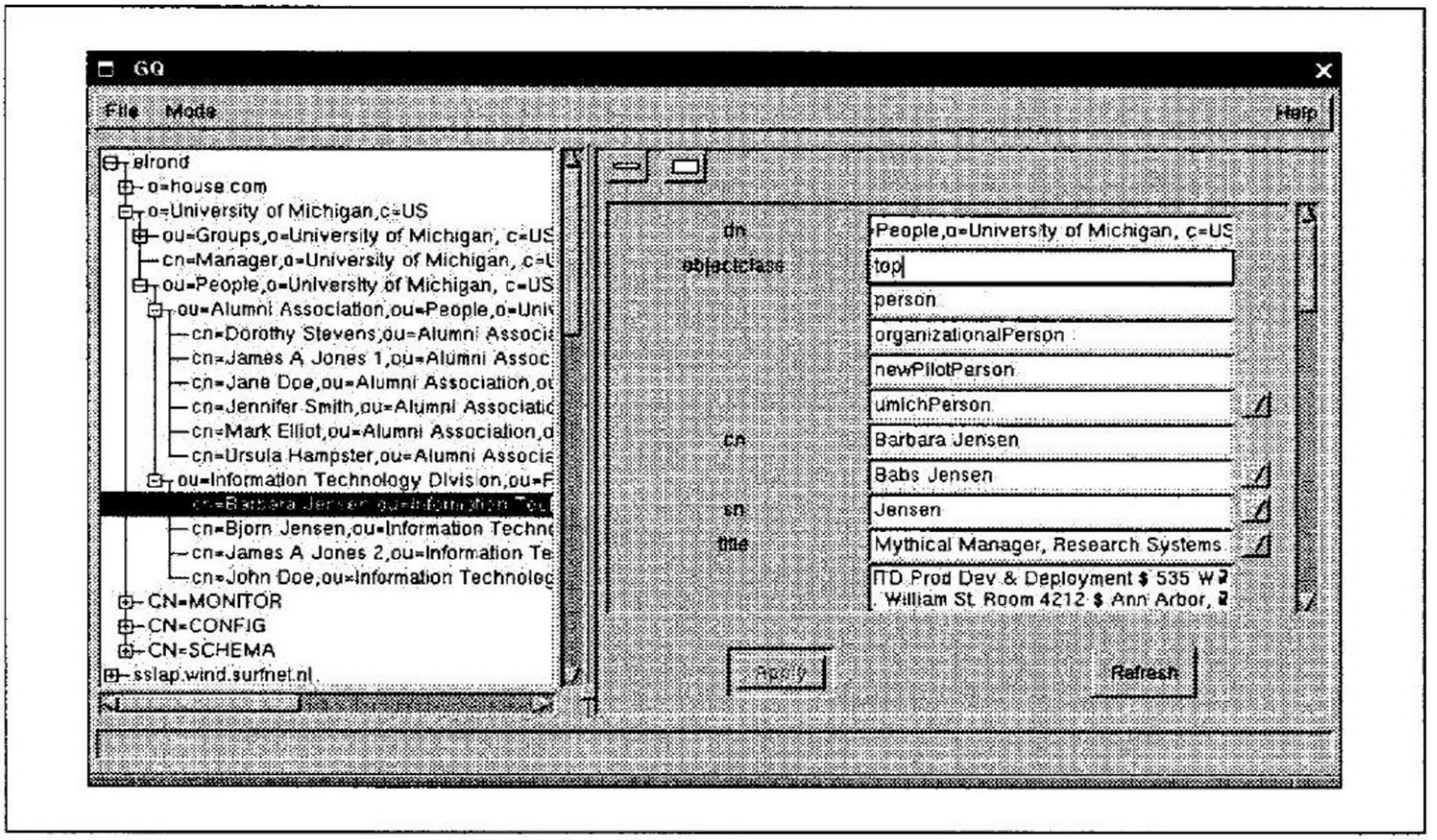
A Linux számos GUI LDAP böngészőt kínál a grafikus felhasználói felületet a parancssorral szemben előnyben részesítő rendszergazdáknak. Megfelelő választás lehet például az Argonne National Laboratory böngészője, amely egy Java applet, és a *http://www-unix.mcs.anl.gov/~gawor/ldap/applet/applet.html* címről tölthető le. Ahogy a 17.2. áb-



17.2. ábra. Java LDAP böngésző

rán láthatjuk, a GUI segítségével könnyedén böngészhetünk a könyvtárakban és módosíthatjuk a bejegyzéseket. A szoftver másik előnye, hogy bármely platformon egyszerűen és a telepítéssel járó bonyodalmak nélkül futtatható, amennyiben a Java elérhető a rendszeren.

A Linuxnak egy másik népszerű és rendkívül hatékony LDAP beviteli felülete a 17.3. ábrán látható GQ. A böngésző GTK+ -szerű felületet használ és a rendszergazdáknak teljes szabadságot biztosít a könyvtárak kezelésében, lehetővé téve az bejegyzések hozzáadását, módosítását és eltávolítását, sablonok készítését, részletes keresések futtatását és még sok egyebet. A GQ letölthető a *http://www.biot.com/gq* oldalról és Linux GTK+ -kompatibilis interfészt igényel.



17.3. ábra. A GQ böngészőmódban

Az OpenLDAP hibakeresése

Az OpenLDAP telepítése és beállítása összetett feladat, amelyben rengeteg hibát ejthetünk. Mint más Linux kiszolgálói folyamatok esetében is, a hibakeresést érdemes a rendszernaplóval kezdeni. A rendszernaplóban találnunk kell a következőhöz hasonló sort, amely mutatja, hogy a kiszolgáló rendben elindult-e.

```
Jun 15 11:33:39 vlager slapd[1323]: slapd starting
```

Ha a *slapd.conf* állományban elgépeltünk valamit, a folyamat gyakran elég értelmes ahhoz, hogy figyelmen kívül hagyja a sort és folytassa a futását. Ha azonban kritikus hibát ejtünk, mint például a következő példában, a *slapd* nem képes elindulni.

```
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 46:
unknown directive "odatabase" outside backend info and database
definitions (ignored)
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 47:
suffix line must appear inside a database definition (ignored)
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 49:
rootdn line must appear inside a database definition (ignored)
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 54:
rootpw line must appear inside a database definition (ignored)
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 57:
unknown directive "directory" outside backend info and database
definitions (ignored)
Jul 25 11:45:25 vlager slapd[10872]: /etc/openldap/slapd.conf: line 59:
unknown directive "index" outside backend info and database
definitions (ignored)
Jul 25 11:45:25 vlager slapd[10873]: backend_startup: 0 databases
to startup.
Jul 25 11:45:25 vlager slapd[10873]: slapd stopped.
Jul 25 11:45:25 vlager slapd[10873]: connections_destroy: nothing
to destroy.
```

A példában elgépeltük a „database" szót, amely a konfiguráció kritikus opciója. Láthatjuk, hogy a kiszolgáló képtelen elindulni, de elegendő információt biztosít ahhoz, hogy rájöhessünk, mi a hiba oka.

Ha ellenőriztük, hogy minden rendben fut-e a kiszolgálón, de az ügyfelek mégsem tudnak kapcsolódni, nézzük meg, hogy a 389-es és 636-os portot nem blokkolja-e a tűzfal. Amennyiben kiszolgálónk az *iptables* tűzfalat futtatja és az alapértelmezett eljárás az összes bejövő kapcsolat elutasítása, a két portot explicit módon kell engedélyeznünk.

