1.344 Ft

II. Évfolyam 11. szám

ISSN 15865185

tech.net

A Microsoft Magyarország szakmai magazinja

Farkasokkal táncoló

4. oldal

WOLFPACK

A **„tech.net magazin Brainstorm"** a Dupla KV rovathoz hasonló, ám a személyes kérdésfelvetést és vitát is lehetővé tevő rendezvény, melynek célja:

- **az elsőre talán ismeretlen technológiák élő bemutatása**
- **a cikkekhez kapcsolódó kódok megírása/kipróbálása**
- **a terjedelmi okokból kimaradt információk átadása**

E magazinnal együtt a rendezvényre érvényes belépőjegyet minden előfizetőnkhöz eljuttattuk.
További információk a belépőjegyen olvashatók.

Várjuk Önöket a NetAcademia Mesterkurzusokon

NETACADEMIA
A legjobbakat tanítjuk.

(Bár belépőjegyet adtunk, ez a tény önmagában nem biztosítja helyét a rendezvényen. A jegy célja előfizetőink elsődlegességének biztosítása, de a korlátozott résztvevői létszám (100 fő) miatt a regisztráció kötelező. Jelentkezzen, amíg nem késő!)

http://technet.netacademia.net/brainstorm

HunNet RL512W

512 kbit/sec-os szimmetrikus forgalomfüggetlen csatlakozás az internetre havi 35.000 Ft-ért

.hu Hungarian Networks

Tel: 06-40-hunnet (486-638)

2001 November

tech.net
A Microsoft Magyarország Szakmai Magazinja

Szerkesztőség
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
Szerkesztőség címe:
1105 Budapest, Ihász utca 13.
Tel.: 263-2732
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti
a **NetAcademia Kft.**
Terjesztési, előfizetési információ:
Tel.: 263-2732
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 3.500

Előfizethető megrendelőlevélben a szerkesztőségnél:
1105 Budapest, Ihász utca 13.
Fax: 261-7145
http://technet.netacademia.net/subs

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Balogh Zoltán**
Tel.: 489-4665
Fax: 489-4660
info@velvethammer.hu
1027 Budapest, Fő utca 67. V. 1.
Grafikai tervezés, kivitelezés,
nyomdai előkészítés:
Bársonykalapács Marketing
Művészeti vezető: **Balogh Zoltán**

Nyomda:
Cerberus Kft.
1066 Budapest, Lovag u. 14.
Felelős vezető: **Schmidt Gábor**

ISSN 1586-5185

Mostan színes kockákról álmodom...

Ha valaki elolvassa az MCP vagy MCSE címek viselésének feltételeit, találkozhat egy olyan kikötéssel, mely szerint nem szabad rossz hírét kelteni a Microsoft szoftvereinek, szakmai fórumokon pedig azok előnyeit kell ismertetni. Jártamban-keltemben elég sok kollégával találkozom, akik - nyíltan vagy burkoltan - becsmérlik a Microsoft rendszereit, különösen a rendelkezésre állásukat tartják nagyon rossznak. Mivel a névjegykártyámon nem virít semmilyen cím, szívesen elbeszélgetek velük inkognitóban, s igyekszem tárgyi tévedéseiket helyreigazítani. Egy-egy diskurzus során kiderül, hogy gyakran még a legelemibb intézkedéseket sem teszik meg annak érdekében, hogy a rendszerük csak egy kicsit is tovább üzemeljen, a finomságokról szó sem esik. Valahogy szélmalomharcnak tűnik az egész.

Elhatároztam tehát, hogy összegyűjtöm, és közkinccsé teszem azokat az információkat, amelyek segítségével hibatűrő Microsoft rendszereket lehet létrehozni. Nem is biztos, hogy az operációs rendszernél kell ezt kezdeni. Ma egy átlagos rendszergazda talán tisztában van egy PC felépítésével, de már egy belépő szintű szerver is titkokat rejt a számára, a négy- és nyolcprocesszoros Góliátokhoz „ingyen hozzácsapott" csodákról már ne is beszéljünk. S ha valaki nincs tisztában a lehetőségeivel, akkor egy beruházási döntés előtt nem lesz képes „súgni" a döntéshozónak, hogy mit is kellene belecsempészni még abba a gépbe, hogy esetleg ne kelljen éjjel egyig küszködni majd, ha beüt a krach. Mert persze a felhasználó ideje szent, ami rendjén is van. Olyanok vagyunk mi, mint Münchausen báró: vagy kirántjuk saját magunkat az összeomlások mocsarából, vagy csak magunkra vethetünk, ha nem tesszük.

Az, hogy korábban nem volt minden rendjén, nemcsak a felhasználók látták, hanem a Microsoft is. Ha belegondolunk, akkor a Windows NT megszületésének egyik mozgatórugója is a nagyobb megbízhatóság, végső soron a nagyobb rendelkezésre állás volt. A Windows 2000 azonban minden elődjén túltett, hogy stabil és megbízható eszköz legyen. Többfrontos támadás volt ez a hibák ellen, igen sok fegyvert felvonultatva. A leghatásosabbnak tűnő ezek közül a fürtszolgáltatás, amely a Windows NT 4.0 Enterprise Editionben debütált, s most a Windows 2000 izmosabb változataiban nyílt ki igazán. Nem tudok róla, hogy lenne még egy ilyen hardverfüggetlen, operációs rendszerbe épített és olcsó megoldás a magasabb rendelkezésre állás biztosítására.

Sok mindenre mondják, hogy „ez a jövő". Én merem állítani, hogy a nagyobb rendelkezésre állás tényleg a jövő, és legalább két okom van ezt feltételezni. Egyrészt a kapitalizmus nálunk is „begyűrűzik", a tulajdonosok facsarják a menedzsmentet, ők pedig tűzzel-vassal keresik majd a profitot és üldözik a profit kiesését. Ha hozzávesszük ehhez, hogy nagyon közel van az idő *(ha el nem jött máris)* amikor a számítógépre bízzuk legalábbis mindennapi jövedelmünket, akkor nem kétséges, hogy nem áll meg a talpán az a rendszergazda, aki hanyagsága, nemtudása vagy hozzá nem értése miatt „kiejti" a cégét az üzletből pusztán azzal, hogy nem gondoskodott időben a rendszere megbízhatóságáról.

A másik ok a leendő IT vezetők becsülete. A jó vezető tudja, hogy bár ő nem ért a bitekhez és a bájtokhoz, egy leállás mégis az ő lelkén szárad, mert – ahogy a közmondás tartja – a fejétől bűzlik a hal. Ez pedig becsületbeli ügy. Egy IT vezető a messiás, aki hirdeti az új világ igéjét: használjátok az informatikát, mert az jövedelmez. De vajon mit ér annak a szava, akinek a munkatársai nem képesek életben tartani az új világnak a rájuk bízott szeletét?

Eléggé nyomós, a fizetésünket és munkahelyünket érintő érvek ezek számunka, a technikai szakemberek számára, hogy komolyan vegyük a dolgunkat, és hibatűrő, lehetőleg agyoncsaphatatlan szolgáltatásokat hozzunk létre. Komoly munka komoly felkészülést kíván. Azt ajánlom a kedves Olvasónak, hogy kövesse folyamatosan a fürtszolgáltatásról szóló cikksorozatot. Ha van már olyan eszközük, amely képes e komponens futtatására, akkor sok hasznos, a mindennapi üzemeltetéssel kapcsolatos tudnivalót találhatnak majd bennük. Ha pedig még nincs ilyen rendszerük, akkor ötlettárként lehet használni a cikkeket, hogy a következő beruházáskor lendíteni lehessen azon a becsületen.

Júliusban a barcelonai TechEd konferencián beszélgetés közben azt mondta az egyik kollégám, hogy egy cluster a „négykilences" rendelkezésre állás biztosítására való. 99.99% üzemidőre pedig még kevés alkalmazásnak van szüksége, ezért a clusternek még nem jött el az ideje. Erre azt válaszoltam, hogy nálunk a szervereinknek jobb rendelkezésre állása van, mint a WAN vonalainknak, mégsem elégedett a főnököm. S közben azon gondolkodtam, hogy egészen más dolog beszélni akár öt kilencesről, mint megcsinálni azt.

Hadd osszak meg egy titkot a kedves Olvasóval. Még egy cluster birtokában sem fog tudni senki egyik pillanatról a másikra 99.99%-os rendelkezésre állást produkálni, mert ennek a tudása nem *(csak)* a szoftverben rejlik. Ilyen precizitáshoz a teljes üzemeltetés módszertanának és szemléletének kell átalakulnia. Nem lehetetlen, csak nem megy rögtön. A Microsoftnak van egy módszertana, amelyet Microsoft Operation Frameworknek neveznek és épp azt a célt szolgálja, hogy kihozza azt a lehetőséget a szoftverrendszerekből, amely egyébként bennük szunnyad. Ennek elsajátítása és alkalmazása azonban olyan rossz beidegződések elhagyását igényli, amelyeket nem lehet csettintésre végrehajtani.

A cikksorozat csak abban segít, hogy az első három-négy kilencest elérjük – micsoda változás már ez is. Ezután könnyebb lesz majd a továbblépés.

Kosztolányi Dezső versének címét kissé elferdítve én most azokról a színes kockákról álmodom, amelyek a nagy üzembiztonságot reklámozták. Jó volna, ha néhányuk stabil építőkövé válhatna a mi rendszereinkben.

Lepenye Tamás (MCSE)
lepenyet@mal.hu

Farkasokkal táncoló *(I. rész)*

Wolfpack annyit tesz: farkasfalka. Wolfpack volt a kódneve annak a szoftvernek, amelyet a Microsoft 1997-ben adott ki, és a világ clusterszerverként ismert meg. Az a szándék, hogy minél nagyobb rendelkezésre állást lehessen biztosítani a Windows NT alapú rendszereknek, ebben a szolgáltatásban kristályosodott ki leginkább. Magyarországon még a Windows megbízhatatlanságának van híre, pedig már sok éve másról szól a fáma. Ezzel a cikksorozattal a Windows 2000 Advanced Server cluster *(magyarul kiszolgálófürt)* szolgáltatását szeretném egy kicsit alaposabban bemutatni. Mivel gyakorló rendszermérnök vagyok, szeretném feltárni az olvasó előtt azokat a motivációkat, amelyek a megvásárlás mellett szólhatnak, a műszaki környezetet, amelyet a szoftver igényel, és mindenekelőtt a tapasztalatot, amelyet az elmúlt hónapokban megszereztem. Találónak érzem az egykori kódnevet: a clusterek nagy, erős és összetett rendszerek, amelyekhez fontos az alapos felkészültség. De aki bátor, az akár a farkasokkal is táncol.

Mi a szerverfürt?

A szerverfürt egy vagy több olyan kiszolgáló, amelyek megfelelő hardvereszközök megléte esetén együttműködnek, hogy a rendszergazdák által definiált szolgáltatásokat a lehető legkisebb kieséssel működtessék. Első pillantásra furcsának tűnhet az „egy" a meghatározásban, de majd látható, hogy ennek is van értelme. Hogy a továbbiakban a fogalmi rendszer egyértelmű legyen, meg kell határoznunk néhány, a későbbiekben gyakran előforduló kifejezést.

Az állomás *(node)*: Ha a cluster azt az együttest jelenti, amely a szolgáltatásokat folyamatosan biztosítja, akkor az állomás a cluster egyik gépe. A Windows NT Server 4.0 Enterprise Edition és a Windows 2000 Advanced Server cluster maximálisan két állomásból állhat. A Windows 2000 Datacenter Server segítségével 4 node-ból álló cluster is építhető.

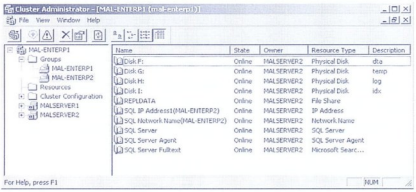
A fürtszolgáltatás *(cluster service)*: Egy szabványos NT szerviz, amely mindkét állomáson fut, feladata pedig a rábízott erőforrások biztosítása az ügyfelek felé. A szolgáltatás egy megfelelő jogosultságú tartományi fiók kontextusában működik.

Az erőforrás *(resource)*: Egy olyan hardver, szoftver vagy logikai komponens, amely működéséért a clusterszolgáltatás felel. Erőforrás lehet egy lemez, egy NetBIOS gépnév, egy szolgáltatás *(service)* vagy akár egy IP cím is.

Quorum disk: Az erőforrások között van egy különleges, a quorum disk. Ez egy olyan lemez, amely a két állomás közös „tudását" tartalmazza *(egy adatbázis formájában)* a cluster konfigurációjáról és állapotáról.

Függőség *(dependency)*: Az erőforrások nem elszigetelt egységek, hanem egymással kapcsolatban állnak, függnek egymástól. Például egy adatbáziskezelőnek szüksége van az adatbázis fájljaira és tranzakciós állományaira ahhoz, hogy elindulhasson. Ezek azonban egy lemezen találhatók. Az adatbáziskezelő szolgáltatás erőforrás tehát függ egy lemezerőforrástól.

A csoport *(group)*: A logikailag együtt kezelt erőforrások csoportot alkotnak. A fürtszolgáltatás általában erőforráscsoportokat és nem egyedi erőforrásokat kezel. A csoportokat a rendszergazdák alakítják ki úgy, hogy minden erőforrást tartalmazzanak, amely egy alkalmazás üzemeltetéséhez szükséges. Egy SQL Serverhez szükséges például egy gépnév, egy IP cím, egy lemez, és az MSSQLServer erőforrás.



SQL Server a fürtben

Átköltözés *(failover)*: Ha egy állomás nem képes működtetni egy adott erőforráscsoportot, akkor a fürtszolgáltatás átköltözteti az egészet a másik állomásra. Az átállás tényét a quorum disken található fürtadatbázisban rögzíti.

Visszaköltözés *(failback)*: Ha már mindkét állomás képes üzemeltetni az erőforráscsoportot, akkor a cluster a visszaköltöztetési szabályok alapján helyreállítja az eredeti feladatmegosztást. A szabályokról a későbbiekben szó lesz.

Virtuális kiszolgáló *(Virtual server)*: Az erőforráscsoportok beállíthatók úgy, hogy a felhasználók számára mint valóságos kiszolgálógépek jelenjenek meg. Ehhez az szükséges, hogy legyen név- és IP cím erőforrásuk. Azok a csoportok, amelyek rendelkeznek ilyen erőforrásokkal, virtuális kiszolgálók is egyben.

Szívhang (Heartbeat): Az állomások folyamatosan figyelik egymás állapotát, pontosabban azt, hogy képesek-e reagálni kérésekre. Ezt a forgalmat nevezzük szívhangnak.

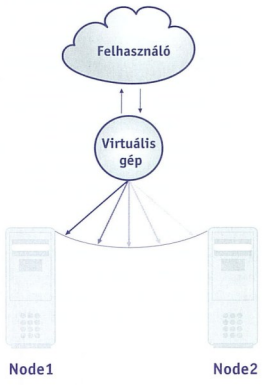
A cluster Microsoft-féle implementációja az ún. „shared-nothing", vagyis a „semmi közös" elven épült. Ez azt jelenti, hogy a cluster által definiált erőforrások egy adott pillanatban csak az egyik szerverhez tartoznak, a másik nem férhet hozzájuk. Hiba esetén fordul a kocka, és a másik állomás válik az erőforrások birtokosává. Közös erőforrások, amelyek fölött egyszerre gyakorolnának felügyeletet a node-ok, nincsenek, - vagyis semmi sem közös. Ez az elv azonban nem tévesztendő össze azzal a ténnyel, hogy léteznie kell olyan *(merevlemez)* eszköznek, amelyeket mindkét állomásnak el kell érnie. Ez az eszköz legtöbbször egy merevlemez-alrendszer, amely fizikailag közös. Tehát egy kiépített cluster rendelkezik egy fizikai szinten mindkét állomás számára elérhető lemezrendszerrel, amelyben az egyes lemezek vagy az egyik, vagy a másik node-hoz tartoznak egy időpillanatban. Még néhány általános tudnivaló. A clusterszolgáltatás leginkább kapcsolatorientált szoftverek rendelkezésre állásá-

nak növelésére való. Ilyen például az SQL Server vagy az Exchange. Ezt a szabályt azonban nem kell kőbe vésettnek tekinteni, mert a DHCP, WINS, sőt még az IIS is fürtözhető. A kapcsolatmentes technológiák *(Például IIS)* rendelkezésre állásának növelésére mégis inkább más módszerek használatosak, elsősorban a Network Load Balancing Server, mert NLBS-sel akár 32 állomást magában foglaló fürt is építhető. Vannak továbbá olyan szolgáltatások, amelyek állandó elérhetőségére az elosztott architektúrák a megfelelőek. Ilyen az Active Directory vagy a DNS.

A clusterszolgáltatás a Windows 2000 Advanced Server és a Windows 2000 Datacenter Server része. A Windows 2000 Server nem tartalmazza ezt a komponenst. Az Advanced Server ára jóval magasabb, mint az egyszerű Windows 2000 Serveré *(kb. háromszorosa)*, alaposan mérlegelni kell tehát a váltást. Windows NT 4.0 Serverről Windows 2000 Advanced Serverre frissíteni a szoftvert alig valamivel kevesebb, mint egy teljesen új licencet vásárolni, Windows NT 4.0 Enterprise Editionről viszont az átállás jóval barátságosabb. A magasabb árért cserébe az Advanced Server elsősorban a rendelkezésre állást növelő szolgáltatásokban nyújt többet, s a cikksorozat végére kiderül, hogy bőven „megéri az árát".

Hogyan működik a fürt

A szerverfürt közös tudása a quorum lemezen található, amelyet leginkább egy dedikált, vagyis csak erre a célra használt, merevlemezen elhelyezkedő tranzakciós adatbázisként lehet elképzelni. Ez a közös tudás definiálja a csoportokat és bennük az erőforrásokat, valamint azt, hogy a csoportokat mely node-ok birtokolhatják. Ha valamilyen oknál fogva az egyik gép nem képes az általa birtokolt csoportok számára a működést biztosítani, akkor a másik állomás átveszi a csoportok működtetését. A felhasználók, akik ténylegesen a virtuális kiszolgálók szolgáltatásait veszik igénybe, nem észlelik a hibát, vagy csak rövid kiesést tapasztalnak. Ha ábrázolni kell a szereplőket, akkor leginkább egy ingaórához hasonló szerkezetet rajzolhatunk.