Gyakran hibaforrás a hiányzó vagy hiányos SSL tanúsítvány. A rendszernapló jelzi, ha az SSL szolgáltatással gondok vannak:

```
Jul 25 12:02:15 vlager slapd[11135]: main: TLS init def ctx failed: 0
Jul 25 12:02:15 vlager slapd[11135]: slapd stopped.
Jul 25 12:02:15 vlager slapd[11135]: connections_destroy: nothing
to destroy.
```

A hibajelentés rendkívül szűkszavú ugyan, de a TSL említése arra utal, hogy az SSL a ludas. Ha ezt a hibát észleljük, ellenőrizzük a tanúsítványok útvonalát és a jogosultságokat. Az LDAP kiszolgáló gyakran azért nem képes a tanúsítványokat olvasni, mert azokat úgy állítottuk be, hogy csak a root felhasználó olvashassa őket.

18

Vezeték nélküli hálózatok

A vezeték nélküli hálózatkezelés sokat ígérő és egyre népszerűbb eljárás, amelynek a hagyományos vezetékes technológiával szemben számos előnye van, a felhasználók nagyobb kényelmétől az alacsonyabb megvalósítási költségeken keresztül a hálózatok kiépítésének egyszerűségéig. Egy új vezeték nélküli rendszer kiépítésével jelentős költségeket takaríthatunk meg, hiszen nincs szükségünk kiegészítő kábelekre, csatlakozókra és hálózati kapcsolókra. Az új felhasználókat egyszerűen hozzáköthetjük a hálózathoz, mindössze egy hálózati kártyát kell elhelyeznünk a számítógépünkben, és máris működik a kapcsolat. A vezeték nélküli eljárást arra is használhatjuk, hogy a hálózati hozzáférést megteremtsük a hagyományos hálózati infrastruktúrával rosszul ellátott vagy a teljesen ellátatlan területeken.

A vezeték nélküli hálózatkezelés hatását a legjobban azon mérhetjük le, hogy a fogyasztók mennyire fogadták el. Népszerűségének nyilvánvaló példája, hogy a laptop számítógépek csaknem mindegyéke beépített 801.11b vagy g kártyával kerül forgalomba. A gyakorlati előnyök miatt az ilyen laptopok kelendők, így a gyártók csökkenthetik áraikat.

A vezeték nélküli technikának azonban előnyei mellett hátrányai is vannak, ezek közül a legfontosabb a biztonság kérdése.

Történeti áttekintés

A vezeték nélküli LAN hálózatok a szórt spektrumú eljáráson alapulnak, amelyet eredetileg az Egyesült Államok hadserege fejlesztett ki a második világháborúban katonai kommunikációs célokra. A hadsereg mérnökei azért tartották a szórt spektrumot kívánatosnak, mert ellenállóbb volt a zavarással szemben. Ezzel egyidejűleg más területek fejlődése lehetővé tette a rádiós adatátvitel sebességének növelését. 1945 után kereskedelmi vállalkozások kezdték fejleszteni az eljárást, miután felismerték a benne rejlő potenciális lehetőségeket a felhasználók számára.

A szórt spektrumú eljárás a Hawaii Egyetem ALOHAnet* projektje révén vált a korszerű vezeték nélküli LAN elődjévé 1971-ben. A projekt lehetővé tette, hogy hét, különbö-

* A nevet sajnos sokszor elírják, az interneten is gyakori a rossz írásmód (AlohNet, AlohNET stb.). A hivatalos forma az ALOHAnet. Az eljárás egyik tervezőjének cikke megtalálható a www.eng.tau.ac.il/~meir/advcn/abramson_ALOHA.pdf címen. Olvashatunk róla a webes enciklopédiában is, lelőhelye: http://encyclopedia.lockergnome.com/s/b/ALOHA_network. Mellesleg az Aloha hawaii szó, a maori változata „aroha" és üdvözlésre használják, körülbelül annyit tesz: szeretlek – *a lektor.*

ző szigeteken üzemelő számítógép kétirányú kommunikációt valósítson meg az Oahu szigeten létesített központi jelelosztón keresztül.

Az ALOHAnettel kapcsolatos egyetemi kutatások vezettek a korszerű vezeték nélküli eszközök első generációjához, amelyek a 901–928 MHz frekvenciatartományban üzemeltek. Ezeket elsősorban a hadsereg alkalmazta, ezért a fejlesztésnek ebben a szakaszában a fogyasztók csak korlátozottan használhatták, mert a frekvencia túlságosan szűk és aránylag lassú volt.

Ekkor a 2,4 GHz frekvenciát jelölték ki a hatósági engedélyt nem igénylő felhasználásra, így a vezeték nélküli technológia ebben a tartományban fejlődött tovább, és létrejött a 802.11 szabvány. A szabványból alakult ki a széles körben elfogadott 802.11b szabvány, amely ma is fejlődik, a specifikáció egyre gyorsabb és biztonságosabb megvalósításával.

A szabványok

A személyi számítógépek vezeték nélküli hálózatkezelésének szabványait a Villamos és Elektronikai Mérnökök Testülete (Institute of Electrical and Electronics Engineers, IEEE) alakította ki. A LAN/MAN technológia átfogó jelölése a 802, amelyet munkacsoportokra bontottak le. A legaktívabb vezeték nélküli munkacsoporthoz tartozik a 802.15, amelyet vezeték nélküli személyes hálózatokhoz (Bluetooth) terveztek; a 802.16, amely a széles sávú vezeték nélküli rendszerek támogatását szolgálja; és végül a 802.11, amely a vezeték nélküli LAN eljárást jelöli. A 802.11 szabványon belül több különleges szabvány is létezik, ezeket betűk jelölik. Lássuk most a legfontosabb 802.11 vezeték nélküli LAN szabványokat:

802.11a
: A szabvány az 5 GHz-es hullámsávban biztosít vezeték nélküli hozzáférést. 54 MBps maximális sebességről gondoskodik, azonban nem terjedt el, talán a rövid hatótávolsága és az eszközök aránylag magasabb ára miatt.

802.11b
: A vezeték nélküli hálózatokról szólva a legtöbben még mindig erre a szabványra gondolnak. A szabvány a 2,4 GHz-es hullámsávban 11 MBps sebességet tesz lehetővé, hatótávolsága pedig elérheti az 500 métert is.

802.11g
: A szabvány célja, hogy a 2,4 GHz-es hullámsávban magasabb adatátviteli sebességet biztosítson, illetve a WiFi Protected Access (WPA) segítségével nagyobb biztonságot teremtsen. A 802.11g eszközök lassan kiváltják a 802.11b eszközöket, és széles körben is egyre inkább elfogadottá válnak.

802.11i
: A még fejlesztés alatt álló szabvány a 802.11b szabványt sújtó számos biztonsági hiányosságot kívánja kiküszöbölni, megbízhatóbb hitelesítési és titkosítási rendszert biztosítva. A könyv írásakor a szabvány még nem nyerte el végleges formáját.

802.11n

A 802.11n a jelenlegi vezeték nélküli hálózatok sebességkorlátját igyekszik áttörni. A 100 MBps működési sebessége nagyjából a duplája a létező vezeték nélküli átviteli sebességeknek, ugyanakkor visszafelé kompatibilis a b és g szabványokkal. A könyv írásának idején a szabvány még nem teljes, egyes forgalmazók azonban a specifikáció korai vázlatai alapján már kiadtak „pre-n" termékeket.

A 802.11b biztonsági kérdései

A 802.11b szabvány létrehozásakor az IEEE felismerte, hogy a vezeték nélküli hálózatkezelés nyitott jellegéből adódóan szükség van valamilyen fajta adatintegritásra és védelmi mechanizmusra, ezért létrehoztak egy újabb, a vezetékessel egyenértékű személyes titkosítást (Wired Equivalent Privacy, WEP). A 128 bites tikosítást ígérő szabvánnyal a felhasználók a hagyományos vezetékes hálózatok biztonságát élvezhették volna.

A biztonsághoz fűzött remények azonban hamarosan összeomlottak. Scott Fluhrer, Itsik Mantin és Adi Shamir a „Weaknesses in the Key Scheduling Algorithm of RC4" (Az RC4 kulcskezelő algoritmusának gyenge pontjai) című írásukban felfedték a WEP kulcsok előállításának és alkalmazásának hiányosságait. Bár amikor az írás készült, ez még csupán elméletileg jelentett támadási lehetőséget, a Rice Egyetem egyik diákja, Adam Stubblefield hamarosan a gyakorlatba is átültette, létrehozva az első WEP támadást. Noha az eszközeit soha nem tárta a nyilvánosság elé, ma már sok hasonló eszköz létezik a Linux rendszerhez a WEP feltörésére, így többé már nem tekinthetjük megbízhatónak.

Tudnunk kell azonban, hogy egy WEP támadás sok időt igényel. A támadás sikere attól függ, hogy a támadó milyen mennyiségű titkosított adatot képes megszerezni. Az olyan eszközök, mint például az AirSnort körülbelül 5-10 millió titkosított csomagot igényelnek. Egy forgalmas vezeték nélküli LAN hálózaton, ahol folyamatosan a lehető legnagyobb forgalom bonyolódik, a rendszer feltörése 10 órát is igénybe vehet. Mivel a legtöbb hálózat nem üzemel teljes kapacitással ilyen hosszú ideig, a támadás valószínűleg több időt igényelne, kisebb hálózatokon akár napokat is.

A rosszindulatú támadások és a lehallgatás elleni valódi védelemhez azonban mégis VPN eljárás szükséges, és a vezeték nélküli hálózatokat soha nem szabad közvetlenül a belső, megbízható hálózatokhoz kapcsolni.

A hardver

A 802.11b funkcionalitást a különböző gyártók némileg eltérő architektúrával valósítják meg. Két jelentős lapkakészletgyártó létezik, a Hermes és a Prism, és e kettőn belül is a gyártók módosításokat végeztek a biztonság, illetve a sebesség növelése érdekében. A USRobotics Prism lapkakészleten alapuló eszközei 802.11b funkcionalitást kínálnak 22 MBps sebesség mellett, ezt azonban nem érik el a DLink 802.11b funkcionalitású, 22 MBps sebességű hardvere nélkül. A kettő viszont csak 11 MBps sebesség mellett képes együttműködni.

802.11g és 802.11b a Linux rendszeren

Az új lapkakészletek és a gyártók közötti különbségek miatt a 802.11g támogatása a Linux rendszeren meglehetősen bonyolult feladat. A könyv írásakor a 802.11g eszközök tá-

mogatása még fejlesztés alatt áll, és egyelőre nem olyan stabil és megbízható, mint a 802.11b. Ezért a fejezetben a 802.11b illesztőprogramokkal és támogatással foglalkozunk. A g típusú eszközök teljes támogatása azonban már a közeli jövőben megvalósulhat. A g illesztőprogramokat fejlesztő csoportok, például a Prism54.org munkájával és az Intel bejelentésével, hogy Centrino lapkakészletükhöz illesztőprogramokat adnak ki, a teljes támogatás egy éven belül megvalósulhat.

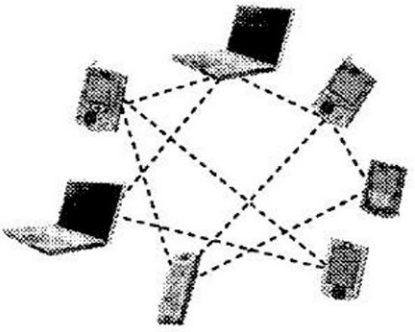
A lapkakészletek

Említettük, hogy két jelentős 802.11b lapkakészlet létezik, a Hermes és a Prism. Kezdetben a Hermes kártyák terjedtek el jobban, azonban a Lucent WaveLAN (Orinoco) kártyák népszerűségének köszönhetően a kártyák többsége a Prism prism2 lapkakészletét használja. Jól ismert Prism kártyák a D-Link, a Linksys és az USR termékei. A kártyák teljesítménye nagyjából azonos, és a 802.11b szabvány keretein belül képesek együttműködni, vagyis egy Lucent vezeték nélküli kártya képes egy D-Link hozzáférési pontra kapcsolódni, és fordítva. Lássuk most a nagyobb kártyagyártók és lapkakészleteik listáját. Ha a saját kártyánkat nem látjuk a listában, a használati útmutatóban, illetve a forgalmazó webhelyén tájékozódhatunk.

- Hermes lapkakészletet alkalmazó kártyák:
  Lucent Orinoco Silver és Gold kártyák
  Gateway Solo
  Buffalo Technologies
- Prism 2 lapkakészletet alkalmazó kártyák:
  Addtron
  Belkin
  Linksys
  D-Link
  ZoomMax

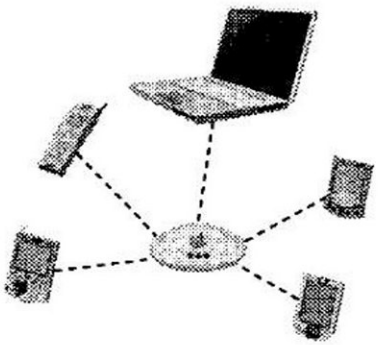
Az ügyfél beállítása

A 802.11b hálózatokat több különböző üzemmódra állíthatjuk. A két fő üzemmód, amelyekkel valószínűleg találkozni fogunk, az infrastrukturális üzemmód (néha menedzselt üzemmódnak is nevezik) és az ad hoc üzemmód. Az infrastrukturális üzemmód gyakoribb, és úgynevezett hub-and-spoke (tengely és küllők) architektúrát alkalmaz, ahol több ügyfél kapcsolódik egy központi hozzáférési pontra, mint a 18.1. ábra is mutatja. Az ad hoc vezeték nélküli hálózati üzemmód peer-to-peer hálózat, ahol az ügyfelek a 18.2. ábrán látható módon közvetlenül kapcsolódnak egymáshoz. Az infrastrukturális üzemmódú kiépítések hatékonyan helyettesíthetik a hagyományos hálózatok vezetékeit, így ideálisak a hivatali környezetben, ahol a vezeték nélküli ügyfeleknek a vezetékes hálózatra kapcsolódó kiszolgálókat kell elérniük. Az ad hoc hálózatok azok számára jelentenek előnyt, akik csupán állományokat szeretnének a számítógépek között mozgatni, és a vezeték nélküli hálózaton kívüli kiszolgálókra nem kívánnak kapcsolódni.