A virtuális kiszolgálók ingázása

Jogos a kérdés, hogy mi történik akkor, amikor az inga épp a két állomás között jár. Akkor is üzemel a szolgáltatás? Nem. A cluster nem biztosítja a teljes, 100%-os rendelkezésre állást, nem tükrözi a hálózati kapcsolatokat, és az ügyféloldali szoftver tulajdonságai határozzák meg, hogy a felhasználó észreveszi-e a kimaradást. Ha például kiszolgálóoldalon egy Exchange szoftvert futtatunk egy fürtön, a ügyféloldalon pedig egy Outlook dolgozik, akkor az Outlook az „inga" átbillenése után ott folytatja, ahol korábban abbahagyta. Egy SQL Server kliens, főleg egy felette futó alkalmazás *(SAP, JDE stb.)* viszont igényelheti az ügyfelek újraindítását. Az inga átlendülése tehát kritikus tényező, szerencsére az állomások ezt nagyon gyorsan elvégzik. *(Egy J.D Edwards B7.3.3.1 és egy SQL Server 2000 150 egyidejű felhasználó esetén 10 másodperc alatt vándorol át egyik fürttagról a másikra. Ez saját tapasztalat, de ettől lehetnek eltérések, amelyet az erőforrások száma és természetesen a kiszolgálók izmossága is befolyásol.)*

Szóval 10 másodperc kiesés. Sok? Hasonlítsuk ezt össze azzal a leállással, amit egy nem cluster gép nyújt, amelyiknek meghibásodik a processzora. Mennyi ideig tart azt kicserélni, még ha van is alkatrész? Egy óra? És ha nincs alkatrész? Két nap? Két nap helyett 10 másodperc. Jó csere? De mondhatok más példát is. Ha tervszerűen karbantartást kell végezni, mikor és mennyi idő alatt történik ez meg? Egy éjszaka alatt? Egy hétvégén? Mennyi ennek a költsége? Nem egyszerűbb nappal, munkaidőben dolgozni? A cluster ezt lehetővé teszi. Az erőforrások éjszaka átcsoportosíthatók az egyik állomásra, reggel pedig lehet karbantartani *(ütni)* a másikat. Erre ideális a szerverfürt.

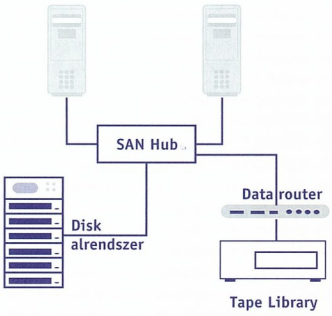
A cluster konfigurációja

Tekintsük át, mire érdemes ügyelni egy clusterkörnyezet tervezésekor. A Microsoft azt javasolja, hogy egy cluster kevés számú, jól definiált szolgáltatást nyújtson, továbbá minden funkciót a fürtszolgáltatáson keresztül érjenek el a felhasználók. Ezt én egyszerű és tiszta konfigurációnak nevezem. Egyszerű, mert kevés alkalmazást felügyel a cluster szerviz, és tiszta konfiguráció, mert nincs fürtön kívüli funkció. Az ajánlás mögött az az elgondolás húzódik meg, hogy ilyen rendszer üzemeltetése jóval kiszámíthatóbb és egyszerűbb, és várhatóan beváltja a hozzá fűzött magas rendelkezésre állás reményét. Egy apró szépséghibája van a fenti gondolatnak: nagyon drága. Ha állandóan szeretnénk biztosítani a felhasználók számára a nyomtatás lehetőségét, a levelezést, a vállalatirányítási rendszert, a felhasználói könyvtárakat, a névfeloldást stb. stb. és mindig egy-egy új fürtbe kötött szerverpárt állítanánk be, igencsak megcsapolnánk a vállalatunk beruházásra fordítható pénzeszközeit. Ez nem járható út. A hasznos azonban lehet viszonylag olcsó is, csupán annyit kell tenni, hogy a szolgáltatásokat egyetlen clusterre kell költöztetni. Ha egyetlen, jól megtervezett, erőforrásokkal ellátott kiszolgálópárra telepítünk mindent, akkor akár már a beruházási költségek is összevethetőek lesznek 5-6 különálló kiszolgáló megvásárlásával. A rendelkezésre állás és az üzemeltetési költségek pedig össze nem hasonlítható módon jobbak lesznek, mivel az egyszer megvásárolt redundancia minden szolgáltatásra kiterjed, míg különálló gépek a tartalékok hiánya miatt nem képesek megbízhatóan kiszolgálni a felhasználókat, legalábbis a fürthöz viszonyítva nem. Mindenképp ajánlatos tehát, hogy minél több szolgáltatást biztosítson egy fürt, mert a kiajánlott szolgáltatásokra jutó

fajlagos redundanciaköltség így alacsonyabb lesz. Igen, igen, ez a szerverkonszolidáció. Meggyőződésem, hogy nem is olyan sokára ez a nézet általánosan elterjedt lesz, és a Windows kiszolgálófürt nem csodabogár ritkaságként gubbaszt egy teremben, hanem általános és olcsó módja lesz a rendelkezésre állás növelésének.

Nem tartozik a cikk keretei közé egy kiszolgáló méretezésének elemzése, ezért processzorszámra és memóriaméretre nem tudok javaslatot tenni. Néhány szót azonban érdemes szentelni a közös lemezek kialakítására. A legegyszerűbb megoldás a közös SCSI busz alkalmazása. Kétségtelenül olcsóbb megoldás, de nem számol a jövővel. A clustert nem egy évig használjuk majd és nem is kettőig. Olyan módszert érdemes alkalmazni, amely a technológiai életciklusa elején jár, így van esélye annak, hogy pár év múlva nem fogunk bővíthetőségi problémákkal küzdeni. Ma a tárolórendszerek terén a jövőbe mutató szabványok a Fibre Channel technológia és a Storage Area Network. Nemcsak nagyobb sávszélességet nyújtanak, de rugalmasabb konfigurációt is lehetővé tesznek. Ha tehát ez lehetséges, javaslom a SAN-megoldások alkalmazását. A SAN szintén egyfajta konszolidáció, csak épp az adattárolás területén. Ez egy olyan adattároló hálózat, amelyben kiszolgálók, merevlemeztömbök és szalagos könyvtárak kapcsolódnak egymáshoz SAN hálózati eszközökön keresztül, amelyeket *(hogy az életet ne bonyolítsuk)* éppúgy huboknak, switcheknek és routereknek hívunk, mint a hagyományos hálózati eszközöket és szerepük a nevükhöz hasonlatos. A SAN belső titkait most nem részletezve annyit érdemes elmondani, hogy SAN hubot akkor érdemes használni, ha az *(adat)* hálózaton kommunikálók száma viszonylag kevés *(pl. egy cluster és egy-két merevlemez alrendszer)*, a switch viszont alkalmasabb a két vagy több clustert magában foglaló rendszerekhez. Mellesleg a SAN rendszer egyik előnye, hogy elegendő egyetlen mentést végző eszközt *(pl. tape library-t, szalagos könyvtárat)* vásárolni, az képes lesz a teljes adattároló hálózat lementésére úgy, hogy a tényleges (pl. Ethernet) hálózatot nem terheli. A mentési eszköz ugyanúgy közös, mint a lemezalrendszerek, tehát bármely gép használhatja.

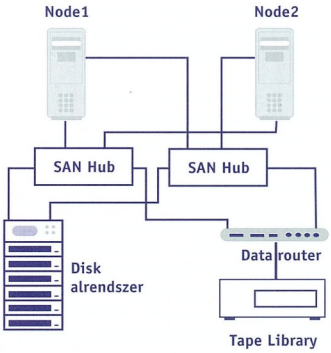
A kiszolgálókat egy SAN-hálózathoz ún. gazdaadapterekkel *(host adapter)* lehet kapcsolni. A nagyobb sebesség érdekében érdemes a 66MHz-es PCI buszokat használni erre a célra. Egy egyszerű fürt SAN rendszerrel az ábrán látható módon nézne ki.



Kiszolgálófürt SAN-nal

Ez szép is, jó is, olcsó is, csak egy apró gondja van.

A fürtszolgáltatást azért telepítjük, hogy hiba esetén a beépített és felkészített redundancia megakadályozza a szolgáltatás kiesését. Csakhogy a szépen felépített szoftverredundanciánkat tönkretettük azzal, hogy egyetlen hubra bíztuk a teljes adathálózat kommunikációját. Ez bizony hiba. A tervezés során módszeresen irtani javallott a „single point of failure"-t, vagyis az egy pontból eredő megbízhatatlanságot, költői fordításban az Achilles-sarkot. A teljes redundanciának persze megint csak ára van, de némi mérlegelés után belátható, hogy ez már nem nagy tétel. Célszerű redundáns vezérlővel ellátni a lemezalrendszert is, be kell építeni egy további SAN hubot *(vagy switchet)*. A „mindent a felhasználóért" konfiguráció valahogy így néz ki:



Redundáns SAN hálózattal Achilles ellen!

Gyakorlati tapasztalatként annyit fűzök a fenti ábrához, hogy nem minden gyártó rendelkezik olyan SAN hubbal, amelyhez data router csatlakoztatható, ekkor viszont az általuk kínált switch árfekvése némileg kedvezőbb a konkurenciáénál. Vásárlás előtt érdemes leülni a lehetséges szállítókkal, és alaposan végigtárgyalni az igényeket. Meggondolandó továbbá, hogy a data router vajon mindkét hubhoz csatlakozzon-e. Amúgy is csak egy van belőle, ráadásul „csak" a mentést biztosítja, vagyis a rendelkezésre állásával szembeni követelmények – tartalék eszközről lévén szó - mindenképp alacsonyabbak, mint a kiszolgálók esetén.

A lemezalrendszer kontrollere képes hardveres redundancia létrehozására (RAID 1, RAID5, RAID 0+1), melyre nagy szükség van, a cluster ugyanis nem kezeli a szoftveres merevlemeztömböket. Hogy ez miért van így, arról még értekezünk. Látható, hogy a kiszolgálófürtök beszerzése nem a „Józsi-most-van-akció-vegyetek-egy-szervert" alapon történik. Alaposság, megfontoltság, tervezés – ezek a fő feltételei annak, hogy az ember nekiugorjon a farkasfalkának. Legközelebb meg is tesszük.

Lepenye Tamás (MCSE)
lepenyet@mal.hu

Névfeloldás

Mi az, hogy névfeloldás, és hogy működik? Miért van rá egyáltalán szükségünk? Milyen sorrendben történik a névfeloldás, és miért? Miért kell ezt nekem ismernem? Nagyjából ezekre a kérdésekre kaphatjuk meg a választ a cikket végigolvasva. Gondolom most sokan hátradőlnek és azt mondják: ja kérem, ezt én mind tudom, nincs erre szükségem. Ennek ellenére mindenkinek azt tudom csak tanácsolni, hogy tessék elolvasni, az ismétlés sosem árt; illetve szeretnék a Windows 2000-rel kapcsolatban néhány hasznos built-in segédeszközt ajánlani. Kezdjünk is bele!

Miért kell erről beszélnünk?

A magyarázat, csakúgy mint a folyamat, igen egyszerű. A számítógépek kommunikációjuk során egy előre kiválasztott protokollon keresztül kommunikálnak egymással *(cikkem során csak a TCP/IP névfeloldásával foglalkozom)*. Tehát amikor gépemen megpróbálom megnézni a www.microsoft.com címet, akkor a kapcsolat ideje alatt az én gépem látszólagosan a www.microsoft.com címmel folytat adatátvitelt, valójában pedig a www.microsoft.com címhez tartozó TCP/IP címmel veszi fel a kapcsolatot. A folyamatot, melynek során egy gép egy másik számítógép *(gyakorlatilag bármely hálózatra kapcsolt, kommunikálni képes eszközre igaz)* nevéhez tartozó IP címet keresi, névfeloldásnak nevezzük. A névfeloldás folyamata igen összetett és elég sok lépésből áll. Pontosan az összetettsége miatt szükséges megismerkednünk vele *(továbbá a Windows 2000 AD alappilléreként emlegetett DNS kiszolgáló adatbázisában történő keresési folyamat is a névfeloldás egy fajtája)*. Megtudtuk, hogy miért fontos a névfeloldás ismerete, most nézzük konkrétan a névfeloldást!

Konkrétumok

Napjainkban a legtöbb számítógépet háromféleképpen lehet elérni:

- NetBIOS név alapján
- FQDN *(fully qualified domain name)* segítségével
- IP cím szerint

Mind a három módszerhez más-más szolgáltatás használatára van szükségünk. Kezdetben zavaró lehet, hogy a Windows 2000 nem tisztán használja a három névfeloldási módszert, hanem - nekünk akar segíteni a jámbor – ha az egyik nem megy, áttér a másikra. Ezzel óriási galibát is tud okozni, mert ha valaki nem tudja, hogy hibás DNS-sel is lehet pingelni *(csak sokkal lassabb a névfeloldás)* és bedöglött WINS-sel is lehet távoli megosztott könyvtárakra csatlakozni, jókat fog éjszakázni. Íme néhány szolgáltatás névfeloldási preferenciája:

Szolgáltatás	Elsődleges névfeloldás	Ha baj van...
PING	DNS	NetBIOS
net use * \\gep\share	NetBIOS	DNS
Active Directory	DNS	... baj van
Outlook	DNS	NetBIOS

Ez a táblázat csak tájékoztató jellegű, majdnem minden átállítható. Az Active Directory az egyetlen következetes jószág: ha nincs DNS, megkukul.

De induljunk el és nézzük meg mind hármat *(kettőt. Az IP->név feloldás most kimarad)*!

NetBIOS név és a hozzá tartozó névfeloldási rendszer

Az 1980-as években a Microsoft és az IBM közösen fejlesztette ki a NetBIOS névteret, a Windows9x és a Windows NT alap névfeloldásként mind a mai napig a NetBIOS használatát támogatja. A NetBIOS egy 16 karakterből álló név, amit tetszésünk szerint megválaszthatunk. Természetesen törekednünk kell az egyediségre, továbbá a beszédes nevek használata megkönnyíti a hálózaton történő kommunikációt. Érdemes egy hálózat tervezésekor annak névkonvencióját is megtervezni. Azt mondtuk, hogy 16 karakter hosszú lehet egy NetBIOS név, ami igaz is, de a gyakorlatban ebből csak 15 karakter használható. A tizenhatodik egy speciális karakter, aminek a szerepe az, hogy a névhez tartozó számítógépen futó szolgáltatásokat a hálózaton egyértelműen elérhetővé tegye. A 16. karakter a felhasználókhoz GUI *(graphic user interface)* szinten nem jut el *(így emiatt valójában egy NetBIOS név legfeljebb 15 karakter lehet!)* Tehát a NetBIOS név fontos, mert egyedi, egy géphez csak egy tartozhat, és egy hálózatban két azonos NetBIOS nevű gép nem működhet. De hogy lesz ebből névfeloldás?

Minden számítógép, melynek van NetBIOS neve, induláskor azt megpróbálja érvényesíteni, regisztrálni. Ilyenkor ellenőrzi, hogy a hálózaton valaki használja-e már az általa preferált NetBIOS nevet. Amennyiben a név szabad, azt hitelesíti, regisztrálja a nevet és a hozzá tartozó szolgáltatásokat. A másik lehetőség, hogy már valaki használja azt a nevet. Mi történik ilyenkor? A válaszhoz közelebb kerülünk, ha megnézzük, mi történik akkor, amikor senki sem használja a nevet: a név és a gép szolgáltatásai regisztrálódnak. Ütközéskor a gép nem tudja majd hitelesíteni a nevet és a hozzá tartozó szolgáltatásokat. Számítógépünk hibát fog jelezni, hogy már létezik egy azonos nevű számítógép a hálózaton, de ettől függetlenül elindul. Mi lesz a távolabbi következménye ennek? Mivel nem tudta regisztrálni a gépnevet, ezért a NetBIOS kommunikációról le kell mondanunk és a NetBIOS-hoz kapcsolódó szolgáltatások sem indulnak el. A probléma könnyedén áthidalható, gépünknek egyszerűen csak más nevet kell adnunk, majd újraindítjuk és már kész is. Amennyiben sikerült gépünknek olyan nevet adni, ami egyedi a hálózaton, gépünk a NetBIOS nevét érvényesíti és a 16. karakter alapján a tallózó *(magyarul: Browse :-) – a szerk.)* szolgáltatás segítségével a hálózaton a szolgáltatásait hirdeti, regisztrálja. Mik lehetnek ezek a szolgáltatások? Ilyen például a Messenger, melynek segítségével a bejelentkezett felhasználónak rövid üzenetet küldhetünk. *(net send username üzenet)* De ilyen még a Server és a

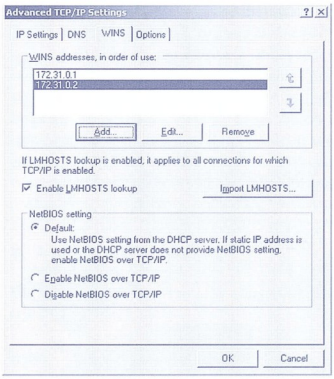
Workstation szolgáltatás is *(nem összekeverendő a Windows NT 4.0 különböző termékeivel, ezek szolgáltatások)*. Tehát alapértelmezésben: akinek van NetBIOS neve, az hirdeti magát és szolgáltatásait a hálózaton. Amikor egy számítógéppel a hálózaton keresztül szeretnénk kommunikálni, először meg kell tudnunk a névhez tartozó TCP/IP címet, vagy a címhez tartozó nevet. Ilyenkor a gépünk egy olyan kérdést küld a hálózaton keresztül, amit a hálózat minden tagja megkap *(Broadcast)*. Ha szeretnénk lefordítani emberi nyelvre ezt az üzenetet, akkor valahogy így hangzik: Szóljon nekem az a számítógép, és küldje el az IP címet, akinek a neve srv01. *(Ilyenkor az srv01-hez tartozó IP címet keressük.)* A hálózatunk, vagyis alhálózatunk minden számítógépe megkapja a kérdésünket, és valamennyi értelmezi is. Nyilván egy kivételével mindegyik eldobta a kérdést *(vagy lehet, hogy senki nem válaszol)*, mert nem ők az srv01 NetBIOS név tulajdonosai. Igen egyszerűnek tűnik, de gondolkodjunk egy kicsit! Egy 172.31.0.0 255.255.0.0 hálózat esetén az alhálózatunkban akár több, mint 240 db számítógép lehet *(sőt, több!)*, mindenki megkapja kérdésünket és abból csak egy számítógép válaszára várunk. *(Az olyan jellegű csomagokat, amit egy egész alhálózat minden tagja megkap, Broadcast üzenetnek hívjuk. Természetesen akár több alhálózatra is átjuthat, ez csak a hálózati útválasztók (router) konfigurációjától függ.)* Beláthatjuk, hogy ez felesleges forgalmat generál a hálózaton, amit könnyedén csökkenteni lehet. A hálózati útválasztók legáltalánosabb konfigurációjuk alapján az ilyen üzeneteket nem továbbítják, mert ezek könnyen a hálózat teljes terhelését eredményezhetnék és tenné mindezt úgy, hogy nem is hasznos forgalomról van szó, vagy ha hasznos, akkor is csak igen kis százaléka.