Hub-and-spoke
wireless network

18.1. ábra. A hub-and-spoke vezeték nélküli hálózat



Ad-hoc (peer-to-peer)
wireless network

18.2. ábra. Az ad hoc (peer-to-peer) vezeték nélküli hálózat

A 802.11b hálózatok előre meghatározott frekvenciakészleten működnek, ezeket csatornának nevezzük. A szabvány 14 önálló csatornát biztosít, noha az észak-amerikai felhasználók csak az első 11-et használhatják, az Európai felhasználók viszont az első 13-at. Csak Japánban van lehetőség valamennyi csatorna alkalmazására. Észak-Amerikában a 11 csatorna a 2400–2483 MHz tartományt öleli fel, és minden csatorna 22 MHz széles. A csatornák átfedik egymást, így fontos, hogy a csatorna kiválasztása előtt felmérjük a helyszínt, mert így elkerülhetjük a interferenciát más vezeték nélküli hálózatokkal, és sok fejfájástól kímélhetjük meg magunkat.

A fejezetben elsősorban a legelterjedtebb, a hub-and-spoke modellen alapuló infrastrukturális üzemmóddal foglalkozunk. A rendszer „tengelye", középpontja (hub) a hozzáférési pont, a küllők (spoke) pedig az ügyfelek. A hozzáférési pont lehet egy boltban megvásárolt dobozolt egység vagy a később tárgyalt HostAP-t futtató linuxos számítógép.

A 802.11b hálózatokban szükségünk van egy hozzáférési pontra, amely jeleket sugároz; a jel neve *beacon.* Ez lényegében olyan üzenet, amely tájékoztatja a közelben lévő ügyfeleket arról, hogy készen áll a kapcsolatok fogadására. A beacon korlátozott meny-

nyiségű információt hordoz, mindössze annyit, amennyire az ügyfélnek a kapcsolódáshoz szüksége van. A beacon legfontosabb tartalma az ESSID, vagyis a hozzáférési pont neve. Amikor az ügyfél észleli a hozzáférési pontot, kapcsolódási kérelmet küld. Ha a hozzáférési pont a kapcsolatot engedélyezi, akkor egy kapcsolódást biztosító kerettel (associate frame) válaszol az ügyfélnek, és létrejön a kapcsolat. Ezen a ponton más típusú hitelesítést is végezhetünk. Egyes hozzáférési pontok az előre megadott MAC című ügyfeleket engedélyezik, mások viszont további hitelesítést igényelnek. A fejlesztők dolgoznak a 802.1x szabványon, amely megfelelő hitelesítési keretrendszerről gondoskodna. Sajnos ez valószínűleg nem lesz kellőképen hatékony, mert már most ismertek bizonyos gyenge pontjai.

Az illesztőprogramok

A Linux vezeték nélküli illesztőprogramokat vagy a kernelbe építjük a fordításkor, vagy *betölthető kernel modulként* (loadable kernel module, LKM) hozzuk létre. Ajánlott kernel modult készíteni, mivel az illesztőprogramokat esetleg gyakran frissítenünk kell majd. Egy új modul megépítése egyszerűbb és kevesebb időt igényel, mint egy új kernel megépítése. A vezeték nélküli bővítményeket mindkét esetben be kell kapcsolnunk a kernel konfigurációjában. A legtöbb terjesztésben ezek már bekapcsolt állapotban vannak, ha azonban mindent magunk végzünk, a konfiguráció így néz ki:

```
# A betölthető modulok támogatása
#
CONFIG_MODULES=y
# CONFIG_MODVERSIONS nincs beállítva
CONFIG_KMOD=y
```

Vegyük észre, hogy a `MODVERSIONS` opciót kikapcsoltuk, mert így egyszerűbben fordíthatjuk az olyan modulokat, amelyek nem tartoznak a kernel modulfához. Nem kötelező ugyan, de ezzel időt takaríthatunk meg, amikor a kernelmodulokat frissítjük és újrafordítjuk. Függetlenül attól, hogy kártyánk milyen lapkakészletet használ, a PC kártyák esetén – de még egyes PCI kártyák esetén is – a PCMCIA kártyakezelő, a pcmcia-cs észleli a kártyánkat és telepíti a megfelelő illesztőprogramot. Több illesztőprogram is elérhető, jelenleg azonban csak kettőt tartanak fenn aktívan.

David Gibson Orinoco_cs illesztőprogramjait tartják általában a Hermes kártyák legjobb meghajtóprogramjainak, ezeket aktívan fejlesztik és javítják, és a legtöbb vezeték nélküli alkalmazással együttműködnek. Az Orinoco_cs illesztőprogramok a 2.4.3 verziótól a Linux kernel részei, és a 3.1.30 verziótól a pcmcia_cs is tartalmazza. Ha a terjesztésünkkel kapott illesztőprogram régebbi, szükség szerint frissíthetjük.

Ne zavarjon meg bennünket, hogy az Orinoco_cs illesztőprogramok ma már egyes prism2 kártyákat is támogatnak. A támogatott kártyák száma minden kibocsátással növekszik, ezért a meghajtó – neve ellenére – lassan a prism2 felhasználók számára is megbízható választás lehet, és a jövőben szabvánnyá is válhat.

Ha azonban az Orinoco_cs nem támogatja Prism kártyánkat, az Absolute Value Systems linux-wlan-ng illesztőprogramját szintén igénybe vehetjük. Pillanatnyilag ez a legjobban ismert és fenntartott ügyfél-illesztőprogram a lapkakészlethez. A legtöbb Linux terjesztés ezt is tartalmazza. Az illesztőprogram támogatja a Prism 2.x és 3.0 verziójú PCI, USB és PCMCIA kártyákat.

Miután a választott illesztőprogramot telepítettük és minden megfelelően működik, telepítjük a Linux Wireless Extension Tools készletet, amely felbecsülhetetlen értékű eszközgyűjtemény. Szerzőjük Jean Tourrilhes, és letölthetők a szerző webhelyéről, a *http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html* oldalról.

A Linux Wireless Extension Tools használata

A Linux Wireless Extension Tools roppant hasznos a vezeték nélküli hálózati eszközök beállításának minden területén. Ha meg szeretnénk változtatni az alapértelmezett vezeték nélküli beállításokat, vagy ha egyszerűen szeretnénk ad hoc hálózatokat beállítani, érdemes megismerkednünk ezekkel az eszközökkel. Szükségünk lesz rájuk a linuxos hozzáférési pont létrehozásához is, amiről a fejezet későbbi részében lesz szó.

Az eszközkészlet a következő programokat tartalmazza:

iwconfig
: A vezeték nélküli hálózatkezelés alapvető beállítási eszköze. A konfiguráció valamennyi aspektusát módosíthatjuk a segítségével, például az ESSID azonosítót, a csatornát, a frekvenciát és a WEP kulcskezelést.

twlist
: A program egy adott interfész elérhető csatornáit, frekvenciáit, bitrátáit és egyéb adatait jeleníti meg.

iwspy
: A program az egyes csomópontok kapcsolatának minőségéről gyűjt információt, és a kapcsolatok hibakeresésénél lehet hasznos.

iwpriv
: A programmal manipulálhatjuk az illesztőprogramunk különleges paramétereit. Ha például az Orinoco_cs illesztőprogram egy módosított verzióját használjuk, az *iwpriv* segítségével lehallgató (promiscuous) üzemmódba helyezhetjük azt.

A linuxos hozzáférési pont beállítása

A Linux vezeték nélküli fegyvertárának egyik kiemelkedően hasznos darabja a HostAP, egy vezeték nélküli illesztőprogram, amelyet Jouni Malinen készített. A program lehetővé teszi, hogy a prism2 kártyával rendelkező felhasználók vezeték nélküli kártyájukból és Linux kiszolgálójukból hozzáférési pontot alakítsanak ki. Mivel a piacon számos olcsó hozzáférési pont kapható, felmerülhet a kérdés: miért kellene átalakítanunk kiszolgálónkat hozzáférési ponttá? A válasz az elérhető szolgáltatásokban keresendő. A legtöbb olcsó, kimondottan erre a célra szánt hozzáférési pont a vezeték nélküli hálózat kiszolgálásán kívül kevés más szolgáltatással rendelkezik. Kevés beleszólásunk van a hozzáférés szabályozásába és a tűzfalkezelésbe. A Linux alapú hozzáférési pontokban rendelkezésünkre áll a Netfilter, a RADIUS és a MAC hitelesítés, illetve bármely más linuxos szoftver, amelyre igényt tartunk.

A HostAP illesztőprogram telepítése

A HostAP telepítéséhez rendszerünkön a következők szükségesek:

- A Linux Kernel v2.4.20 vagy újabb változata (a 2.4.29 verzióhoz tartozó kernel javítóprogrammal);
- a Wireless Extension eszközkészlet;
- a legújabb HostAP illesztőprogram, amely letölthető a *http://hostap.epitest.fi/releases* címről.

A HostAP illesztőprogram beszerzése és megépítése

Bár RPM és .deb csomagok is elérhetők lehetnek, valószínűleg forrásból kell a HostAP-t megépítenünk ahhoz, hogy a legfrissebb verziót telepíthessük. Bontsuk ki az archív állományt egy munkakönyvtárba. A HostAP-nek szüksége van a Linux kernel forráskódjára, amelyet az */usr/src/linux* útvonalon keres. Egyes terjesztések, például a Red Hat a kernel forrását az */usr/src/linux-2.4* útvonalra helyezi. Ilyen esetben készítsünk egy szimbolikus csatolást *linux* néven, amely a kernel forrásának könyvtárára mutat. A forrás előkészítése a telepítéshez egyszerű, és a következő módon hajthatjuk végre:

```
[root@localhost root]# tar xzvf hostap-0.0.1.tar.gz
hostap-0.0.1/
hostap-0.0.1/COPYING
hostap-0.0.1/ChangeLog
hostap-0.0.1/FAQ
.
.
hostap-0.0.1/Makefile
hostap-0.0.1/README
hostap-0.0.1/utils/util.h
hostap-0.0.1/utils/wireless_copy.h
```

Sok más csomagtól eltérően a HostAP nem rendelkezik a forrás megépítése előtt futtatandó konfigurációs szkripttel. Meg kell azonban választanunk, milyen modulokat szeretnénk megépíteni. A HostAP jelenleg a PCMCIA, PLX és PCI eszközöket támogatja. Az USB eszközök nem kompatibilisek, de a jövőben ezek támogatása is elképzelhető. A példában a PC Card változatot építjük meg.

```
[root@localhost root]# make pccard
gcc -I/usr/src/linux/include  -O2 -D_ _KERNEL_ _ -DMODULE -Wall -g -c  -
I/usr/src/linux/arch/i386/mach-generic -I/usr/src/linux/include/asm/mach-
default -fomit-frame-pointer -o driver/modules/hostap_cs.o driver/mod-
ules/hostap_cs.c
gcc -I/usr/src/linux/include  -O2 -D_ _KERNEL_ _ -DMODULE -Wall -g -c  -
I/usr/src/linux/arch/i386/mach-generic -I/usr/src/linux/include/asm/mach-
default -fomit-frame-pointer -o driver/modules/hostap.o driver/mod-
ules/hostap.c
.
.
Run 'make install_pccard' as root to install hostap_cs.o

[root@localhost root]# make pccard_install
Installing hostap_crypt*.o to /lib/modules/2.4.20-8/net
```

```
mkdir -p /lib/modules/2.4.20-8/net
.
.
Installing /etc/pcmcia/hostap_cs.conf
[root@localhost hostap-0.0.1]#
```

A fordítás után vegyük észre az */etc/pcmcia* könyvtárba telepített *hostap_cs.conf* állományt. Ez a modul konfigurációs állománya, amely utasítja a modult, hogy töltődjön be, ha megfelelő kártyát érzékel. A listában számos népszerű kártyát beállíthatunk, ha azonban a sajátunkat nem találjuk közöttük, nekünk kell azt hozzáadnunk. A feladat egyszerű, a bejegyzések legtöbbször mindössze háromsorosak:

```
card "Compaq WL100 11Mb/s WLAN Card"
   manfid 0x0138, 0x0002
   bind "hostap_cs"
```

A kártyánk gyártójának pontos adatait a következő paranccsal határozhatjuk meg:

```
[root@localhost etc]# cardctl ident
Socket 0:
  no product info available
Socket 1:
  product info: "Lucent Technologies", "WaveLAN/IEEE", "Version 01.01", ""
  manfid: 0x0156, 0x0002
  function: 6 (network)
[root@localhost etc]#
```

Ezt követően az eszköz telepítéséhez használhatjuk a *modprobe* parancsot, vagy újraindíthatjuk a rendszert, hogy az eszköz automatikusan betöltődjön. A rendszernaplóban ellenőrizhetjük a megfelelő betöltődést; a következőhöz hasonló üzenetet kell találnunk:

```
hostap_crypt: registered algorithm 'NULL'
hostap_cs: hostap_cs.c 0.0.1 2002-10-12
  (SSH Communications Security Corp, Jouni Malinen)
hostap_cs: (c) Jouni Malinen
PCI: Found IRQ 12 for device 00:0b.0
hostap_cs: Registered netdevice wlan0
prism2_hw_init( )
prism2_hw_config: initialized in 17775 iterations
wlan0: NIC: id=0x8013 v1.0.0
wlan0: PRI: id=0x15 v1.0.7
wlan0: STA: id=0x1f v1.3.5
wlan0: defaulting to host-based encryption as a workaround for
  firmware bug in Host AP mode WEP
wlan0: LinkStatus=2 (Disconnected)
wlan0: Intersil Prism2.5 PCI: mem=0xe7000000, irq=12
wlan0: prism2_open
wlan0: LinkStatus=2 (Disconnected)
```

A HostAP beállítása

Ahogy már említettük, a HostAP beállításához szükségünk van az *iwconfig* programra. Mindenekelőtt megmondjuk a HostAP-nek, hogy infrastrukturális üzemmódban szeretnénk használni. Ezt a következő utasítással tehetjük meg:

```
vlager# iwconfig wlan0 mode Master
```

Ezután beállítjuk az ESSID azonosítót. Ez lesz a hozzáférési pont neve, amelyet valamennyi ügyfél látni fog. A példában a hozzáférési pontunk neve „pub":

```
vlager#  iwconfig wlan0 essid pub
```

Most állítsuk be az IP címet:

```
vlager# iwconfig wlan0 10.10.0.1
```

A csatorna kiválasztása fontos lépés, és mint korábban említettük, ehhez érdemes először felmérni a helyszínt, hogy a legkevésbé „zsúfolt" csatornát választhassuk:

```
vlager# iwconfig channel 1
```

Miután ezzel elkészültünk, ellenőrizzük, hogy mindent jól adtunk meg. A következő parancs kimenetén ezt látjuk:

```
vlager# iwconfig wlan0
wlan0     IEEE 802.11-DS  ESSID:"pub"  Nickname:" "
          Mode:Managed  Frequency:2.457GHz  Access Point:00:04:5A:0F:19:3D
          Bit Rate=11Mb/s   Tx-Power=15 dBm   Sensitivity:1/3
          Retry limit:4   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality:21/92  Signal level:-74 dBm  Noise level:-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:2960
          Tx excessive retries:1  Invalid misc:0   Missed beacon:0
```

A HostAP konfigurálását befejeztük, most már be tudjuk állítani az ügyfeleket, hogy a vezeték nélküli hálózaton keresztül kapcsolódjanak a Linux kiszolgálóra.