Foglaljuk össze az eddigieket egy kicsit: minden gép rendelkezhet NetBIOS névvel. A NetBIOS névhez tartoznak szolgáltatások, és mind a NetBIOS név, mind a névhez tartozó szolgáltatások listáját a tulajdonos gép regisztrálja és hirdeti. Amennyiben egy NetBIOS névvel szeretnék kommunikálni a hálózaton, szükséges ismernem a NetBIOS névhez tartozó TCP/IP címet. Amikor ezt a hálózaton szeretném megtalálni, egy Broadcast típusú üzenetet küld el a számítógépem, amiről megtudtuk, hogy ugyan biztosra megy, de felesleges üzenetekkel telítődik a hálózat a használata esetén. Kell lenni valami más megoldásnak is, mint a Broadcast, és van is. Mi a legnagyobb probléma jelenleg? Nincs a hálózaton olyan kitüntetett funkciójú gép, amely egy listát vezetne a NetBIOS nevekről és a hozzájuk tartozó szolgáltatásokról; nincs olyan gép, akinek címezhetnénk a kérdésünket. Rögtön rá is találtunk a megoldásra. Egy olyan gépre van szükségünk, ami a NetBIOS neveket egy központi adatbázisban tartja.

Windows Internet Name Service

Ilyen szolgáltatás a Windows NT Serveren vagy Windows 2000 Serveren futó WINS. A WINS kiszolgáló egy olyan gép, ami fogadja az ügyfélgépek névregisztrációját, és azokat egy JET típusú adatbázisban tárolja, frissíti.

Ügyféloldalon szükséges egy WINS kliens, amit a Microsoft TCP/IP implementáció tartalmaz. Csak annyit kell beállítani, hogy mi a WINS kiszolgáló TCP/IP címe és már kész is.



TCP/IP protokoll WINS adatlap

Tehát amennyiben van egy WINS kiszolgálónk és egy helyesen konfigurált WINS ügyfelünk, a munkaállomás regisztrálja az IP címet a WINS adatbázisában minden induláskor, majd minden sikeres leálláskor ki is veszi onnan *(deregisztrálja)*. A WINS kiszolgáló adatbázisa olvasható is a WINS ügyfelek számára, és amennyiben a kliens számítógép keresi az srv01-hez tartozó IP címet, előbb megkérdezi a WINS kiszolgálót, hogy milyen IP cím tartozik a névhez. Természetesen azt azért nem mondhatjuk, hogy a Broadcast típusú üzenetszórásra már nincs is szükségünk, mert ez nem így van. A WINS kiszolgáló adatbázisa csak a megfelelően beállított gépekről tartalmaz adatokat, és csak az általa regisztrált címekről *(természetesen lehet több WINS kiszolgálót szinkronizálni egymással, és akkor további WINS kiszolgálók által regisztrált címekről is lesz információ)*. Tehát ha az srv01 nem WINS kliens, vagy nincs rendesen beállítva, abban az esetben azt a WINS adatbázisban nem is lehet megtalálni – csak Broadcast típusú üzenetszórással. Ezenkívül, létezhetnek még olyan esetek amikor nem kell használni a WINS adatbázist, például:

- Ha a keresett név a sajátgép neve
- Ha nem NetBIOS név amit keresünk

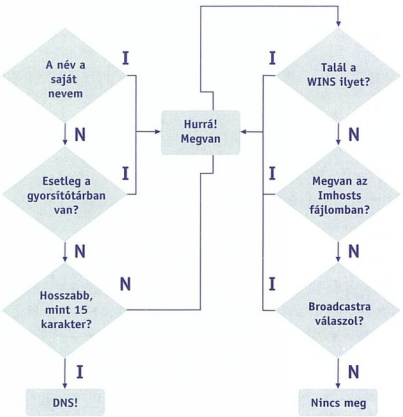
Meg kell meg említeni, hogy a WINS kiszolgáló szolgáltatásnak van egy helyileg minden kliens gépen elérhető kicsinyített változata – egy fájl. A

```
%SystemRoot%\System32\Drivers\etc
```

könyvtár alatt található az lmhosts nevű fájl. Ebbe kézzel felvehetünk NetBIOS neveket és IP címeket, amiket kliensgépünk majd képes lesz feloldani. Használata csak speciális esetekben ajánlott, mivel az lmhosts fájl csak az adott gépen érvényes, és ha a hálózatunk minden számítógépén azonos névfeloldást szeretnénk használni, akkor azonos lmhosts fájlt kell*(ene)* használni. Továbbá az lmhosts fájl statikus, minden változtatás kézimunkázást igényel.

Tehát nem kell használnunk a WINS adatbázist akkor, ha a NetBIOS gépnév megegyezik a sajátgépünk nevével. Ez azért lehetséges, mert minden számítógép a saját nevéről

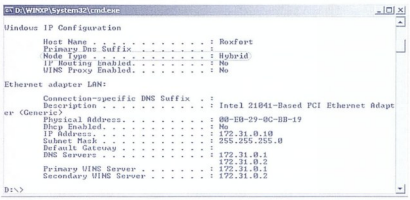
tudja, hogy az a sajátja. A másik lehetőség pedig az, amikor nem NetBIOS névről van szó. Emlékszünk milyen név lehet meg? Például FQDN, azaz DNS név *(www.kutyafule.hu)*! Azt nem a WINS adatbázisban kell keresni ... de honnan tudhatja a gépünk, hogy mi a NetBIOS név és mi FQDN? A NetBIOS név maximum 15 karakter hosszú lehet, az ennél hosszabb nevek biztosan FQDN nevek lesznek, amiket a DNS kiszolgálón kell keresni. Olvastunk már pár szabályt, amiből a NetBIOS névfeloldás sorrendjét is megállapíthatjuk:



A NetBIOS névfeloldás blokkdiagramja

Befolyásolhatjuk kliensünket, hogy milyen sorrendben és hol keressék a neveket. A NodeType határozza meg egy gép számára, hogy milyen sorrendben próbálja meg feloldani a kéréseket. A következő node-type-ok léteznek:

- B-node *(Broadcast)*: minden kérést Broadcasttal próbál meg feloldani. Ha ez nem sikerül, az LMHOSTS fájlba is belenéz
- P-node *(Point-to-point=WINS)*: a kliens minden kérést a WINS kiszolgálóhoz küld, a névfeloldás során más kiszolgálót nem használ. Még kínjában sem broadcastol. Nagyon rossz módszer, mert ha nincs elérhető WINS kiszolgáló, a gép nem lát semmit a hálózaton.
- M-node *(Mixed=B+P)*: A P és a B típus keveréke. A kliens először broadcastol, majd ha sikertelen a feloldási kísérlet, akkor a WINS kiszolgálóhoz fordul. Nincs DNS kiszolgáló a névfeloldásban, természetesen a 15 karakternél hosszabb neveket nem is tudja a rendszerünk ekkor feloldani.
- H-node *(Hybrid=P+B+DNS)*: Hasonló az M-node-hoz csak a sor végén ott van a DNS kiszolgáló. Természetesen ha egy név hosszabb mint 15 karakter azonnal a DNS kiszolgálóhoz továbbítja a kérdést.



H-node-type ügyfél

Az ipconfig /all paranccsal könnyen lekérdezhetjük gépünk node-type-ját.

Egy alapértelmezésben telepített kliens b-node típusú, ilyenkor nincs WINS, vagy DNS kiszolgáló használatára beállítva. Amint egy WINS kiszolgáló címét beállítjuk, a gép egyből átáll h-node-ra. A node-type a DHCP kiszolgálóval központilag szabályozható minden munkaállomáson A fenti lépések mindegyikét megbeszéltük, egy kivételével. Nem beszéltünk még az FQDN-ről. Nézzük meg most azt is egy kicsit közelebbről!

FQDN és annak feloldása

A FQDN a fully qualified domain name rövidítése, ami annyit jelent, hogy teljesen hitelesített név. Azért van szükség erre, mert ez teszi lehetővé, hogy egy név valóban egyedi legyen. Míg a NetBIOS is egyedi, az FQDN is az, csak a hatáskörük más egy kicsit. Az Internetes kommunikáció során a NetBIOS névteret nem használjuk, ennek több oka is van. Az egyik a névtér tulajdonságából adódik: a NetBIOS névtér sík, hierachia bevezetésére nem alkalmas. Ez azt jelenti, hogy párezer gépen túl „használhatatlanná" válik, mert szinte képtelenség olyan sok teljesen egyedi, de teljesen értelmes gépnevet kiválasztani. Az FQDN névnél többszörös összekapcsolást is csinálhatunk pl.: mail.geniusgroup.hu. Ha a NetBIOS név lenne az Internetes kommunikáció elfogadott névkonvenciója, akkor a „mail" nevet csak egyetlen egy gép használhatná :-) Valljuk be, ez egy kicsit szegényes megoldás lenne. E helyett használjuk a DNS-t, amiben összekapcsolhatjuk egy host nevét egy domainnévvel, és ez a hosszú név biztos, hogy egyedi lesz, továbbá a „mail" hostnév használható bármely tartománynév esetén – mint ahogy minden faluban lehet Kovács Pista. Tehát mint láthatjuk, az FQDN némileg hasonlít a NetBIOS névre és a NetBIOS névfeloldásra. Az FQDN feloldásakor szintén az IP címet keressük, csak ilyenkor nem a WINS kiszolgálóval kommunikálunk, hanem a DNS-sel. A Windows 2000 operációs rendszer alap névfeloldási rendszerkent már a DNS-t használja *(lassan leszokunk a NetBIOS-ról)*. Sokat, rengeteget változott a Windows NT 4.0-ban található DNS Serverhez képest a Windows 2000 azonos szolgáltatása. Míg NT 4.0-ban a DNS kiszolgáló adatbázisa egy statikus adatbázis volt, addig a Windows 2000-ben található DNS kiszolgáló képes a WINS-hez hasonlóan dinamikus frissítésre és a DNS nevekhez tartozó szolgáltatásokat is képes eltárolni. Mi több, az Active Directory egyik alappillére a DNS kiszolgáló. Mit jelent ez? Gyakorlatilag egy homogén Windows 2000-es rendszerben nincs szükség többé a WINS kiszolgálóra, a NetBIOS kommunikációra. Használhatjuk helyette az FQDN-t mert a DNS kiszolgálóink adatbázisában minden szükséges információt megtalálunk!

Azért a Windows NT 4.0-ban is volt lehetőségünk az FQDN-ek használatára a hálózatunk minden számítógépének elérésére. A DNS kiszolgálók képesek a WINS kiszolgálóval történő kommunikációra. Ez azt eredményezi, hogy ha egy DNS zóna alatt a keresett host nem elérhető, a DNS kiszolgáló a hostot megpróbálja feloldani a WINS adatbázisból. Így nem szükséges minden gépet felvenni manuálisan a DNS zónába. Természetesen a Windows 2000 alapú DNS kiszolgáló támogatja a dinamikus frissítést és így nem szükséges a WINS-LOOKUP használata. *(Egyébként a DNS-WINS együttműködés problémákat is okozhat, mert egy WINS típusú rekordba kerül be, hogy hol van a WINS Server. Ezzel az a baj, hogy a DNS szabványok nem tartalmaznak WINS rekordtípust. Van A, meg MX, meg NS – de WINS az NINCS! Ha egy ilyen „fertőzött" zónafájlt valamilyen „ellenséges" DNS Serverre szeretnénk replikálni, az jó eséllyel visszautasítja a számára értelmezhetetlen adatot! Láttam én már ilyet!)*
Áttekintettük a névfeloldás témakörét. Úgy érzem, hogy ezen ismeretek elengedhetetlenül fontosak egy sikeres TCP/IP hibakeresés eseten. Ha hibakeresés akkor hibakeresési eszközök:

Ping

ICMP kommunikáció, sok hasznos kapcsolóval. Használatával két gép közti ICMP kommunikáció megléte ellenőrizhető. Windows 2000-es változata már egy statisztikát is közöl a felhasználóval. Vigyázat, MINDIG először DNS névfeloldást választ, még rövid nevek megadása esetén is! Ha irgalmatlanul lassan, de mégiscsak válaszol, akkor a következő történik: Vár-vár-vár a DNS-re, majd X idő múlva megunja, és áttér a NetBIOS feloldásra – ami akár Broadcasttal is sikerülhet!
Használata: ping állomásnév

Tracert

Hasonlóan a ping-hez itt is ICMP kommunikáció meglétét ellenőrizhetjük. Amiben több, az az hogy a két gép közötti összes útválasztó meglétét is láthatjuk.

Pathping

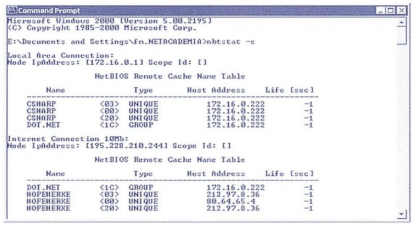
Csak Windows 2000 alatt elérhető eszköz. A ping és a tracert parancsok keveréke és mind két eszköz jó tulajdonságait magában hordozza.

Nbtstat

A TCP/IP protokoll, a NetBIOS Cache és a hálózati csatoló tulajdonságai kérdezhetők le vele. Például:

```
NBTSTAT -c
```

A parancs hatására megjelenik a *(két hálózati kártyás)* gépem NetBIOS Cache tartalma, benne nemcsak a nevek, hanem az az ominózus 16. bájt, a szolgáltatáskód!



A NetBIOS Cache tartalma

Az alábbi táblázatban láthatók a legfontosabb szolgáltatáskódok:

0x00	Workstation szolgáltatás
0x03	Messenger
0x20	Server
0x1c	Tartományvezérlő

A teljes NetBIOS szolgáltatáslista az **[1]** címen olvasható *(NetBIOS suffixes)*. Sohanemvolt-hackerként ehhez annyit tennék hozzá, hogy a –a kapcsolóval távoli gépek NetBIOS Cache-e is lekérdezhető, és például a 0x03 rekordra pillantva rögtön látszik, ki van bejelentkezve – hisz Messenger rekordot a felhasználó regisztrál. Talán nem véletlen, hogy a Google erre a keresésre „netbios tcp port" első találatként egy hackerszájtot dob fel **[2]** :-). A NetBIOS szolgáltatások Internet felőli láthatóságát célszerű korlátozni! Ne engedje át tűzfalunk a 135-139 portokat!

Összegzés:

A névfeloldást és a névteret megfelelően meg kell tervezni egy tartomány építése előtt. Olyan terület ez, ami nagyon befolyásolja a géppark helyes működését. Szerteágazó és nehéz feladat. Cikkem során ennek csak egy kis részébe nyertünk bepillantást, lényegesen nagyobb feladat mint amilyennek ez elsőre látszik. Szeretném megjegyezni, hogy hiába ajánlás a WINS elhagyása és 10 számítástechnikusból x azt mondja, hogy ne használjuk a WINS-t és vele együtt a NetBIOS-t, én azt mondom, hogy kompatibilitási okok, illetve a redundancia miatt a WINS megmaradhat. Sokan félnek a WINS-től, de úgy gondolom ahogy a királykobra is veszélyes, mégis a mérgéből komoly gyógyszereket készítenek, a WINS is lehet hasznos, ha megfelelően kezelik.
Reményeim szerint közelebb kerültünk egy kicsit a névfeloldási folyamat megismeréséhez és az elméleti tudás gyakorlatba való átültetésével a napi munkánk egy kicsit könyebb lesz. Aki pedig ezt mind tudta, azoknak pedig egy könnyed olvasmány volt.

Harmath Zoltán
zoli@geniusgroup.hu
MCSE, MCP+i

A cikkben szereplő URL-ek:
[1] http://support.microsoft.com/support/kb/articles/Q163/4/09.asp
[2] http://www.networkice.com/advice/Exploits/Ports/

Remote Installation Services *(I. rész)*

Mi is a RIS valójában? Egy Windows 2000 beépített eszköz, amely az ügyfélgépek üzembehelyezését hivatott megkönnyíteni.
Segítségével úgy telepíthetünk Windows 2000 és Windows XP munkaállomásokat – amelyekben sem floppy, sem CD nincs *(de egy HDD azért persze van)* - hogy nem is kell közvetlen rendszergazdai beavatkozás.
A Windows NT 4.0 környezetben megismert, asztali gépek távoli telepítések lehetőségeihez képest mindenképp nagy előrelépés ez! Megvalósítható hogy maga a felhasználó akár telefonos segítséggel is el tudja indítani a telepítést, és valóban hozzá sem kell nyúlnia, pikk-pakk minden szükséges programmal felvértezett Windows 2000 munkaállomás a végeredmény.
Mielőtt azonban az égbe emelném ezt a szolgáltatást, meg kell jegyeznem, hogy komoly korlátai is vannak, látni fogjuk hogy egy sor követelménynek kell megfelelni ahhoz, hogy egyáltalán használni tudjuk. Ha megteremtjük a feltételeket, akkor senki nem tarthat vissza attól, hogy valóban egységes, központilag szabályozott, rendezett munkaállomás parkot alakítsunk ki.

Mi megy? Mi nem megy?