További beállítási lehetőségek

Ahogy egyre jobban megismerjük a HostAP-t, további hasznos beállítási lehetőségekkel is találkozunk, például a MAC szűréssel és a WEP konfigurációval. Érdemes e biztonsági intézkedések közül egyet, esetleg mindkettőt alkalmaznunk, mert az alapértelmezett konfiguráció nyílt hozzáférési pontot hoz létre, amelyet bárki kifürkészhet és igénybe vehet, ha a hatótávolságán belül tartózkodik. Noha nem teszi lehetetlenné, a WEP megnehezíti a fürkészést, és az illetéktelen felhasználók is csak bonyolult módszerekkel kapcsolódhatnak hálózatunkra. A MAC címek szűrése szintén jó módszer a hívatlan vendégek távoltartására. Szeretnénk ismét hangsúlyozni, hogy a 802.11b protokoll hiányossá-

gai miatt egyik eljárás sem garantál biztonságos számítógépes környezetet. Egy vezeték nélküli telepítést a hagyományos biztonsági eljárásokkal, például VPN-ekkel tehetünk biztonságossá. A VPN segítségével gondoskodhatunk a titoktartásról és a hitelesítésről, hiszen egy vezeték nélküli LAN ügyfelére ugyanúgy kell tekintenünk, mint bármely internetes ügyfélre – megbízhatatlanként.

A WEP bekapcsolása egyszerű és az *iwconfig* paranccsal végezhető. Megválaszthatjuk, hogy 40 vagy 104 bites kulcsot szeretnénk használni. A 40 bites WEP kulcsot 10 hexadecimális számjeggyel állíthatjuk be:

```
# iwconfig wlan0 key 1234567890
```

A 104 bites WEP kulcsot 26 hexadecimális számjeggyel állíthatjuk be, amelyeket négyes csoportokban, kötőjellel elválasztva adunk meg:

```
# iwconfig wlan0 key 1000-2000-3000-4000-5000-6000-70
```

A következő utasítással ellenőrizhetjük a kulcsot:

```
# iwconfig wlan0
```

Fontos tudnunk azt is, hogy a WEP kulcsot ASCII karakterlánccal is beállíthatjuk. Számos oka van annak, hogy ez a megoldás miért nem igazán jó, a legfontosabb, hogy az ASCII kulcsok nem mindig működnek, amikor a megadásuk az ügyfél oldalán történik. Ha mégis úgy döntünk, hogy szeretnénk kipróbálni, az ASCII kulcs beállításához adjuk meg az *iwconfig* parancsban az *s:* argumentumot, majd a kulcsot:

A 40 bites kulcshoz 5 karakter szükséges, ez felel meg a 10 számjegyes hexadecimális kifejezésnek:

```
# iwconfig wlan0 key s:smile
```

A 104 bites kulcshoz 13 karakter szükséges, ez felel meg a korábbi példa 26 számjegyes hexadecimális kifejezésének:

```
# iwconfig wlan0 key s:passwordtest3
```

A WEP kikapcsolására a következő utasítás szolgál:

```
# iwconfig wlan0 key off
```

A HostAP egy további hasznos szolgáltatása az ügyfelek MAC címek alapján történő szűrése. Az eljárás nem „bombabiztos", de bizonyos mértékig véd az illetéktelen felhasználóktól.

A MAC címek szűrésére két alapvető eljárás létezik: vagy engedélyezzük, vagy tiltjuk a listában szereplő felhasználókat. Mindkét opciót az *iwpriv* parancsban állíthatjuk be. A következő utasítás bekapcsolja a MAC szűrést, engedélyezve a MAC listában szereplő ügyfeleket:

```
# iwpriv wlan0 maccmd 1
```

A maccmd MAC szűrő parancs opciói:

`maccmd 0`
A MAC szűrés kikapcsolása

`maccmd 1`
A MAC szűrés bekapcsolása, a megadott MAC címek engedélyezése

`maccmd 2`
A MAC szűrés bekapcsolása, a megadott MAC címek tiltása

`maccmd 3`
A MAC szűrő táblázatának törlése

A MAC címeket a listánkhoz az *iwpriv* utasítással adhatjuk hozzá ismét:

```
# iwpriv wlan0 addmac 00:44:00:44:00:44
```

A parancs felveszi a listába a 00:44:00:44:00:44 MAC című ügyfelet. Az ügyfél rákapcsolódhat vezeték nélküli hálózatunkra. Ha azonban egyszer úgy döntünk, hogy az ügyfelet többé nem kívánjuk engedélyezni, a következő utasítással eltávolíthatjuk:

```
# iwpriv wlan0 delmac 00:44:00:44:00:44
```

A MAC címet eltávolítottuk, a továbbiakban már nem képes kapcsolódni. Amennyiben sok ügyfél MAC címével rendelkezünk és mindegyiket el szeretnénk távolítani, a teljes MAC hozzáférés-szabályozási listát a következő utasítással törölhetjük:

```
# iwpriv wlan0 maccmd 3
```

A hozzáférés-szabályozási lista törlése után vagy kikapcsoljuk a szűrést, vagy olyan érvényes MAC címeket adunk meg, amelyeket engedélyezni szeretnénk.

Hibakeresés

Mivel nem használnak vezetékeket, a 802.11b hálózatokban nagyobb a meghibásodás esélye, mint a hagyományos vezetékes hálózatokban. A vezeték nélküli telepítés tervezésénél számos kérdésre kell választ adnunk.

Az első a jelerősség. Természetesen elég erős jelre van szükségünk ahhoz, hogy minden ügyfelünket elérjük, mégsem szeretnénk azonban a vakvilágba sugározni. A jelerősséget az antenna, a hozzáférési pont elhelyezése és egyes szoftveres beállítások befolyásolják. A környezetünkben a legmegfelelőbb kiépítést és elhelyezést kísérletezéssel határozhatjuk meg.

Az interferencia szintén fontos tényező lehet. Sok más eszköz is a 802.11b által használt 2,4 GHz-es frekvencián osztozik. A vezeték nélküli telefonok, a gyerekfigyelő rendszerek és mikrohullámú sütők egyaránt okozhatnak bizonyos mértékű interferenciát hálózatunkban. A választott csatornánkkal azonos vagy ahhoz közeli csatornán működő szomszédos hozzáférési pontok szintén zavarhatják hálózatunkat. Nem valószínű, hogy kimondottan ez a probléma leállást okozna, egészen biztosan rontja azonban a teljesítményt. Ismét csak azt ajánlhatjuk, hogy egy nagyobb rendszer kiépítése előtt végezzünk kísérleteket.

A fizikai tényezők mellett sok szoftver vonatkozású kérdéssel is szembe kell néznünk. A legtöbb gondot az illesztőprogramok, illetve a kártyák összeférhetetlensége okozza. Az ilyen jellegű hibákat elkerülhetjük, ha pontosan ismerjük a használt hardvert. Ha meg tudjuk határozni a lapkakészletet, a megfelelő illesztőprogramot is könnyebben megtalálhatjuk.

A kártya azonosításához kiadhatjuk a *cardctl* utasítást, illetve vethetünk egy pillantást a rendszernaplóra:

```
[root@localhost etc]# cardctl ident
Socket 0:
  no product info available
Socket 1:
  product info: "Lucent Technologies", "WaveLAN/IEEE", "Version 01.01", ""
  manfid: 0x0156, 0x0002
  function: 6 (network)
[root@localhost etc]#
```

A példában egy könnyen azonosítható WaveLAN kártyát láthatunk. Természetesen az utasítás csak a sikeres beállítás és a modul megfelelő betöltése után működik.

A hálózatok összekötése híddal

A HostAP szoftver és az illesztőprogramok helyes beállítása után hasznos lehet, ha az ügyfeleknek hozzáférést biztosítunk vezetékes LAN hálózatunkhoz. Ezt egy híddal (bridge) valósíthatjuk meg. A szükséges szoftver letölthető a *http://bridge.sourceforge.net* oldalról. Egyes terjesztések tartalmazzák a telepítésre alkalmas csomagokat valamennyi szükséges eszközzel és kernelmódosítással. A Red Hat RPM-ek mindkettőt tartalmazzák. A Debian felhasználóknak elegendő az *apt-get install bridge-utils* utasítást kiadniuk. A részleteket megtaláljuk terjesztésünk dokumentációjában.

A hídkezelő szoftver szabályozására a *brctl* program szolgál. Ez a szoftver fő konfigurációs eszköze, és számos beállítási lehetősége van. Mi az elérhető opciók közül csupán egypárat használunk, átfogóbb listát a szoftverrel együtt telepített brctl súgóoldalakon találunk.

A híd kialakításának első lépése egy virtuális híd interfész létrehozása a következő utasítással:

```
vlager#  brctl addbr br0
```

A parancs létrehoz a számítógépen egy interfészt, amelyet a két kapcsolat összekötésére használunk. Az interfészt az *ifconfig* futtatásával ellenőrizhetjük:

```
vlager# ifconfig br0
br0       Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

Emellett a rendszernaplóban is megnézhetjük, sikerült-e üzembe helyeznünk a hidat:

```
Jan 22 13:17:54 vlager kernel: NET4: Ethernet Bridge 008 for NET4.0
```

Ezután a híddal összekötni kívánt két interfészt adjuk meg. A példában a `wlan0` vezeték nélküli interfészt és az `eth0` vezetékes Ethernet interfészt szeretnénk híddal összekapcsolni. Fontos azonban, hogy először eltávolítsuk az interfészek IP címeit. Mivel a hálózatokat a kettes rétegen kapcsoljuk össze, nincs szükségünk az IP címekre.

```
Vlager#  ifconfig eth1 0.0.0.0 down
Vlager#  ifconfig wlan0 0.0.0.0 down
```

Az IP címek eltávolítása után a hídhoz adhatjuk az interfészeket.

```
vlager# brctl addif br0 wlan0
vlager# brctl addif br0 eth1
```

Az `addif` opcióval a híddal összekötni kívánt interfészeket adhatjuk meg. Ha más vezetékes vagy vezeték nélküli interfészünk is van, és ezeket a hídhoz szeretnénk adni, most megtehetjük.

A híd létrehozásának utolsó lépése az interfészek elindítása. Most dönthetünk arról is, hogy a híd interfészhez kívánunk-e IP címet rendelni. Az IP cím megadásával távolról is kezelhetjük a kiszolgálót. Bár egyesek szerint az eszköz biztonságosabb IP cím nélkül, a legtöbb esetben hasznos, ha a távoli kezelés lehetőségét megteremtjük. A híd üzembe helyezéséhez bekapcsoljuk a híd interfészt, illetve a két hardver interfészt.