Sajnos a RIS csupán Windows 2000 Professional vagy Windows XP telepítését teszi lehetővé, de azt legalább kétféle módon. Az egyik az ún. image alapú telepítés, a másik a scriptalapú.
A RIS-sel történő scriptes Windows 2000 Professional telepítés – ezt nevezik CD alapú telepítésnek is - sokban hasonlít a régről ismert hálózatos felügyelet nélküli telepítéshez, hisz egy kiszolgálóról a telepítéshez szükséges állományok az ügyfélre kerülnek, majd onnan a válaszokat tartalmazó állomány alapján telepítésre kerül az operációs rendszer és egyéb programok, ha mi is úgy akarjuk. Különbség a kettő közt a megoldás módjában van. Főként az gyorsítja a RIS-es telepítést, hogy PXE képességekkel rendelkező hálózati kártyával megáldott gépekez nem kell floppy vagy CD a telepítéshez. Ebben az esetben a RIS maga csak az operációs rendszert telepíti, a szükséges programok telepítéséről nekünk kell gondoskodni, persze ez is megvalósítható rögtön az operációs rendszer telepítése után, de ez plusz munkát igényel.
Az imagealapú telepítés Windows NT környezetben csak 3rd party eszközök segítségével volt megvalósítható, a RIS viszont lehetővé teszi ezt is – ezt hívják RIPREP telepítésnek -sőt, a Windows 2000 plug and play rendszere révén az etalonként használt, és a később telepített gépeken csak a HAL kell, hogy azonos legyen *(ACPI vagy nem ACPI)*, tehát nem szükséges teljesen egyforma hardverelemek megléte. Ez az image nem egy gigantikus állomány *(mint általában más gyártóknál)*, hanem tulajdonképp a háttértár első partíciója szőrőstül-bőröstül felkerül a kiszolgálóra a RIPREP.EXE futtatása során. Hátránya tehát, hogy csak egy partíció telepítését teszi lehetővé, tehát ha valaki előkészít egy masinát, számoljon vele, hogy csak a boot ill. system partíciót fogja tudni később telepítéshez használni; viszont nem korlátozza a CD-ROM lemezek limitált tároló kapacitása a telepítőkészlet előállítását. A RIS hátrányos tulajdonsága az is, hogy nem tesz lehetővé operációs rendszer frissítést, csak teljesen szűz telepítés végeztethető el vele. Ellentétben a scriptes telepítéssel, ezzel a módszerrel a RIS egy menetben telepíti az operációs rendszert és a rajta lévő programokat, hisz a mintaként használt gép egy partíciójának az egész tartalmát tudjuk újra meg újra felhasználni a telepítésekben.

Szükséges szolgáltatások

Még mielőtt belevágunk a telepítésekbe tényleg fontos, hogy a működéshez szükséges feltételek meglegyenek, egyébként vagy semmi nem fog működni, vagy később kerülünk szembe problémákkal.
Telepítés során az ügyfél DHCP-től kapja meg IP címét, ezért mindenképp kell DHCP kiszolgáló, ez azonban nem kell hogy Windows 2000 legyen. Miután a gép kapott IP címet, a DNS kiszolgálóhoz fordul hogy Active Directory tartományvezérlőt találjon, tehát egy DNS is kell a hálózaton, méghozzá olyan, amely támogatja az SRV rekordokat és lehetőleg a bejegyzések dinamikus frissítését is *(ez utóbbi nem feltétlen szükséges)*. Ha már az Active Directorynál tartunk: csak akkor fog Active Directory tartományvezérlőt találni az ügyfél - akitől megtudhatja ki is a RIS kiszolgáló – ha van Active Directory. Hogy ne keverjem tovább a dolgokat, kell tehát Active Directory, DHCP és DNS a RIS működéséhez. Ez nem azt jelenti, hogy a távoli telepítéseket végző kiszolgáló kell legyen a tartományvezérlő, a DHCP és a DNS kiszolgáló is egymagában, hanem azt, hogy a hálózaton elérhetőnek kell lennie mindhárom szolgáltatásnak. Bőven elég, ha a RIS egy tagkiszolgálón fut.
Szigorúan ajánlott azonban külön partíció létrehozása a távoli telepítésekhez használt állományok tárolására, mert telepítés után az adott partíciót a SIS *(Single Instance Store)* veszi uralmába. Később még szólok ennek működéséről, most csak annyit, hogy ennek segítségével egy rakás helyet spórolunk, mert minden többször előforduló állomány egyszer foglalja majd a helyet a RIS által használt partíción. Fogadjuk meg és készítsünk külön NTFS partíciót a telepítés előtt. A beállításhoz használt varázsló nem engedi, hogy system vagy boot partícióra kerüljön a RemoteInstall könyvtár, de nem figyelmeztet, ha az egyébként megjelölt partíción vannak más állományok. A partíció legalább 2Gb-os legyen, bár a telepítés nem fog ennyit elhasználni. Nem nehéz megtölteni, ha többféle telepítést készítünk elő, és mozgatása nem egyszerű a SIS miatt. Egy utolsó megjegyzés a RIS partíciójával kapcsolatosan, hogy a titkosítás *(EFS)* és a SIS nem szeretnek közösködni, nem működnek ugyanazon a partíción.

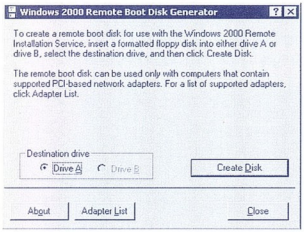
Ügyféloldali követelmények

A munkaállomásként használt számítógépekbe olyan hálózati kártya kell, amelyiken van Pre-Boot eXecution

Environment *(PXE)* ROM. Ez teszi lehetővé, hogy az ügyfél a kártyáról bootolva indítsa a telepítést.

Legkönnyebb dolgunk akkor van, ha a munkaállomások megfelelnek a PC98 szabványnak, ugyanis ilyenkor a hálózati kártyán helyből ott van a PXE ROM. Csupán a BIOS-ban kell beállítanunk a bootsorrendet, előre téve a hálózatos boot lehetőségét, majd a következő boot során már alkalmas a gép arra, hogy a RIS kiszolgálót elérjük vele. Ha nem ilyen masináink vannak, akkor is van mód a RIS használatára, ilyenkor PXE emulátort tartalmazó floppy-t gyárthatunk. Sajnálatos módon azonban a támogatott hálózati kártyák köre meglehetősen szűk. Kizárólag néhány PCI hálózati kártyához van mentőöv, sem az ISA - hiába plug and play -, sem a PCMCIA hálózati kártyák nem támogatottak.

A RIS-hez szükséges boot floppy a RemoteInstall\admin\i386 könyvtárban található rbfg.exe segítségével készíthető. Egy olyan floppy készül, amely az összes *(mind a 25 :-( )* támogatott hálózati kártyával rendelkező géphez használható. A lemezen egy picike fájlt *(89,3 kilobájt)* találunk, mely nem más, mint a boot ROM tartalma. Ebből következően a támogatott kártyák körét házilag bővíteni nem lehet, mert csak olyan jó nekünk, ami tudNA bootolni ROM-ról – csak éppen nincs bedugaszolva neki olyan.



A Remote Boot Disk Generator, RBFG.EXE

Akár van PXE boot ROM a hálózati kártyán, akár a fenti módszerrel készített floppyt használjuk, ugyanúgy történik a telepítés elindítása: a PXE segítségével. A floppy csupán emulálja a PXE környezetet a hálózati kártya helyett.

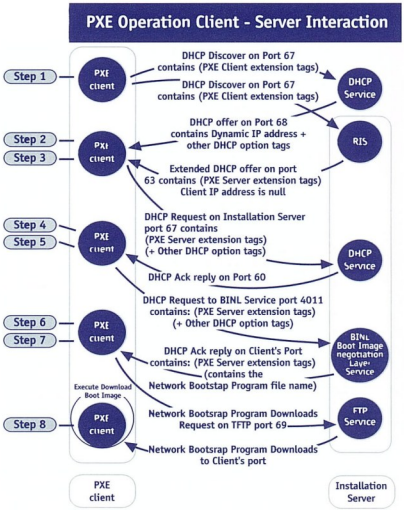
Mi is a PXE és hogy működik?

Ha mondjuk masinában levő háttértárról indítjuk a számítógépet, akkor a BIOS betöltése után - a boot sorrendben beállítottaknak megfelelően – az első eszközön *(mondjuk az első háttértáron)* megkeresi a bootszekort, betölti, majd az ott találtaknak megfelelően elindul az operációs rendszer, vagy annak menüje.

A PXE – teljes nevén Pre-Boot eXecution Environment – ha a BIOS boot sorrendjében a hálózat van legelöl, illetve meg van engedve a hálózatról való boot, a háttértárat vagy a floppyt is megelőzve a hálózaton keresztül a kiszolgálón elhelyezett boot szektort használva indítja a számítógépet. Nézzük, hogy is csinálja mindezt RIS kiszolgáló esetén! Az alábbi esetben külön DHCP és RIS kiszolgáló van a hálózaton.

1. lépés

Az ügyfél broadcast DHCP Discover üzenetet küld a szabványos DHCP portra *(67)*, DHCP és RIS kiszolgálót keresve. Az üzenetnek opcionális mezője is van, amelyben az ügyfél azonosítója (GUID), a hálózati kártya, valamint a munkaállomás architektúrájának azonosítója szerepel.



A PXE működése

2. lépés

A DHCP kiszolgáló egy ajánlattal válaszol az ügyfélnek, amely tartalmazza az IP címet és más DHCP opciókat. Most jön a csoda. A DHCP Discover üzenetre a RIS kiszolgáló is válaszol *(sic!)*, PXE kiszolgálóként megadja saját IP címét. *(Emiatt kell a RIS gépet ugyanúgy, és ugyanott Authorizálni, mint a DHCP Servereket: a DHCP Administratorral!)*

Ezen a ponton más DHCP és RIS kiszolgálók is válaszolhatnak az ügyfélnek. Ezek az üzenetek szabványos DHCP paramétereket tartalmaznak – az ügyfél IP címét, és amit a rendszergazda még beállított. Ha a DHCP a RIS kiszolgálón van telepítve, akkor a telepítő kiszolgáló által küldött DHCP válasz szintén tartalmazza a szabványos DHCP paramétereket *(ez gyakorlatilag az ügyfél IP címét jelenti)*.

Az ügyfél először olyan DHCP csomagot vár, amely vagy csak IP címet, vagy IP címet és egy PXE rendszertöltő kiszolgáló IP címét tartalmazza. Megjegyzi azonban a RIS kiszolgáló IP címét is, ezt később használni fogja.

3. lépés

Az ügyfélnek – miután feldolgozta a csomagokat - egy újabb üzenettel *(a DHCP Request csomaggal)* kérnie kell a számára felajánlott címet a DHCP kiszolgálótól.

4. lépés

A DHCP kiszolgáló nyugtázza ügyfél a kérését.

Vegyük észre, hogy eddig a pontig nem más történt, mint szabályos DHCP által történő IP címkiosztás *(plusz némi oldalról belekotyogás a RIS kiszolgáló által)*.

5. lépés

Az ügyfél egy DHCP Request csomagot küld a RIS kiszolgálón működő BINL szolgáltatásnak a 4011-es portra. A RIS kiszolgáló IP címét a 2. lépésben már megjegyezte. Ez a

csomag ugyanaz, mint az első lépésben leírt DHCP Discover csomag, annyi különbséggel, hogy ez most DHCP Requestként van kódolva. Jó mi?

6. lépés.

A telepítő kiszolgáló BINL szolgáltatása egy DHCP Acknowledge *(nyugtázás)* csomagot küld vissza az ügyfélnek, szintén a 4011-es portra, amelyben megküldi az indításhoz használt állomány nevét – RIS esetén startrom.com - és helyét, valamint az ügyfél egyedi azonosítóját *(GUID)*.

7. lépés.

Az ügyfél szabványos TFTP használatával letölti a végrehajtható állományt. Az, hogy melyik fájlt tölti le, valamint hogy a letöltött kód hol fog elhelyezkedni a memóriában, az ügyfél CPU architektúrájától függ.

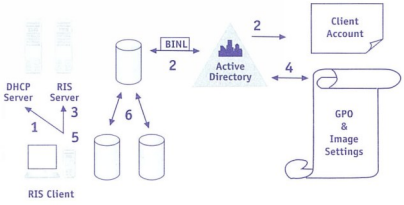
8. lépés.

A PXE ügyfél megkezdi a letöltött kód végrehajtását.

A fenti eset akkor megy végbe, amikor a DHCP kiszolgáló és a RIS kiszolgáló külön számítógépekről szolgálja ki az ügyfeleket. Ha a RIS kiszolgáló egyben DHCP is, akkor sokkal rövidebb ez a folyamat, mert a második lépésben a DHCP kiszolgáló helyből elküldi a RIS használatához szükséges paramétereket is. Az ügyfél a harmadik lépésben a DHCP Request csomagban kéri az IP címet, az IP beállításokat és a RIS használatához szükséges paramétereket is egyben, ezután a DHCP - ami ugye RIS kiszolgáló is -, egy menetben megküldi mind az összes kért paramétert egy DHCP nyugtázásban *(Ack)* csomagban.

Működés közben

OK. Ez volt maga a PXE, de hogy működik mindez RIS környezetben, amikor mi a munkaállomás előtt ülünk? Hogyan lesz ebből működő öperenciás rendszer? Hát nagyjából így:



A munkaállomás telepítésének folyamata

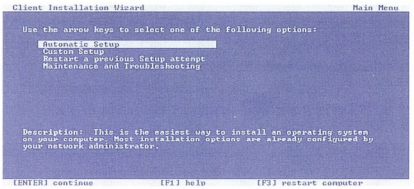
A munkaállomáson *(miután beállítottunk hogy először a hálózatról próbáljon indulni)* a BIOS betöltése után a képernyőn megjelenik a gép MAC címe, és a rendszer megkezdi az IP cím kérését, és a RIS kiszolgáló keresését. Az előzőekben tulajdonképp arról volt szó, amit itt az 1. nyíl jelöl, az ügyfél megbeszéli a DHCP kiszolgálóval, hogy az adjon neki IP címet és mindent, amit a rendszergazdi beállított a DHCP kiszolgálón.

A RIS kiszolgáló az Active Directoryhoz fordul *(2. nyíl)* és leellenőrzi a számítógép accountját a tartományban, mindezt a gép által a DHCP Discover csomagban elküldött azonosítója alapján teszi *(GUID)*. Megnézi van-e már számítógéphez Computer Account. Ha talál, akkor később azt fogja használni, ha nincs az sem baj, lehet létrehozni később a telepítés során. Ha talált RIS kiszolgálót és ott minden rendben, felszólítja a felhasználót az F12 funkciógomb lenyomására. A felhasználónak 4 másodperce van erre, egyébként a rendszerindítási lista következő eleméről folytatódik a rendszer indítása. Bizonyos rendszerekben úgy is be lehet állítani a PXE egyeztetést, hogy a felhasználónak meg kelljen nyomnia az F12 funkciógombot. Ebben az esetben a felhasználónak kétszer kell megnyomnia az F12-t – egyszer a PXE egyeztetés aktiválásához, és másodszor, amikor a RIS kiszolgáló válaszol. Ha minden ügyfél támogatja az F12-t, akkor a Startrom.com fájlt az Oschooser\I386 könyvtárban kicserélhetjük a Startrom.n12 fájlra, így nem kell az F12-t másodszor is megnyomni.

Tehát az F12 után a kiszolgálóról letöltődik TFTP-vel a startrom.com *(3. nyíl)* ami tulajdonképp a boot szektor majd az első CIW – Client Installation Wizard - képernyő a Welcome.osc – köszönti a felhasználót.

Ezután a bejelentkezési képernyő *(Login.osc)* jelenik meg. A képernyők a választásoknak megfelelő sorrendben töltődnek le. Miután a felhasználó bejelentkezett, a RIS kiszolgáló a megadott felhasználói név és jelszó alapján ellenőrzi az Active Directoryban, mit ajánlhat fel a következő képernyőkön *(az adott felhasználó melyik operációs rendszert úgy értem melyik telepítőkészlet telepítheti a sok közül, ami a kiszolgálón található. 4. nyíl)*.

Amennyiben a felhasználónak az van beállítva, hogy ő maga nem választhat, akkor számára a Group Policy segítségével megjelölt telepítőkészlet fog települni, ha azonban van választási lehetősége, akkor az alábbiak *(5. nyíl)*:



Választási lehetőségek

- Automatic Setup *(Automatikus telepítés)*: A felügyelet nélküli alapértelmezett beállításokkal telepíti az operációs rendszert, a felhasználótól csak a telepítéshez használandó telepítőkészlet kiválasztását kéri. Az automatikus telepítéshez használt Setup Information File-t *(SIF)* testreszabhatjuk szervezetünk igényeinek megfelelően.
- Custom Setup *(Egyedi telepítés)*: Lehetővé teszi a felhasználóknak, hogy megadják a számítógép nevét és azt, hogy az újonnan létrehozott Computer Account objektum *(CAO)* melyik OU-ba kerüljön. Ha ezek a mezők üresen maradnak, akkor RIS kiszolgáló által megadott alapértelmezések lépnek életbe *(mint az automatikus telepítés esetében)*. Fontos megjegyezni, hogy ha az egyedi telepítés során megadott név és hely megegyezik egy létező CAO nevével és helyével, és az ügyfél GUID-je egyezik a létező CAO GUID-jével, akkor a létező CAO-t a Telepítő felhasználja. Ha a GUID, a név és a hely közül csak egy egyezik meg, akkor „duplicate name" vagy „duplicate GUID" hibaüzenetet kapunk.
- Restart a previous Setup attempt *(Megszakadt telepítés folytatása)*: Ha a telepítés még a grafikus mód *(GUI)*

elérése előtt megszakad *(például áramszünet vagy a hálózati kapcsolat hibája miatt)*, ennek az opciónak a kiválasztásával folytathatjuk a félbeszakadt telepítést, így a felhasználó átugorhatja a már kitöltött képernyőket. A Telepítő indulása után a használt válaszfájl bekerül egy könyvtárba a távoli telepítőkészleteket tartalmazó köteten *(ez a távoli telepítő könyvtárfájának Tmp könyvtára)*. A fájl neve a telepítést végző rendszer GUID-je lesz *(vagy 24 darab nulla plusz a MAC címe, ha az ügyfél a PXE-emulátor floppylemezt használva indult)*, kiegészítve a .sif kiterjesztéssel.

- Maintenance and Troubleshooting *(Karbantartás és hibael hárítás)*: A más gyártók által készített karbantartó és diagnosztikai eszközökhöz biztosít hozzáférést. Ilyen például a BIOS frissítése vagy a távirányítás.

A következő képernyőn megjelenik a választható operációs rendszer telepítőkészletek listája. Ha csak egy van, akkor az automatikusan kiválasztódik, és folytatódik a telepítés. Ezután egy üzenet figyelmeztet bennünket, mely szerint a program megformázza a helyi merevlemezt és a rajta lévő adatok elvesznek, kivéve, ha a SIF tartalmaz olyan bejegyzést, hogy ne particionálja újra a merevlemezt. *(Ez a bejegyzés a SIF fájlban a Repartition = Yes|No. Ha a No értéket adtuk meg, akkor a program nem formázza meg a lemezt és a figyelmeztető üzenet sem jelenik meg.)* Az ENTER lenyomása után a következő információ jelenik meg az utolsó képernyőn:

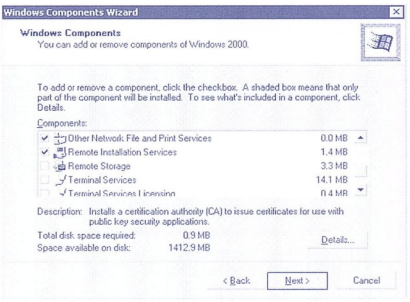
- A létrehozott CAO
- A GUID
- A telepítőkiszolgáló neve *(az a RIS kiszolgáló, amelyről a telepítőkészlet származik)*

Amennyiben csak elő akarjuk készíteni a számítógépet, akkor ezen a ponton kikapcsolhatjuk. Ennek a módszernek a használatával kicsomagoljuk a gépet a dobozából, elindítjuk, hogy létrejöjjön a CAO-ja, majd átadjuk a felhasználónak. Így elkerülhető a CAO felhasználó általi elírása. A RIS kiszolgáló észre fogja venni, hogy a GUID már hozzá van rendelve egy CAO-hoz és az abban levő információk felhasználásával fogja befejezni a telepítést.

Még egy ENTER és elindul a telepítés *(6. nyíl)*, innen már nincs visszaút. Körülbelül egy óra kell ahhoz, hogy települjön az operációs rendszer, persze ez az idő változhat a hálózat sebességétől, ill. a gép egyéb paramétereitől függően. A RIPREP módszerrel előkészített telepítések esetén hamarabb végez.

Kiszolgálótelepítés

Készítsünk RIS Servert! Maga a telepítés nem nagy kunszt: mint a legtöbb Windows 2000 szolgáltatást, az Add/Remove Programs – Add/Remove Windows Components menüjéből telepíthető a RIS.



Add/Remove Windows Components

A RIS-t telepíti a SIS szűrőmeghajtót, és létrehoz egy könyvtárat a Windir\System32\Reminst könyvtár alá, amely a Risetup.exe futásához szükséges állományokat tartalmazza. Ha RIS-t a Telepítő futtatása után adtuk hozzá, újra kell indítanunk a gépet a SIS NTFS szűrő telepítéséhez.

A Risetup.exe futtatása

Az előzőekben telepített RIS kiszolgáló még korántsem működőképes: még egy rakás beállítás hátra van. Ebben segít a risetup.exe

A Risetup.exe fájlt kétféleképpen lehet futtatni:

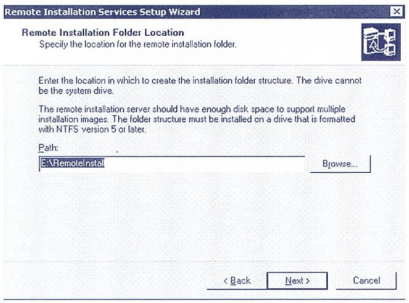
- A Configure Your Server képernyőről, amikor a számítógép a telepítés után újraindul *(lásd az előző bekezdésben)*.
- A Start menüben a RUN (futtatás) menüpontra kattintva írjuk be, hogy risetup.exe, majd kattintsunk az OK gombra.

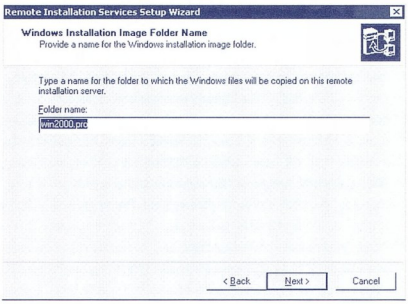
A Risetup.exe indítása után számos párbeszédablakkal találkozunk. Az első *(lásd alább)* egy üdvözlő képernyő, amely a RIS számára szükséges összetevőket sorolja fel.

A RIS varázslója

A következő párbeszédablakban *(lásd alább)* azt a könyvtárat kell megadnunk, amelyben a RIS az ügyfelek telepítéséhez szükséges telepítőkészleteket – image-nek is hívják - tárolni fogja. A Risetup automatikusan megkeresi az első olyan NTFS kötetet, amely nem rendszerindító. Bármely NTFS kötetet ki lehet választani, amely nem tartalmazza a rendszer indításához szükséges fájlokat *(Boot.ini, Windir könyvtár)*.

A könyvtár neve csak „alacsony" ASCII karaktereket tartalmazhat. A könyvtár lehet csatolt *(mount)* köteten, feltéve, hogy az NTFS formátumú, és nem rendszerkötet.



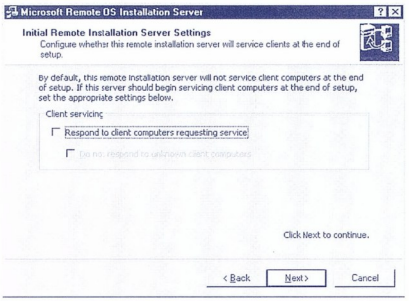


A reminst megosztás helye a háttértáron

A következő képernyőn *(lásd alább)* megadhatjuk, hogy a kiszolgálónk válaszoljon-e a hozzá érkező kérésekre. Korlátozhatjuk a válaszokat a PXE-képes ügyfelekre vagy csak azokra, amelyeket előre létrehoztuk a CAO-t az Active Directoryban.

Használhatjuk ezt a beállítást például akkor, ha azt szeretnénk, hogy kiszolgálónk csak az ismert ügyfeleknek válaszoljon. Ismert ügyfél az a számítógép, amelynek van saját CAO-ja az Active Directory-ban. Amikor a PXE ügyfél kapcsolatba lép a RIS kiszolgálóval, elküldi a GUID-jét *(egyedi azonosító, hasonló a Media Access Control (MAC) címhez)*, amelyet össze kell hasonlítani a CAO-val, hogy egyezik-e. Biztonságos környezetben ez hasznos *(nem tud akármelyik ügyfél telepíteni)* és egyúttal terhelésmegosztásra is lehetőséget teremt a szervezet RIS kiszolgálói között. Akkor is hasznos, ha más távoli telepítő szolgáltatás is működik a szervezeten belül.

Ezek a beállítások csak a RIS kiszolgáló és az ügyfél közötti DHCP párbeszédre vonatkoznak. Ha a RIS ügyfél a RIS kiszolgáló IP címét mástól tudta meg *(például egy hivatkozó RIS kiszolgálótól)*, és az ügyfél erről a kiszolgálóról kíván rendszert tölteni, azt is megteheti.



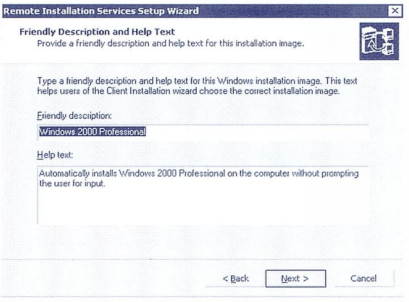
A RIS szolgáltatás beélesítése

A következő párbeszédablakban *(lásd alább)* azt a könyvtárat kell megadnunk, ahova a munkaállomások telepítéséhez szükséges állományok kerülnek majd. A könyvtár neve csak betűket és számokat tartalmazhat, szóközök vagy nem ASCII karakterek nem lehetnek benne, és a könyvtár neve legfeljebb 39 karakterből állhat.

Az I386 könyvtár ide kerül a CD-ről

A következő párbeszédablakban *(lásd alább)* a CIW leendő felhasználói számára magyarázó szöveget adhatunk meg az operációs rendszer választásához. Testreszabott Help Desk információkat is beírhatunk. Ez az információ a telepítőkészlet Templates alkönyvtárában található Rstndrd.sif állomány [Oschooser] szakaszába kerül.

Ha ezzel a képernyővel végeztünk, meg kell adnunk a telepítési médiát



Az alapértelmezett telepítőkészlet adatai

Ezek után a Risetup.exe a következőkkel fejezi be a telepítést:

- Létrehozza a könyvtárstruktúrát
- Bemásolja a Windows 2000 Professional telepítéséhez szükséges fájlokat – gyakorlatilag az egész I386 könyvtárat.
- Bemásolja az Ügyféltelepítő varázsló *(Client Installation Wizard, CIW)* képernyőit.
- Beállítja a RIS szolgáltatásokat.
- Elindítja a RIS számára szükséges szolgáltatásokat *(BINL, TFTP, és a SIS Groveler szolgáltatás)*.
- Reminist névvel megosztja a RIS könyvtárstruktúra főkönyvtárát.
- Az Active Directoryban létrehozza a megfelelő IntelliMirror management technologies Service Control Point *(SCP)* objektumot
- Létrehozza a SIS Common Store könyvtárat és azokat a fájlokat, amelyek szükségesek a SIS működéséhez ezen a köteten.

Ezen a ponton a RIS kiszolgáló telepítése befejeződött, a Windows 2000 Professional alapértelmezett beállításokkal telepíthető a munkaállomásokra. Már csak hitelesíteni kell a kiszolgálót, hogy megkezdhesse az ügyfelek kiszolgálását.

A RIS kiszolgáló hitelesítése

Mielőtt a RIS megkezdhetné az ügyfelek kiszolgálását, hitelesíteni kell őt az Active Directory-ban. Ez megakadályozza azt, hogy valaki kaló RIS kiszolgálót indítson el a hálózaton, amely értelmezhetné az ügyfelek kéréseit. Sajna ez azonban csak a RIS kiszolgáló és a RIS ügyfél közötti DHCP párbeszédre vonatkozik. Ha az ügyfél egy nem hitelesített RIS kiszolgálóra hivatkozik, akkor az is gyönyörűen telepíteni fogja az operációs rendszert. Ahhoz, hogy hitelesítsük a RIS kiszolgálót az Active Directory-ban, használjuk a Microsoft Management Console *(MMC)* DHCP snap-injét bármelyik DHCP kiszolgálón, ami ugyanebben az Active Directoryban van, és adjuk hozzá, mint meghatalmazott kiszolgálót. Miután hozzáadtuk, ellenőrizzük a RIS eseménynaplóját. Ebben a BINLSVC-től *(az a szolgáltatás, amelyet a RIS az ügyfélszámítógépekkel való kapcsolattartásra használ)* származó eseménynek kell lennie, mely szerint a kiszolgáló hitelesítetté vált.

Nem mindig kell hitelesítenünk, hogy a RIS dolgozhasson. Ha a kiszolgáló, amelyre a RIS-t telepítjük, már futtatja a Windows 2000 DHCP szolgáltatását és az aktív, akkor a meghatalmazás automatikusan megtörténik.

Vagy ha mondjuk van egy Windows NT 4.0 futó DHCP kiszolgálónk, a RIS miatt nem kell Windows 2000 DHCP-t beállítanunk, ettől még hitelesíteni tudjuk az Active Directory-ban. Lássuk hogyan!

Tehát ha a kiszolgálón nem fut a Windows 2000 DHCP szolgáltatás, telepítenünk kell az „Administrative tools"-t, hogy hozzáférhessünk a DHCP MMC snap-inhez, mellyel végrehajthatjuk a szükséges lépéseket. Az „Administrative tools" telepítéséhez egy Windows 2000 Servert vagy Windows 2000 Professionalt futtató számítógépen indítsuk el az adminpak.msi-t a Windows 2000 Server CD-ROM-ról.

Csak az „Enterprise Administrators" csoport tagjainak van joguk a RIS kiszolgálók meghatalmazására. Ha más felhasználóknak vagy csoportoknak is meg kívánjuk adni ezt a jogot, az alábbiak szerint járjunk el:

1. Indítsuk el az Active Directory Sites and Servicest
2. Az MMC konzol View menüjében kattintsunk a Show Services Node pontra
3. Bontsuk ki a Service kulcsot, majd keressük meg a NetServices kulcsot
4. Kattintsunk rá a jobb egérgombbal, és válasszuk a Properties menüpontot
5. A Security keretben engedélyezzük a Read, Write, és a Create All Child Objects jogokat a megfelelő felhasználó vagy csoport számára
6. Kattintsunk az Advanced-re
7. Az Access control Settings for NetServices párbeszédablakban kattintsunk az éppen hozzáadott felhasználóra vagy csoportra
8. A View menüben válasszuk az Edit menüpontot
9. Az Apply onto keretben kattintsunk a This object and All Child Objects-re.

Ha nem akarjuk telepíteni az „Administrative tools"-t a számítógépen, hanem csak a DHCP Manager MMC snap-int, akkor az alábbiak szerint járjunk el:

1. Bontsuk ki a Dhcpsnap.dl_ fájlt a kiszolgáló CD-jéről a Windir\System32 könyvtárba
2. Bontsuk ki a Dhcpmgmt.ms_ fájlt a kiszolgáló CD-jéről a Windir\System32 könyvtárba
3. Futtassuk a %systemroot%\system32\regsvr32 dhcpsnap.dll parancsot
4. Futtassuk a dhcpmgmt.msc állományt.

Miután végrehajtottuk ezeket a lépéseket, futtathatjuk a snap-int és meghatalmazhatjuk a kiszolgálót, amennyiben megvan a szükséges jogosultságunk. Ha egyéb eszközöket *(például az Active Directory Users and Computers MMC snap-int)* is szeretnénk használni, akkor telepítenünk kell az „Administrative Tools"-t.

Egyelőre ennyit e végtelen történetről, folyt. köv.

Dorner Csilla
dorner.csilla@hms.hu
MCSE

Windows eXPressz: újítások a Windows XP Kernelben

Az elmúlt hetekben többször bemutattuk már a Windows XP komolyabb vagy kevésbé komolyabb újításait. Most ezt a sorozatot folytatjuk, ezúttal közel a tűzhöz: lássuk, milyen teljesítménynövelő újítások vannak az operációs rendszer magjában, a Windows XP Kernelben!

Gyorsabb rendszerindítás

A felmérések szerint a felhasználók többsége azt szeretné, ha az operációs rendszer az eddiginél sokkal hamarabb lenne munkára kész, hidegindítás, hibernálás, vagy stand by mód után egyaránt. A nem titkolt cél **[4]** az volt, hogy egy átlagos felhasználó gépén a bekapcsoló gomb megnyomásától a használható állapotig legfeljebb fél perc teljen el:

Tervezett indítási idők, legfeljebb...	
Hidegindítás esetén	30 mp
Hibernálás után	20 mp
Stand By állapotból	5 mp

Ezt az időt persze erősen befolyásolja a hardverek sebessége is, ezért a Windows XP fejlesztésének ebben a szakaszában a Microsoft szorosan együttműködött az egyes hardvergyártókkal. A legelső dolog a számítógép életében a BIOS – éppen ezért ahol lehet, még a BIOS-on is gyorsítottak.

A Simple Boot Flag

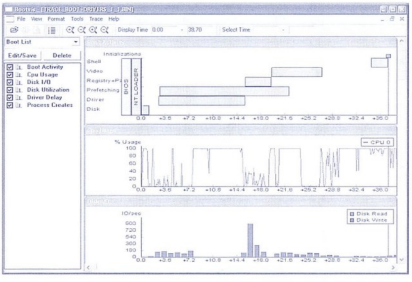
A számítógép indulásakor futó BIOS POST *(Power-On Self Test)* rutin, a memória ellenőrzése, a lemezek felpörgetése, a flopimeghajtó ellenőrzése, a gép gyártója által esetleg megjelenített logók, a videokártya esetleges logója, valamint az egyéb diagnosztikai eljárások sokszor már önmagukban meghaladják az előírt 30 másodpercet. Éppen ezért találták ki a Simple Boot Flag nevű CMOS BIOS regisztert, amit az operációs rendszer a 0x70-0x71-es porton keresztül ér el. Ezen a regiszteren keresztül „beszélgethet" a BIOS és az operációs rendszer. Az egyes bitek jelentése a következő:

A Simple Boot Flag regiszter		
0	PNPOS	Plug and Play operációs rendszer
1	BOOTING	Az előző boot sikeres volt
2	DIAG	Diagnosztika futtatása szükséges
3-6	-	Nem használt, értéke 0
7	PARITY	Ellenőrzőbit

- A PNPOS bit 1 értéke azt jelzi, hogy a számítógépen Plug & Play operációs rendszer fut. Ilyenkor a BIOS-nak semmi dolga a perifériák konfigurálásával, leszámítva persze a rendszerindításhoz elengedhetetlenül szükséges eszközöket *(például a merevlemezt)*. Ha értéke 0, a BIOS-nak kell kiosztania az eszközök részére az erőforrásokat
- A BOOTING bit 1 értéke azt jelzi, hogy az előző rendszerindítás sikeres volt, diagnosztikai tesztekre nincs szükség. A BIOS ilyenkor a POST futtatása *(és további csicsák)* helyett azonnal belefog a boot szektor betöltésébe
- A DIAG bit 1 értéke azt jelzi, hogy szükség van a diagnosztika futtatására *(ezt beállíthatja az operációs rendszer, de maga a BIOS is, ha például a BOOTING bitet 0-nak találta)*
- A PARITY bit ellenőrzőbit: ha a beállított paritásérték nem jó, a regiszter tartalma érvénytelen, teljes rendszerdiagnosztika következik

Prefecthing

A Windows XP gyorsaságának lelke az úgynevezett prefecth *(előtöltés)*: a rendszer indulásakor naplózza az összes lemezműveletet. A legközelebbi induláskor pedig az előző naplók alapján előre, párhuzamos, kötegelt lemezműveletekkel betölti a rendszerindítás során szükséges fájlokat, adatokat *(eszközmeghajtók fájljait, a registry részeit, stb.)* – még mielőtt a rendszer rájönne, hogy szüksége van rá. Amikor pedig kell, rögtön kéznél *(azaz a memóriában)* van az adat. A betöltés a „rendes" indítással párhuzamosan működik, így míg például az eszközmeghajtók az eszközök felismeréséhez szükséges elkerülhetetlen üresjáratukat töltik, a Windows szépen előkészíti a terepet a folytatáshoz. A Windows XP emellett a naplók alapján automatikusan átrendezi a rendszertöltésben résztvevő fájlokat a merevlemezen *(így lényegesen megnő az egy műveletben betölthető komponensek száma, és persze lecsökken a betöltésükhöz szükséges idő)*. Ennek köszönhetően a boot loader máris négyötször gyorsabb lesz, mint a Windows 2000-ben. A műveletekhez az XP az előző nyolc rendszerindítás adatait veszi alapul. Mindezeken túl a Windows XP optimalizálja az eszközmeghajtók betöltésének sorrendjét, hogy a felhasználó minél előbb hozzájuthasson a bejelentkezőképernyőhöz. Ha például a Windows XP nincs tartományban, és a bejelentkezéshez nincs szükség a hálózati komponensekre, azok indítása időben hátrébb csúszik *(a Winlogon tehát nem vár többé a hálózat elindítására!)*. Sok eszközmeghajtó már nem sorban, egymást megvárva, hanem egymással párhuzamosan indulhat el.