```
vlager#  ifconfig br0 10.10.0.1 up
vlager#  ifconfig wlan0 up
vlager#  ifconfig eth0 up
```

A parancsokat sikeresen lefuttatva most már elérhetjük a hidat a 10.10.0.1 IP címen. Természetesen a címnek a híd mindkét oldaláról elérhetőnek kell lennie.

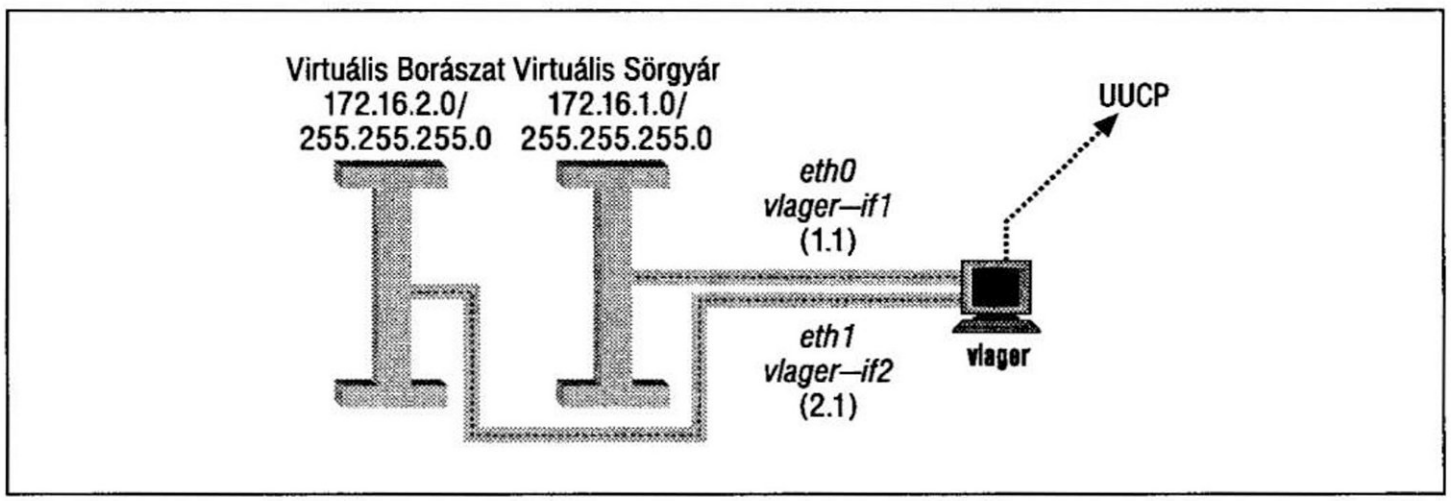
Mindezt akár automatikusan, a rendszer indításakor is betölthetjük. A feladatot szinte minden Linux terjesztésen eltérő módon végezzük. A terjesztésünk dokumentációjában megtaláljuk, hogyan szükséges módosítanunk az indítószkripteket.

A

Egy mintahálózat: a Virtuális Sörgyár

A könyvben többször is találkozhatunk a következő példával, amely a Groucho Marx Egyetem példájánál egyszerűbb, és valószínűleg közelebb áll azokhoz a feladatokhoz, amelyekkel a valóságban találkozni fogunk.

A Virtuális Sörgyár egy kis cég, amely - ahogy neve is sugallja - virtuális sörök főzésével foglalkozik. Azért, hogy hatékonyabbá tegyék a menedzsmentet, a virtuális sörfőzők hálózatba szeretnék kötni számítógépeiket, amelyek történetesen egytől egyig a legokosabb és legnagyszerűbb Linux kernelt futtató személyi számítógépek. Az A1. ábra a hálózat kiépítését szemlélteti.

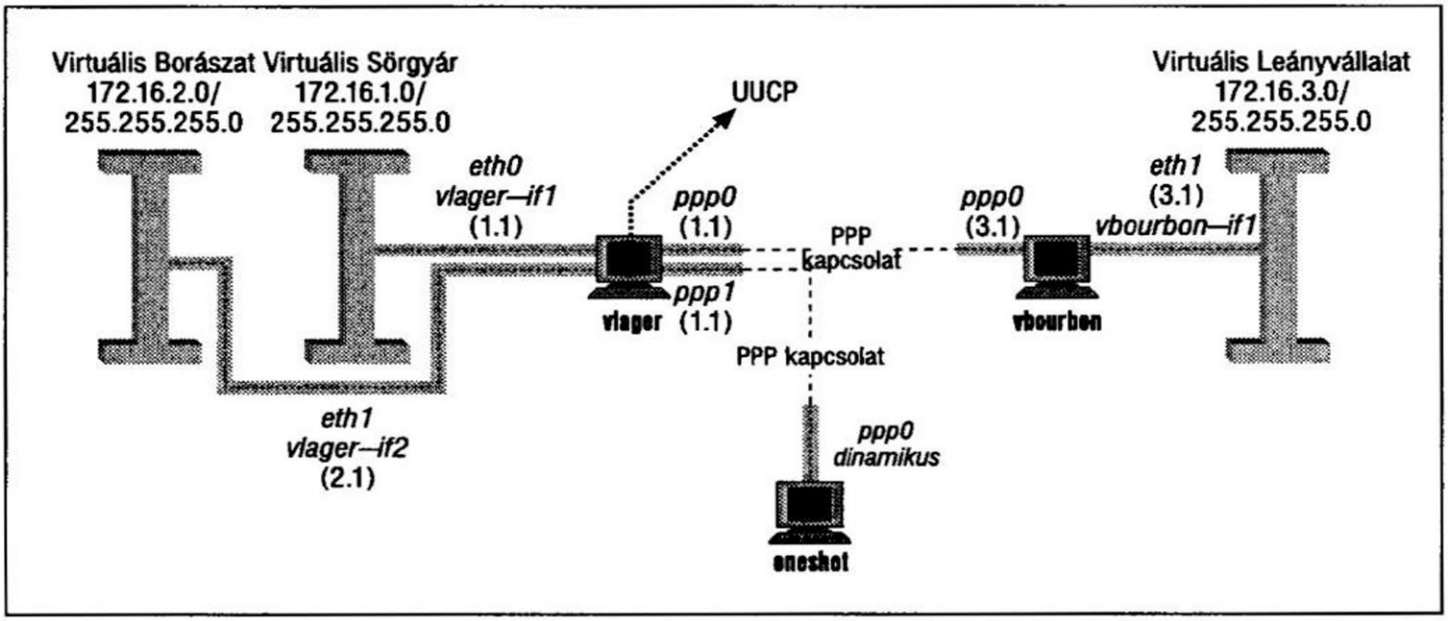


A1. ábra. A Virtuális Sörgyár és a Virtuális Borászat alhálózatok

Ugyanazon a szinten, a nagyteremmel szemben helyezkedik el a Virtuális Sörgyárral szorosan együttműködő Virtuális Borászat. A borászok saját Ethernet hálózatot tartanak fenn. Érthető módon a két cég szeretné a hálózatait összekötni, mihelyt azok működőképesek. Első lépésben egy átjáró gazdagépet szeretnének felállítani, amely a datagramokat továbbítja a két alhálózat között. Később szeretnének kialakítani egy UUCP kapcsolatot is a külvilággal, ezen keresztül bonyolódna a levelezés és a hírek beszerzése. Hosszú távú terveik között szerepel PPP kapcsolatok létrehozása is az üzemen kívüli helyszínekhez és az internethez.

A Virtuális Sörgyár és a Virtuális Borászat a Sörgyár B osztályú hálózatának egy-egy C osztályú alhálózatával rendelkezik. Az alhálózatok közötti kapcsolatot az átjáróként

szolgáló *vlager* gazdagép biztosítja, amely az UUCP kapcsolatot is támogatja. Az A2. ábrán ezt a felépítést láthatjuk.



A2. ábra. A Virtuális Sörgyár hálózata

A Virtuális Leányvállalat hálózatának kapcsolódása

A Virtuális Sörgyár napról napra bővül, és egy új leányvállalatot nyit egy másik városban. A leányvállalat saját Ethernet hálózatot tart fenn, ahol az IP hálózati szám a **172.16.3.0**, amely a sörgyár B osztályú hálózatának 3-as alhálózata. A **vlager** a sörgyár hálózatának átjárójaként lép fel, és támogatja a PPP kapcsolatot; a leányvállalatnál a társgép neve **vbourbon**, IP címe pedig a **172.16.3.1**. A hálózatot az A2. ábrán láthatjuk.

Tárgymutató

B

D

H

J

M

O, Ö

Q

R

T

U, Ü

A könyv szerzőiről

Tony Bautts az elmúlt években független biztonsági tanácsadóként a Fortune 500 vállalatcsoportnak dolgozott az Egyesült Államokban és Japánban. Előadásokat tartott a The Information Systems Audit and Control Association (ISACA) biztonsági kérdésekkel foglalkozó rendezvényein, valamint felszólalt, illetve elnökölt a MIS Training Institute tanácskozásain. Társszerzőként közreműködött a *Hack Proofing Your Wireless Network,* a *Nokia Network Solutions Handbook* és a *Security Certification Handbook* című kötetek, továbbá műszaki lektorként az *Implementing IPv6 on Cisco IOS* című kötet megalkotásában. Szakterületéhez tartozik a vezeték nélküli hálózatkezelés, a biztonsági infrastruktúrák tervezése és a behatolást követő hibafeltárás és elhárítás.

Terry Dawson a távközlésben végzett két évtizedes munkájával és ott szerzett tapasztalatával jelenleg a Telstra Research Laboratories katonai támogató rendszereket kutató intézet egyik csoportjának vezetője.

Gregor N. Purdy az Amazon.com gazdasági szolgáltatásokkal foglalkozó kiterjedt csoportjának egyik mérnök vezetője. Mielőtt 2003-ban az Amazon.com-hoz csatlakozott volna, Gregor tíz évet töltött el tanácsadóként a fejlett adattárolás, a rendszerintegráció és a szoftverek, valamint az internetes szabványok előzetes kutatásával. Részt vett számos nyílt forráskódú fejlesztésben, többek között a Perl mag és bővítő moduljai, a Perl shell és a Perl 6-hoz készült Parrot virtuális gép elkészítésében.