A bootvis.exe sok mindent elárul a rendszerindítási folyamatról

A Windows XP rendszerindításának folyamatát figyelemmel kísérhetjük a **[4]** címről letölthető bootvis.exe nevű eszközzel, amellyel nem csak a különféle indulási idők, de a lemezműveletek, a processzor terheltsége, a „lusta" eszközmeghajtók is megfigyelhetők, sőt, a bootvis.exe képes a bootkörnyezet – már leírt, egyébként előbb-utóbb automatikusan is végbemenő – optimalizálására is. Az optimalizálás után a tesztgépünkön futó Windows XP teljes indulási ideje 27%-kal csökkent *(az ábrán látható 36 másodperc egy átlagos indulási idő, P450 processzoron, UDMA-66-os merevlemezekkel, 512MB RAM-mal)*.

Prefetching az alkalmazásoknál

Ezúttal tényleg igaz a marketingfogás: a Windows XP tényleg gyorsabban futtatja az alkalmazásokat! A prefecthing algoritmus ugyanis nemcsak az operációs rendszer indításakor, de – az XP Server verzióinak kivételével, illetve, hacsak kifejezetten le nem tiltjuk – minden egyes alkalmazás esetén is működik. Nézzünk csak bele a Windows XP \Windows\PreFetch könyvtárába! Jónéhány .pf kiterjesztésű fájlt találunk majd benne *(és egy layout.ini-t, de erről később lesz szó)*. Az egyes prefetch fájlok tartalmazzák az alkalmazások futása során használt lemezterületek adatait *(fájlokat, .dll-eket, registry kulcsot, satöbbi)*. Ha egy alkalmazáshoz már létezik prefetch fájl, a Windows XP az alkalmazás indításakor automatikusan betölti a szükséges adatokat, hogy mire az alkalmazásnak szüksége lesz rá, kéznél legyen. (A rendszerindítás prefetch fájlja az notosbootBOODFAAD.pf; ez a fájl tartalmazza az előző nyolc rendszerindítás adatait. Ez a fájl a rendszerindítás után 1 perccel jön létre, illetve frissül. *(Ha letöröljük, a következő rendszerindításnál nem lesz prefetch – ideális eszköz a prefetch hasznosságának kipróbálására)*.

Egy fontos dolgot jegyezzünk meg: a prefecth akkor képes igazán jól működni, ha van hova betölteni az adatokat. Éppen ezért a Windows XP memóriaigényének kielégítése teljesítményokokból fontosabb, mint az elődöké volt. A minimális 128 megabájt a mai memóriaáraknál már nem okozhat gondot.

A lemezterület automatikus optimalizálása

A prefetch jó és gyors működésének fontos feltétele, hogy a betöltendő adatok a lemezen is megfelelő sorrendben legyenek. Mit sem ér a prefetch algoritmus, ha a gyors betöltés során ide-oda kell rángatni a vincseszter fejét! A Windows XP ezért a számítógép pihenőidejében *(idle állapot esetén)* folyamatosan optimalizálja a merevlemezek tartalmát *(!)*. A prefetch szolgáltatás a prefetch fájlokból kiolvassa a boot és az alkalmazások indításának adatait, és ezek alapján felépíti a layout.inf fájlt. Az automatikus defragmentálás ezután ez alapján működik. A layout.inf fájl 32 alkalmazás elindítása után jön létre először, és a rendszerindítások során automatikusan frissül.

Hibernálás

A hibernálás is tartalmaz újításokat: a Windows XP ugyan továbbra is a fizikai memóriával megegyező méretű fájlt foglal le a bootpartíció gyökerében, de ma már szó sincs arról, hogy minden egyes bitet a lemezre mentene. A hibernálás előtt az XP felszabadítja a zero, free, standby listában található memórialapokat, ezek nem kerülnek ki a lemezre. A hibernálás során *(az adatok lemezre írásával egyébként egyidejűleg)* az XP még tömöríti is az adatokat. A lemezre írásnál érdemes tudni, hogy az XP már DMA-t használ a hibernáláshoz.

A hibernált gép felélesztése is kicsit másképp működik: a bootszektorba ugyanis nem fér bele a lemez eléréséhez és kitömörítéséhez szükséges kód, ezért az XP csak a kitömörítéssel foglalkozik, a beolvasást a BIOS funkciókra *(!)* bízza. Ez természetesen lassabb, mintha a saját rutinját használná, de a mentéskor alkalmazott adatszűrésnek és tömörítésnek köszönhetően ez a lassúság nem számottevő. *(A tesztgépünkben található 384MB RAM-ról hibernált Windows XP 6 (!) másodperc alatt éledt fel a teljes fagyhalálból. Ez már megközelíti a stand by értéket – az is igaz, hogy a 384 MB memóriának csak nagyon kis hányadát használtuk valójában)*.

A Registry

A Registry statikus konfigurációs adatbázis, amely a számítógép, illetve a szoftverek különféle beállításait tartalmazza – legalábbis szeretnénk ezt hinni. A legnagyobb tévedés a statikusság: valójában a programok gyakran használják a Registry-t *(nem feltétlenül szerencsésen)* dinamikus, gyakran változó adatok tárolására is, pedig eredetileg nem arra tervezték. A folyamatos használat során a Registry töredezetté válik, a benne tárolt adatok elérése lassul. A töredezettség pedig addig nem múlik el, míg azt egy arra alkalmas eszközzel *(például a korábban általunk is bemutatott SysInternals PageDefrag-gal)* meg nem gyógyítjuk.

Amikor egy alkalmazás új kulcso*(ka)*t hozott létre, a régebbi Windows-ok megkeresték az első rendelkezésre álló helyet a Registry-ben, és elkezdték a kulcsok létrehozását. Ha a hely nem volt elég az összes kulcshoz, az „összetartozó" kulcsok a Registry elszórt pontjaira kerültek. Amikor az alkalmazás használni kezdte volna ugyanezeket, az olvasás során újabb és újabb lapokat kellett betölteni, és ez természetesen időbe került.

Windows 2000
1. alkalmazás, kulcs 1.
4. alkalmazás, kulcs 1.
1. alkalmazás, kulcs 2.
1. alkalmazás, kulcs 3.
2. alkalmazás, kulcs 1.
1. alkalmazás, kulcs 4.
3. alkalmazás, kulcs 2.
2. alkalmazás, kulcs 2.
4. alkalmazás, kulcs 2.
3. alkalmazás, kulcs 1.
2. alkalmazás, kulcs 3.
3. alkalmazás, kulcs 3.
5. alkalmazás, kulcs 1.
6. alkalmazás, kulcs 1.
3. alkalmazás, kulcs 4.

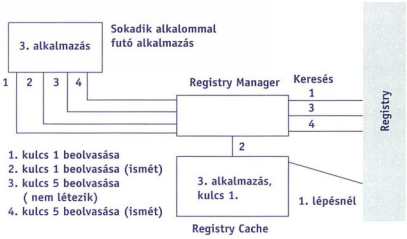
Windows XP
1. alkalmazás, kulcs 1.
1. alkalmazás, kulcs 2.
1. alkalmazás, kulcs 3.
1. alkalmazás, kulcs 4.
5. alkalmazás, kulcs 1.
2. alkalmazás, kulcs 1.
2. alkalmazás, kulcs 2.
2. alkalmazás, kulcs 3.
4. alkalmazás, kulcs 1.
4. alkalmazás, kulcs 2.
6. alkalmazás, kulcs 1.
3. alkalmazás, kulcs 1.
3. alkalmazás, kulcs 2.
3. alkalmazás, kulcs 3.
3. alkalmazás, kulcs 4.

A Windows XP már a kulcsok létrehozásakor ügyel a töredezettségre

Amikor a Windows XP létrehozza a kulcsokat, megpróbálja megkeresni azt a helyet, ahol az összes kulcs *(illetve az egyes kulcsokhoz tartozó további adatok, például jogosultságlista)* egymáshoz közel tárolható, így az adatok visszaolvasása is sokkal gyorsabb lesz majd.

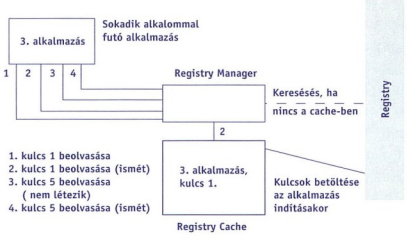
A Windows korábbi változataiban a Registry másolata a kernel memóriájában tanyázott. Ez *(memóriatervezési okokból)* legfeljebb körülbelül 160MB méretűre duzzadhat, ezért előfordult, hogy egy-egy bonyolultabb, nagyobb Registry a kernel rendelkezésére álló memória nagy részét felemésztette. A Windows XP-ben a Registry nem a kernel memóriába töltődik be, így többé nem falja fel a memóriát az operációs rendszer elől. Ennek viszont ára van: ha a registry nem a kernel memóriában van, azt lassabban lehet elérni. Az XP újraírt *(és újratervezett)* Registry-kezelő rutinjai ezért számos trükköt alkalmaznak.

Kulcskeresés a Windows 2000-ben



Registry-műveletek a Windows 2000-ben...

Kulcskeresés a Windows XP-ben



... és a Windows XP-ben

E trükkök egyike az alkalmazáshoz tartozó kulcsok gyorsítótárazása. A registry-gyorsítótárat már a korábbi Windows verziók is alkalmazták, de csak a létező kulcsok esetén. Az alkalmazások viszont sokszor csak egy-egy kulcs létezését ellenőrzik. Amikor az alkalmazás olyan kulcsra hivatkozik, ami nincs a gyorsítótárban, a Windows keresni kezdi azt a registryben. Mivel a Windows 2000 a nemlétező kulcsokat nem tárolta a gyorsítótárban, minden ilyen kulcsra irányuló kérés a Registry végigpásztázását eredményezte. A Windows XP tárolja a kulcs nemlétezésére vonatkozó információkat is, sőt, a prefetch műveletek során az alkalmazáshoz tartozó, általa használt registry kulcsok már az alkalmazás indulásakor betöltődnek a gyorsítótárba, így a Windows XP már az első kérést is onnan szolgálja ki.

Még egy kellemes újdonság a témával kapcsolatban: megszűnt végre a Registry Editor *(regedit)* zavaró skizofréniája. A Windows XP-ben található eszköz az eredeti regedit.exe-hez hasonlít *(azaz nem többablakos)*, viszont mindent tud: kezeli a kulcsok jogosultságait, felismeri a többsoros szöveges értékeket. Kompatibilitási okokból a regedt32.exe is megmaradt, de nem csinál mást, mint hogy elindítja a regedit.exe-t.

Hibakeresés

A Windows XP új hibakereső eszközöket *(debuggereket)* tartalmaz. Az eszközök között megtaláljuk a WinDbg, KD, CDB újraírt verzióit, sőt, ezek 64 bites változatát is. Az új debuggerek használhatók a Windows NT 4.0 és a Windows 2000 rendszerekkel is, bár néhány funkció értelemszerűen csak a Windows XP-ben használható. Az eszközökről széleskörű információ a **[2]** címen található.

Néhány újdonság:

- Cross-Session Debugging: a régebbi debuggerek a Windows Session Manageren *(csrss.exe)* keresztül működtek. Ez gondot okozott a Terminal Services használatakor, hiszen akkor minden felhasználó saját Win32 alrendszerét futtatja, saját csrss példánnyal. Így természetesen a terminálok közötti hibakeresés nem működött – a Windows XP-ben már igen.
- Quit and Detach: Kilépés és lecsatlakozás. A Windows XP Kernel Debugger új parancsa *(qd)* lehetővé teszi, hogy a debuggerből való kilépéskor a felügyelt alkalmazás ne álljon le. Ha a qd paranccsal hagyjuk el a debuggert, az alkalmazás tovább futhat.
- IEEE 1384: Az új debugger képes a soros port helyett FireWire *(IEEE 1384)* csatlakozón keresztül működni *(a Windows Me esetén ez már most is működik)*. Egy IEEE 1384 buszra a debugger gép mellé akár 62 másik számítógép csatlakoztatható.

```
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS=
  "Microsoft Windows XP Professional" /debug
  /debugport=1384 /channel=1
```

Ha a boot.ini /debugport kapcsolójának a 1384 értéket adjuk, a debugger az IEEE 1384-et használja

- Debug-child: A debugger által „betöltött" processz beállítható jellemzője, hogy csak őt, vagy az általa indított processzeket is szeretnénk-e debuggolni. A Windows XP előtt ez a beállítás a processz indításakor eldőlt, később nem lehetett változtatni – ma már lehet.
- Gyorsabb soros port-használat: a lassú, soros porton keresztüli hibakeresés esetén jól jön, hogy a Windows XP debugger hatékonyabban használja a rendelkezésre álló kapcsolatot
- Eszközmeghajtó feltöltés a Kernel Debuggeren keresztül: amikor a Windows XP egy eszközmeghajtót készül indítani, lekérdezi a debuggert, hogy rendelkezik-e újabb, jobb eszközmeghajtóval. Ezzel megspórolható az eszközmeghajtó-telepítéshez szükséges újraindítás.

Harc a memóriafolyás ellen

A feladat elkapni a memóriát lefoglaló, de azt fel nem szabadító alkalmazások grabancát. A Windows XP két fronton veszi fel a harcot:

- Memóriafolyás felismerése a processz befejezésekor: ha a

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\
  CurrentVersion\Image File Execution Options\
  <imagename>\ShutdownFlags
```

DWORD értéket 3-ra állítjuk *(az <imagename> helyére írjuk a felügyelendő alkalmazás nevét, pl. notepad.exe)*, a Windows XP a program kilépésekor a körmére néz, és ha felszabadítatlan memóriaterületet talál, riasztja a felhasználót:

A Windows XP elkapja a memóriapocsékoló alkalmazások grabancát

Szerencsére *(?)* azért nem mindig olyan súlyos a helyzet, mint amit a fenti ábrán a **[3]** címről letölthető példaprogrammal előállítottunk....

- Heap számlálók a Performance Monitorban: ha a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
  Services\PerfProc\Performance\
  DisplayHeapPerfObject
```

DWORD értéket 1-re állítjuk, a Performance Monitor-ban több mint 20 új számláló jelenik meg.

A prefetch kikapcsolható

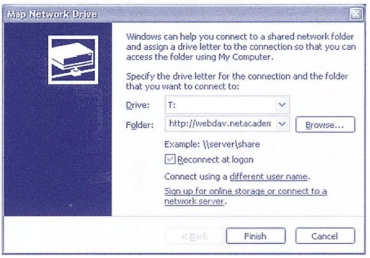
Érdemes tudni, hogy a prefetching műveletet az új Logical Prefetcher rendszerszolgáltatás, míg a futás közbeni naplók feldolgozását, és prefetch-fájlba írását a Task Scheduler *(!)* végzi. Amiért érdekes, *(és amiért pont a memóriamenedzsment címszó alatt említjük meg)*, az a prefetch letilthatóságának helye a Registry-ben:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
  Control\Session Manager\Memory Management\
  PrefetchParameters\
```

A fenti kulcs alatt található EnablePrefetcher DWORD érték határozza meg, hogy a prefecthing működjön-e. Az érték 0. bitje engedélyezi az alkalmazás, az 1. bit pedig a rendszertöltés gyorsítását *(ha tehát a fenti értéknek 3-at adunk meg, a teljeskörű prefecthet engedélyezzük)*. A Windows XP munkaállomásokon *(Professional, Home Edition)* ez az érték 3 *(azaz teljeskörű)*, a Windows .NET Server-eken pedig 2 *(azaz csak boot)*.

A WebDAV File System Driver

A File System Driver-ek feladata, hogy a Windows fájlrendszerébe integráljanak eredetileg nem ott található tárolókat. Ilyen FSD például a Windows hálózati meghajtókat kezelő komponense *(a „Map Network Drive")* is. A Windows XP-ben egy új File System Driver-rel találkozhatunk, amely a WebDAV File System Driver névre hallgat.



Munkában a WebDAV File System Driver

Az új FSD lehetővé teszi, hogy távoli kiszolgálók WebDAV mappáit beépítsük a Windows fájlrendszerébe, ugyanúgy, mintha hagyományos Windows megosztott mappákhoz csatlakoznánk. *(A WebDAV protokollról szeptemberi számunkban olvashatnak részletesen)*. A WebDAV FSD gyermegbetegsége, hogy nem képes https:// mappákhoz csatlakozni – ezeket továbbra is csak *(például a My Network Places-ből elérhető)* webmappaként használhatjuk.

Még néhány szó a defragmentálásról

A beépített töredezettségmentesítő előnyeit a Windows 2000 óta élvezhetjük. Az XP defragmenter programozói felülete *(ezzel együtt majd nyilván a ráépülő alkalmazások többsége is)* tartalmaz néhány újdonságot:

- Alapvető változás, hogy a defragmentálás ezentúl nem a System Cache-en keresztül történik *(egy átlagos felhasználónak ez annyit jelent, hogy a töredezettségmentesítés során ezentúl a fájlok nem lesznek olvasásra megnyitva)*
- Az MFT *(Master File Table)* is defragmentálható
- Defragmentálható lett az NTFS fájlok attribútuma, az indexek, valamint a bitmap fájl is.

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://msdn.microsoft.com/library/en-us/appendix/enhancements5_9b74.asp
[2] http://www.microsoft.com/ddk/debugging/
[3] http://technet.netacademia.net/download/leak
[4] http://www.microsoft.com/hwdev/fastboot/

Golyóálló IIS

Az elmúlt hetekben *(főleg az új internetes férgeknek – Code Red, Nimda és társai – köszönhetően)* megszaporodtak a Windows webkiszolgálók elleni támadások. Ma már nem elég az hinni, hogy az Internet valamelyik sarkában el tudunk bújni, a férgek szisztematikusan támadnak: végigpróbálnak minden címet, idejükből kitelik. Egyetlen esélyünk a védekezés. Cikkünkben bemutatunk két új eszközt az IIS biztonságának növelésére, valamint másik két eszközt, melynek segítségével a gyorsjavítások ellenőrzését és telepítését könnyíthetjük meg *(ezek az utóbbi eszközök nemcsak a webkiszolgálóként működő gépeken, de minden Windows NT-n és Windows 2000-n használhatók)*.

IISLockD – IIS Lockdown Tool

Az eszköz neve máris félreértésre adhat okot, ugyanis az IIS védelméről szól, holott az eszköz csak a webkiszolgáló biztonsági beállításaira használható. Ne feledjük el, hogy az IIS-nek a webkiszolgálón kívül része az FTP, az SMTP és az NNTP kiszolgáló is *(különösen tartsuk ezt a fejünkben, amikor a gyorsjavításokat telepítjük)*. Az IISLockD letöltése **[1]** és telepítése után a könyvtárban három fájlt találunk: magát az eszközt, a hozzá tartozó help fájlt, valamint egy 404.dll-t. Ez utóbbi különös fontosságú lesz a későbbiek során.

Legyünk óvatosak! Az eszköz nagyon szigorú, felkészületlenül használva könnyen üzemen kívül helyezhetjük az IIS-ünket. Szerencsére ha kell, visszaállíthatjuk az eredeti állapotot az Undo parancs használatával. De lássuk, mit is tud az IISLockD! Az IISLockD varázsló Windows NT 4.0 *(IIS 4.0)* és Windows 2000 *(IIS 5.0)* esetén is használható. Indulásakor két konfigurációt ajánl fel:

- Express Lockdown – automatikus konfigurálás, statikus webkiszolgálók részére. Vigyázat! Minden dinamikus szolgáltatást letilt *(az ASP-t is)*!
- Advanced Lockdown – testreszabható beállítások

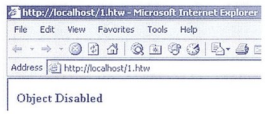
A testreszabható beállítások között pedig az alábbiakat találjuk:

- Az ASP oldalak letiltása – tiltsuk le, ha nincs szükség dinamikusan generált tartalomra *(.asp, .asa, .cer, .cdx)*
- Az Index Server webes interfészének letiltása – ehelyett már régóta a programozható scriptfelületét használjuk *(.idq, .htw, .ida)*. Ez a komponens sok IIS hiba forrása volt
- A Server Side Includes letiltása – Tiltsuk le, hacsak nem használjuk *(.shtml, .shtm, .stm)*
- Az Internet Data Connector letiltása – Ezer éve nem használt webes adatbázisinterfész, emellett sok-sok biztonsági rés melegágya (képzavar – a szerk.) *(.idc)*
- Az Internetes nyomtatás letiltása – Tiltsuk le, hacsak nem használjuk *(.printer)*
- .htr scriptek letiltása – Lásd mint az előző *(.htr)*

Akinek ezek a lépések ismerősek, nem téved: a legtöbb biztonsági ajánlásban szerepel ezek eltávolítása. Ha viszont egyszerűen töröljük a hozzárendelést, előfordulhat, hogy az adott komponens újratelepítése, vagy egy javítócsomag esetén a hozzárendelés újra megjelenik. Az IISLockD azonban ennél sokkal trükkösebb:

A letiltott kiterjesztések valójában nem tiltódnak le...

Az IISLockD futása során a könyvtárában található 404.dll fájlt telepíti a \System32\inetsrv könyvtárba, majd minden „tiltott" kérést ráirányít. Ezután minden ilyen kérés a következő választ kapja *(egyébként szabványos, de rövid HTTP hibaüzenet formájában)*:



... hanem diszkrét hibát adnak vissza

Ennek a megoldásnak két előnye van: egyrészt, a rövid *(kevesebb, mint kétszáz bájtos)* hibaüzenet a minket ért támadás során nem terheli annyira a vonalunkat, mint az alapértelmezett, több *(kb. négy)* kilobájtnyi színes-szagos üzenet; a másik, hogy – miután a scripthozzárendeléseket nem töröltük – azok nem fognak a későbbiekben „véletlenül" újra létrejönni, és lyukat ütni a rendszerbe.

Az IISLockD varázsló további beállításai:

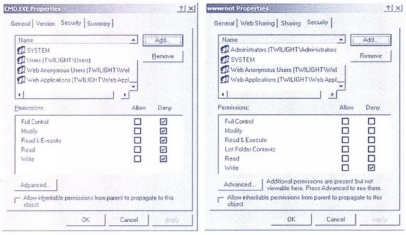
- A példaként telepített webfájlok törlése *(ez az IISSamples virtuális könyvtár törlését jelenti, az eredeti fájlok a lemezen (az \inetpub\iissamples könyvtárban) megmaradnak, de nem lesznek elérhetők)* – ebben a könyvtárban néhány hibás példakódnak köszönhetően jópár veszélyes lyuk lakik.
- A Scripts virtuális könyvtár törlése – alapértelmezett, futtatási joggal felruházott könyvtár. Ha a támadó valamilyen más módon e könyvtár alá bármit be tud tölteni, azt „kívülről" már gyerekjáték futtatni. E könyvtár tartalma is megmarad, csak a virtuális hozzárendelés szűnik meg. Ha a gépen Proxy Server 2.0 fut, még a virtuális könyvtárat se töröljük, mert a Proxy Server azután nem fog működni.
- Az MSADC virtuális könyvtár törlése – az MSADC adatbáziskomponens telepítője által létrehozott virtuális

mappa, sokszor hibás, biztonsági rést okozó példakódokkal. Hacsak nincs rá kifejezetten szükségünk, töröljük ki.

- A WebDAV letiltása – a WebDAV HTTP protokollon keresztül megvalósított, írásra és olvasásra egyaránt használható protokoll, melyről korábbi számunkban már írtunk. Ha nem használjuk, itt letilthatjuk, de vigyázzunk, mert a WebDAV globális letiltásának egyetlen módja, ha a httpext.dll-hez megszüntetjük a System felhasználó hozzáférési jogait. Az IISLockD is ezt teszi. A dolog azért veszélyes, mert hozzáférési jog híján a javítócsomagok sem lesznek képesek frissíteni ezt a komponenst *(tehát javítócsomag, hotfix telepítése előtt a hozzáférési jogot adjuk vissza)*.

Az IISLockD két új csoportot hoz létre: ezek a Web Anonymous Users *(tagja az IUSR_<gépnév> felhasználó)* és a Web Application *(tagja az IWAM_<gépnév>)*. Az NTFS jogosultságokat ezután ezeknek a csoportoknak osztja ki, és ezentúl tegyünk így mi is.

- Az IIS anonymous felhasználó rendszereszközökhöz való hozzáféréseinek letiltása – ha kiválasztjuk, az IISLockD a fontos rendszerkomponensekhez a fenti csoportoknak explicit tilt minden hozzáférést *(Full Control Deny jogosultságot állít be)*.
- Az IIS anonymous felhasználó írásjogának letiltása a webkiszolgáló tartalmához – ha ezt választjuk az IISLockD explicit írás tiltást állít be többek között az \InetPub\wwwroot könyvtáron.



Explicit tiltó NTFS jogok az IISLockD futása után

Joggal merülhet fel a kérdés, hogy mi történik akkor, amikor az IISLockD-t Windows NT 4.0-n használjuk (ott ugyanis a jogosultságlistában nincs tiltó típusú jog)? Nos, az eredeti eszközöket használva valóban nincs ilyen, és előfordulhat, hogy az átállított fájlok jogosultságlistájához az IISLockD futtatása során nem fogunk hozzáférni. Az igazság azonban az, hogy bár a felhasználói felületre nincs kivezetve, a Windows NT 4.0 a Service Pack 4 óta a Windows 2000-kompatibilis NTFS fájlrendszert használja, ahol pedig definiálhatók ilyen jogosultságok. Ezután már csak egy valami hiányzik: az eszköz, amivel ezeket a beállításokat elérhetjük. Kaparjuk csak elő a Windows NT 4.0 Service Pack 4 CD-t, és – ha még nem tettük meg – telepítsük az \MSSCE könyvtárban található Microsoft Security Configuration Editort! Ezután a fájlok tulajdonságlapján a fentihez hasonló biztonsági oldalak jelennek majd meg, és máris kezelhetjük a Deny jogokat!

A műveletekről az IISLockD naplófájlt készít, amiből kiolvashatjuk, mit tett a konfigurálás során *(naplófájl: \System32\inetsrv\oblt-log.log; riport: oblt-rep.log)*. A futtatás előtt az IISLockD automatikusan elmenti az IIS konfigurációját *(a MetaBase-t is)*, így az IISLockD visszaállítás teljeskörű lehet, ha mégis szükség lenne rá.

Fontos! A drasztikus beállítások és NTFS jogosultságok miatt mindenképpen teszteljük a beállításokat mielőtt az éles kiszolgálót módosítanánk!

Az IISLockD és az Exchange Outlook Web Access

Az Outlook Web Access képes komolyan megsínyleni az IISLockD futtatását **[2]**, legyen szó akár az Exchange Server 5.5-ről, akár az Exchange 2000-ről. Azért a helyzet nem reménytelen. Exchange Server 5.5 esetén a következőkre figyeljünk:

- Ne tiltsuk le az .asp hozzárendelést
- Ha a .htr hozzárendelést letiltjuk, csak az Outlook Web Access jelszómódosító komponense nem fog működni

Exchange 2000 esetén:

- Ha az .asp hozzárendelést letiltjuk, a Outlook Web Access multimédia gombja nem fog működni
- Ha a .htr hozzárendelést letiltjuk, csak az Outlook Web Access jelszómódosító komponense nem fog működni
- Ne tiltsuk le a WebDAV szolgáltatást
- Ne engedélyezzük az IIS anonymous felhasználók explicit írásjogának letiltását a webkiszolgáló tartalmához *(az Exchange virtuális mappái miatt)*

Ehelyett kézzel, a fájlrendszerben letilthatjuk az összes többi mappához való hozzáférést, csak az Exchange virtuális mappáit kíméljük meg.

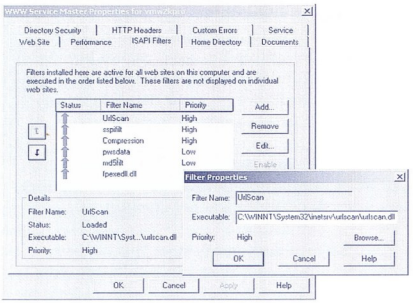
URLScan – a webkiszolgálóhoz érkező URL-ek ellenőrzése

Az URLScan **[3]** egy régóta várt eszköz. Feladata, hogy szűrje a webkiszolgálóhoz beérkező kéréseket. Menet közben *(még mielőtt az IIS nekikezdene a kiszolgáláshoz)* normalizálja a kéréseket, kiszűri belőle a gyanús vagy éppenséggel közismerten veszélyes elemeket. *(Ún. ISAPI filterként működik, ezért képes a beérkező kérések, sőt, akár a kimenő válaszok tartalmának módosítására is.)* Az URLScan az alábbi feltételeket ellenőrizheti, illetve szűrheti:

- A HTTP kérés parancsát *(GET, POST, stb.)*
- A kért fájl kiterjesztését
- Az URL kódolás típusát
- A nem-ASCII karaktereket a kérésben
- Adott karaktersorozatot a kérésben
- Adott fejléceket a HTTP kérésben

Ha az URLScan automatikus telepítését választjuk, az a teljes webkiszolgálóra érvényes lesz, kiszolgálószintű ISAPI filterként jelenik majd meg. *(Lásd: Internet Services Manager / <gépnév> / Properties / Master Properties: WWW Service / Edit... / ISAPI filters fül)*

Ha ezt szeretnénk elkerülni, a telepítőt az urlscan.exe –c paranccsal indítsuk, ilyenkor az csak kitömöríti a fájlokat, és magunknak kell elvégeznünk az ISAPI filterek telepítését. Ennek módját a csomagban található urlscan.txt-ben, illetve a **[4]** címen találjuk.



Az URLScan kiszolgálószintű komponensként települ

Az alapértelmezett telepítés szigorú beállításai csak a statikus HTML és az ASP oldalak működését engedélyezik, minden más *(WebDAV, FrontPage Server Extension, Index Service, CGI programok, SSI, Internet nyomtatás, stb.)* tiltott.

Az URLScan.ini

Az URLScan működését a \winnt\system32\inetsrv\urlscan könyvtárban található URLScan.ini fájl segítségével befolyásolhatjuk. *(Ugyanebben a könyvtárban készül egyébként az urlscan.log fájl is, ami az aktuálisan érvényes beállításokat, valamint a visszautasított kéréseket tartalmazó napló.)* Ahhoz, hogy a változtatások érvényre jussanak, újra kell majd indítanunk a webszolgáltatást. Az URLScan.ini tartalma több fő részre bomlik. Az [Options] rész tartalmazza a globális beállításokat, a fontosabbak:

- UseAllowVerbs – HTTP kérések szűrésének módja. Ha értéke 1, akkor csak az [AllowVerbs] szakaszban felsorolt HTTP parancsokat fogadja el; ha értéke 0, akkor viszont csak a [DenyVerbs] szakaszban felsorolt HTTP parancsokat utasítja vissza.
- UseAllowExtensions – Kiterjesztések szűrésének módja. Ha értéke 1, akkor csak az [AllowExtensions] szakaszban felsorolt kiterjesztéseket fogadja el, ha pedig 0, akkor csak a [DenyExtensions] szakaszban felsorolt kiterjesztéseket utasítja vissza.
- AllowDotInPath – ha értéke 1, akkor engedélyezzük, hogy az URL-ben a kiterjesztésen kívül is legyen pont. Ha 0 *(és ez az alapértelmezés)*, akkor vigyázzunk arra, hogy ne legyenek olyan könyvtárak, illetve fájlnevek a kiszolgálón, amelyek nevükben pontot tartalmaznak *(pl. /pelda.dir/benne.pont.txt)*.
- EnableLogging – ha értéke 0, nem készül urlscan.log fájl
- RemoveServerHeader – ha értéke 1, az IIS által küldött válaszban nem szerepel a webkiszolgáló verziójára utaló sor Hogy ez mire jó? Egy távoli webkiszolgáló verzióját a legegyszerűbben úgy deríthetjük ki, ha egy üres kérdést küldünk neki, majd megnézzük, hogy a válaszban a Server: HTTP fejléc értéke mi. Egy hagyományos IIS5 esetén az alábbi:

```
Server: Microsoft-IIS/5.0
```

A fejléc kikapcsolása mellett egy másik lehetőségünk is van: ha az [Options] szakaszban az AlternateServerName szakasznak értéket adunk, mi magunk határozhatjuk meg, hogy a webkiszolgálónk minek hazudja magát. Az alább látható beállítás hatására például az IIS-ünk egyszerre indiánnak érzi majd magát *(ne felejtsük el a változtatás után a webszolgáltatást újraindítani)*:

```
AlternateServerName=Apache/1.3.20 (Unix)
↳ mod_ssl/2.8.4 OpenSSL/0.9.6 mod_perl/1.26
```

A globális opciók után a felsorolások következnek. Az [AllowVerbs] / [DenyVerbs], illetve [AllowExtensions] / [DenyExtensions] párosokról már volt szó, ezen kívül a következők találhatók az urlscan.ini-ben:

- [DenyHeaders] – Ebben a szakaszban sorolhatjuk fel azon HTTP fejlécek nevét, amelyeket szeretnénk a kérésekből kiszűrni.
- [DenyURLSequences] – Itt pedig olyan karaktersorozatok listája található, amelyeket nem szeretnénk az URL-ekben viszontlátni.

FrontPage Server Extension és WebDAV

Ha szeretnénk, hogy a FrontPage Server Extensions az URLScan telepítése után is működjön, az alábbiakat kell tennünk:

- Az AllowLateScanning mező értékét állítsuk 1-re.
- Az [AllowVerbs] szakaszban engedélyezzük az OPTIONS parancsot.
- A webszolgáltatás újraindítása után a korábban már említett helyen ellenőrizzük, hogy az fpexedll.dll magasabban áll-e, mint az URLscan. Ha nem, cseréljük meg őket, majd indítsuk megint újra a webszolgáltatást.

Ha pedig a WebDAV szolgáltatást szeretnénk engedélyezni:

- Vegyük fel a WebDAV parancsokat az [AllowVerbs] szakaszba *(OPTIONS, PROPFIND, PROPPATCH, MKCOL, DELETE, PUT, MOVE, COPY, LOCK, UNLOCK)*.
- A [DenyHeaders] szakaszból töröljük a Translate:, If:, Lock-Token: sorokat.
- A [DenyExtensions] szakaszból töröljük az olyan kiterjesztéseket, amelyek fel- és letöltését engedélyezni szeretnénk.
- Szükség szerint az AllowDotInPath opciónak adjunk 1 értéket.

Az URLScan és az Exchange Outlook Web Access

Az Outlook Web Access természetesen az URLScan beállításaira is kényes **[2]**. Exchange 5.5 esetén egyszerűbb a helyzet:

- Állítsuk az UseAllowExtensions értékét 0-ra
- Ha a webes jelszóváltoztatást használni szeretnénk, a [DenyExtensions] szakaszból távolítsuk el a .htr sort

Exchange 2000 esetén kicsit több a teendő:

- AllowDotInPath=1
- Az [AllowVerbs] szakaszba az alábbiakat vegyük fel: GET, POST, SEARCH, POLL, PROPFIND, BMOVE, BCOPY, SUBSCRIBE, COPY, MOVE, PROPPATCH, BPROPPATCH, DELETE, BDELETE, MKCOL, OPTIONS
- A [DenyHeaders] szakaszból töröljük a Translate: sort
- Ha a cég belső DNS neve tartalmazza a .com karaktersorozatot, a [DenyExtensions] szakaszból töröljük a .com sort

Ha a naplózás engedélyezve van, az URLScan minden visszautasított kérést *(és annak okát)* lejegyzi az

URLScan.log fájlba. Ha nem vagyunk biztosak abban, hogy mit kellene engedélyeznünk, próbálkozzunk a webkiszolgáló használatával, majd a naplóból nézzük ki, mit és milyen okból utasított vissza az URLScan.

HfNetChk – Network Security Hotfix Checker

Az eszköz **[5]** csak angol nyelvű operációs rendszerre telepíthető, és igényli az Internet Explorer legalább 5.01-es verzióját. A kis parancssori eszköz feladata, hogy ellenőrizze a számítógépekre telepített biztonsági javításokat. Jelenleg a következő szoftverek *(hiányzó)* hotfixeinek felismerésére képes:

- Windows NT 4.0 minden verziója, Windows 2000 Professional, Server, Advanced Server
- Windows XP Home, Professional
- Internet Information Server *(IIS)* 4.0
- Internet Information Services *(IIS)* 5.0
- SQL Server 7.0, SQL Server 2000, MSDE
- Internet Explorer 5.01 vagy újabb verziója

A HfNetChk az ellenőrzéshez egy XML fájlt használ, amit maga a szoftver telepítőkészlete nem tartalmaz. Hacsak erre kapcsolóval külön nem kérjük meg, a HfNetChk minden indítása után megpróbálja letölteni a legfrissebb változat digitálisan aláírt telepítőkészletét a Microsofttól. Erről a tényről a Windows kis ablakban figyelmeztet is *(a frissen letöltött „MSSecure XML File" elfogadásához kattintsunk a Yes gombra)*. A jelenlegi *(2001 október)* adatbázis kitömörítve 858 kilobájt. Ha ez megvan, a kis eszköz a registry-ben, illetve az érintett fájlok verzióinak és ellenőrzőösszegeinek ellenőrzése után korrekt kis riportot készít az elmaradásokról.

A HfNetChk szintaxisa és kapcsolói *(lásd még hfnetchk /? illetve* **[6]***)*:

```
hfnetchk.exe [-h hostname] [-i ipaddress]
[-d domainname] [-n] [-r range] [-a action]
[-t threads] [-o output] [-x datasource] [-z] [-v]
```

- -h: az ellenőrizendő számítógép NetBIOS neve; vesszővel elválasztva többet is megadhatunk
- -i: az ellenőrizendő számítógép IP címe; vesszővel elválasztva többet is megadhatunk
- -r: az ellenőrizendő IP cím tartomány *(pl. 192.168.0.1-192.168.255.255)*
- -d: az adott tartomány minden gépét ellenőrzi
- -n: az alhálózat minden gépét ellenőrzi
- -a: módosítókapcsolók, a listázandó hotfixek beállítása: „i" *(telepített)*; „m" *(hiányzó)*; „n" *(szükséges)*; „b" *(telepített és hiányzó)*. Az alapértelmezés az „n"
- -t: az ellenőrzéshez igénybe vett végrehajtási szálak száma *(1..128, alapértelmezés 64)*
- -o: a kívánt kimeneti formátum: „tab": tabulált, „wrap" szöveges *(az alapértelmezés az utóbbi)*
- -x: a XML adatforrás helye. Lehet maga az XML fájl, lehet az ezt tartalmazó CAB fájl, vagy URL is. Ha nem adjuk meg, megpróbálja a legfrissebb verziót letölteni
- -z: a registry-ellenőrzés letiltása
- -v: részletes jelentés bekapcsolása szöveges üzemmódban

Az eszköz helyi futtatásához nem szükséges, bár javasolt, a távoli számítógépek ellenőrzéséhez érthető okokból viszont elengedhetetlen a rendszergazdai jogosultság.

QChain - Sok legyet egy csapásra

Miután a HfNetChk közölte velünk az elmaradásunkat, el kell gondolkodnunk azon, hogyan telepítsük a javításokat. A hotfixek telepítésének legutálatosabb és legveszélyesebb része az, hogy *(elvileg)* minden egyes hotfix telepítése után újra kell*(ene)* indítani a számítógépet. Aki próbált már egy frissen telepített Windows NT 4.0-t naprakészre hozni, az tudja, hogy a közel száz gyorsjavítás telepítése és az ugyanennyi újraindítás szinte lehetetlen.

Miért nem lehet újraindítás nélkül telepíteni a hotfixeket? Azért, mert a hotfixek által cserélt fájlok sokszor menet közben még használatban vannak, ezért azok igazából nem a hotfix telepítésekor, hanem a gép újraindításakor, egy automatizmusnak köszönhetően cserélődnek le. Ha két vagy több hotfix ugyanazt a fájlt módosítja, és a hotfixeket újraindítások nélkül, nem a megfelelő sorrendben telepítjük, a végeredmény a teljes sikertől a teljes kudarcig terjedő skálán bárhol lehet.

Ezt az áldatlan állapotot hivatott megoldani a QChain.exe nevű kis programocska **[7]**, amely a szó szoros értelmében rendet tesz a számítógépen *(Windows NT 4.0-tól egészen a Windows XP friss verzióiig használható)*. Ha letöltöttük **[8]**, más dolgunk sincs, mint tetszőleges sorrendben (!) telepíteni a gyorsjavításokat, majd – fontos – még mielőtt a rendszer újraindítanánk, futtassuk a QChain.exe-t. A többit csak bízzuk rá.

A hotfixek jellemzője, hogy a telepítés után feldobnak egy vissza nem vonható ablakot, amelyben a rendszer újraindítását ajánlják fel. Ezt a hotfixek –z kapcsolójával kerülhetjük el *(megadása esetén a hotfix nem akarja majd újraindítani a gépet)*. Ha akarjuk, a sok-sok hotfixet egy batchfájlba tehetjük, a végén a qchain.exe-vel. Száz hotfix telepítése egyetlen kattintással? Kánaán! :-)

```
Q123450_w2k_sp2_x86.exe -z
Q123451_w2k_sp2_x86.exe -z
Q123452_w2k_sp2_x86.exe -z
qchain.exe
```

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://www.microsoft.com/Downloads/Release.asp?ReleaseID=32362
[2] http://support.microsoft.com/support/kb/articles/Q309/5/08.ASP
[3] http://www.microsoft.com/Downloads/Release.asp?ReleaseID=32571
[4] http://support.microsoft.com/support/kb/articles/Q307/6/08.ASP
[5] http://www.microsoft.com/downloads/release.asp?releaseid=31154
[6] http://support.microsoft.com/support/kb/articles/q303/2/15.asp
[7] http://support.microsoft.com/support/kb/articles/Q296/8/61.ASP
[8] http://www.microsoft.com/downloads/release.asp?ReleaseID=29821

Digitális aláírás

Ez a cikk azért született, mert a múltkori hash-fejtegetésem kapcsán többen jelezték, hogy nem ártana ennél földhözragadtabb témakörben is körüljárni a működés módját. A digitális aláírásról hamarosan törvényünk lesz, mégis sokan vannak, akik nem ismerik e megoldás technikai hátterét, így a működés közben felmerülő hibák elhárításában sem fognak jeleskedni. Ez az írás laza cikksorozatot alkot az előző hash-művel.

Algoritmusok

Kezdjük a nyílt kulcsú titkosítás és digitális aláírás során felhasznált algoritmusokkal. Titkosítási algoritmusból alapvetően kétfélét ismerünk: a szimmetrikus és az asszimmetrikus fajtát. A szimmetrikus, vagy más néven egykulcsú titkosítás jellemzője, hogy ugyanaz a kulcs nyitja a ládikát, mint amelyikkel korábban bezártuk. Szoktam még hívni Sándor Mátyás titkosításnak is, mert abban is szerepel szimmetrikus titkosítás: Sándor Mátyás postagalamb lábára kötözött rejtjelezett írás segítségével üzen Raguzába. *(A rejtjelező eszköz nem más, mint egy – mondjuk 10 x 10-es - négyzetrács, melyen különböző helyen lyukak vannak. A lyukakon át kell írni az üzenetet a papírra, s a rács forgatásával fokozatosan válik szabaddá mind a 100 karakter. Ilymódon maximum 100 betűs üzenetek kódolására nyílik lehetőség.)* A gonosz Sárkány lelövi a galambot, de szerencsére nem jut értékelhető információhoz, mivel nem áll rendelkezésére a megfejtéshez szükséges rács *(a kulcs)*. A rács nélkül semmire sem megy. *(A helyszínt és a szereplők nevét teljesen fejből írtam, bárminemű tévedés lehetséges...)* Ebben az esetben a titkosírás megfejthetetlenségét a titkosító kulcs elérhetetlensége garantálja. Ilyen algoritmus például a jó öreg DES *(Digital Encryption Standard)*.
Ezek az algoritmusok azonban igen nehézkesen használhatók napjaink hálózatain, mert borszasztó kényelmetlen előre leosztani a titkosítási kulcsokat. A hálózatot nem használhatjuk kulcstovábbításra, mert az olyan lenne, mintha Sándor Mátyás a galamb bal lábára erősítené a rejtjelezett írást, jobb lábára meg a rácsot.
Ide tessenek lőni...
Még ha a kulcsok előzetes leosztását technikailag meg is oldhatnánk a hálózat kiküszöbölésével *(például telefonon, vagy SMS-ben küldenénk át)*, újabb, immár áthághatatlan problémát jelent az a szomorú tény, hogy sajnos nemcsak ismerősöknek szeretnénk titkosított levelet küldeni, hanem vadidegeneknek, vagy személytelen szervezeteknek is. Egy pályázati anyagot például lehet, hogy olyan címre kell eljuttatnunk, amelyről az égvilágon semmit sem tudunk a puszta emailcímen kívül. Nincs kinek a fülébe súgni a kulcsot. Be kell látnunk, hogy a szimmetrikus titkosítási algoritmusok megbuknak az Internet próbáján.

Nem baj, van mááásik!

A nyílt kulcsú, vagy aszimmetrikus titkosítási algoritmusok két kulcsot használnak. Amit az egyik zár, azt a másik nyitja és vica versa. Az egyik legismertebb ilyen algoritmus az RSA *(Rivest, Shamir, Adleman matematikusok nevéből)*, melynek matematikáját a következő számban fogom megfejteni *(már most lehet gondolkodni a nadrág megfelelő felkötésén)*. A Research rovatban e hónapban a 45. oldalon pedig az elliptikus titkosítás magyarázata olvasható. Az RSA nem titok, így működik:

- $szöveg^D \bmod N$=titok, ahol D és N a csukókulcs
- $titok^E \bmod N$= szöveg, ahol E és N a nyitókulcs

Kész csoda, hogy működik nem? Annyira letisztult, mint az $E=MC^2$. Az MD5 hash algoritmushoz képest maga a nirvána. A kulcsok tehát cserészabatosak, s ha az egyiket soha nem adjuk idegenkézre *(ez a privát kulcs)*, a másikat akár weblapunkra is kitehetjük *(ez pedig a publikus)*.

Nyílt kulcsú titkosítás - deszkamodell

Most lássuk a titkosítás elvi menetét nyílt kulcsú algoritmusunk felhasználásával. Az ábra a könnyű érthetőség kedvéért végletekig leegyszerűsíti a folyamatot, a valóság ennél bonyolultabb. A cikk végére a valóságot is belekeverjük a folyamatba, egyelőre maradjunk a deszkamodellnél:

A nyílt kulcsú titkosítás deszkamodellje

Első lépésben ① a feladó bejelenti a címzettnek, hogy titkosított üzenetet szeretne küldeni. Erre a címzett elküldi a hálózaton keresztül a publikus kulcsát ②, hogy azzal titkosítson a feladó. *(Itt jegyzem meg, hogy Hacker Henry természetesen lesben állt és a hálóról leszedi magának a publikus kulcsot.)* A feladó ezután a címzett publikus kulcsának segítségével titkosítja üzenetét, és eljuttatja a címzetthez ③. *(Hacker Henry mint mindig, most is éberen figyel, s a titkosított üzenetet is elkapja.)* A címzett fogja a titkosított üzenetet, kibontja a privát kulcsával, és boldogan elolvassa. És Hacker Henry? Nála van a titkosított üzenet, és a publikus kulcs, mellyel a titkosítás készült. Amit a publikus kulcs zárt be, azt a publikus NEM nyitja! Emlékezzünk az aszimmetrikus algoritmus lényegére: két kulcs van, amit az EGYIK zár, azt csak és kizárólag a MÁSIK nyitja! Így Henry maximum még egyszer titkosíthatja a levelet *(mintha mégegyszer ráfordítaná a zárat)*, de kinyitni nem képes. Mégegyszer: a publikus kulcsot bárkinek odaadhatjuk, hisz publikus. A privát kulcs védelméről kell gondoskodnunk: jelszóval védeni, Smart Cardon tárolni vagy lenyelni.

Digitális aláírás - deszkamodell

A kétkulcsú titkosítások kapcsán kell megemlékeznünk a digitális aláírásokról. Talán e lap olvasói közt nincs olyan, aki a digitális aláírást úgy képzeli el, hogy beszkenneljük a főnök aláírását és azzal a bitmappel manipulálunk valamit. A digitális aláírás felhasználói között azonban bőven vannak ilyen személyek. Őket józereknek hívjuk, és valahol a mi közös felelősségünk *(az Ön felelőssége is, Kedves Olvasó!)*, hogy megszabadítsuk kényszerképzeteiktől ezeket az emberi lényeket. Mit tud a digitális aláírás? Pont arra képes, mint a hagyományos: lehetetlenné teszi, hogy egy adott dokumentumot a rendelkezésünkre álló időintervallum alatt, s a kezünk ügyébe eső eszközkészlettel meghamisítsuk. Szó sincs rejtjelezésről, titkosításról! Egy aláírt *(pl. közbeszerzési)* szerződés esetleg teljesen publikus lehet - Utópia államban. Az aláírás szerepe, hogy a bárki által elolvasható dokumentumot ne lehessen könnyedén meghamisítani. Most nézzük meg a digitális aláírás folyamatának deszkamodelljét:

A digitális aláírás deszkamodellje

A jobboldali feladó saját kulcspárjának privát tagjával titkosítja a dokumentumot, de mellé teszi a kibontáshoz, elolvasáshoz nélkülözhetetlen publikus kulcsot. Így bárki képes kibontani és elolvasni az iratot, de senki sem képes azt meghamisítva ismét visszacsukni, hogy úgy tűnjön, hogy az eredeti feladó készítette. Miért nem? Mert a feladó szigorú őrizet alatt tartja a privát kulcsát, senkinek oda nem adja.

A valóság

Sajnos a gyönyörű elméletek sokszor megbuknak a gyakorlat próbáján. Nincs ez másként az RSA algoritmussal sem. Deszkamodelljeink a fenti formában működésképtelenek azon egyszerű okból fogva, hogy az RSA hihetetlenül lassan dolgozik. Ha valóban ráeresztenénk egy másfél megás Word dokumentumra, elfogadhatatlan válaszidővel lenne képes azt D-edik hatványra emelni *(nem is ezt teszi, de jól hangzik...:-)*! Összehasonlító tanulmányok szerint a nyílt kulcsú titkosítás három nagyságrenddel lassabb, mint szimmetrikus párja. A valóságban sohasem tisztán használjuk az RSA-t, hanem egy másik algoritmussal karöltve. Így mindkettőnek kihasználhatjuk az előnyeit, és megszabadulhatunk a hátrányoktól.

A titkosítás deszkamodelljét tehát módosítanunk kell: NEM a teljes dokumentumot titkosítjuk a címzett publikus kulcsával, mert az túl lassú lenne, hanem a feladó generál magának egy random szimmetrikus *(például 3DES)* kulcsot, azzal villámgyorsan titkosítja a dokumentumot, majd az RSA következik, és titkosítja a 3DES kulcsot, hogy azt ne kelljen SMS-ben átküldeni a fogadó félhez. Így a nyílt kulcsú titkosítás erősségét paradox módon a benne használt szimmetrikus algoritmus erőssége határozza meg. Hacker Henry választhat: vagy az RSA-t töri *(kulcshossztól függően mintegy négymilliárd év)*, hogy hozzájusson a 3DES kulcshoz, vagy közvetlenül nekiesik a doksinak és 3DES törést alkalmaz *(szintén mintegy négymilliárd év)*.

A digitális aláírás deszkamodellje pedig így változik: NEM a teljes dokumentumot titkosítjuk a feladó privát kulcsával, mert az túl lassú lenne, hanem ehelyett egy, a dokumentumra jellemző egyedi azonosító kerül be az RSA fogaskerekei közé. Kitalálták már mi az? A dokumentumból képzett hash *(például MD5)*! Tehát a hálózaton a következő „tárgyak" utaznak:

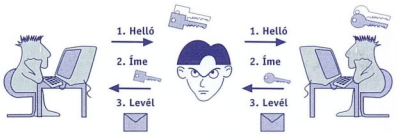
- az eredeti dokumentum
- a dokumentumból képzett hash, titkosítva a a feladó privát kulcsával
- a feladó publikus kulcsa

Az aláírás sértetlenségét pedig így állapítjuk meg:

1. Vedd a megkapott doksit, és hash-eld meg
2. Vedd ki a csomagból a feladó publikus kulcsát, és nyisd ki a titkosított hash-t
3. Ha az előző két lépés azonos hash-t ad, a doksi sértetlen.

The Man In The Middle

Szép is, jó is, de nem azért találták fel a kapanyelet, hogy az időnként ne süljön el. Vegyük elő ismét a titkosított dokumentumküldés deszkamodelljét, de most lesz mégegy szereplőnk: Hacker Henry, aki a két kommunikáló fél közé ékelődve minden hálózati forgalmat elkap, és saját kényekedve szerint módosít:



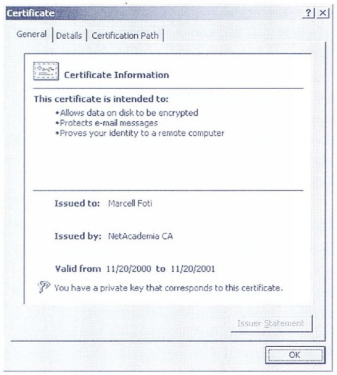
Hacker Henry „bedolgozik" a kulccserébe

A feladó bejelenti ①, hogy titkosított üzenetet szeretne küldeni. Ezt az üzenetet Hacker Henry elfogja, kiveszi a hálózatból, és saját üzenetével helyettesíti. A címzett úgy érzékeli, hogy nem Henry kíván neki titkosított üzenetet küldeni. Szokás szerint elküldi a publikus kulcsát a vélelmezett feladónak, tehát Henrynek ②, aki azt köszöni szépen, majd saját kulcspárjának publikus részét küldi el a feladónak *(ezt szemlélteti, hogy itt más a kulcs alakja)*. A feladó kap egy publikus kulcsot, hite szerint a címzettét, s azzal titkosítja az üzenetet *(illetve, mint tudjuk: az üzenet titkosítására valójában felhasznált szimmetrikus kulcsot)*. Ezt elküldi Henrynek ③, aki vidáman elolvassa a saját privát kulcsával. Henry nem tett mást, mint kicserélt minden kulcsot, amely átment a keze között. Magát a támadásformát igen nehéz végrehajtani, mert kevés olyan pont van az Interneten, ahol a csomagok egytől egyig átmennek - kizárólag a feladó és a címzett Internetszolgáltatójának alkalmazottai lennének képesek erre. De az elvi lehetőség is lehetőség! Tenni kellene ellene valamit! Biztosítani kellene a hálózaton utazó kulcsok kicserélhetetlenségét. Hogyan? Hát digitális aláírással! Kellene egy harmadik fél, akiben mindenki megbízik, és nem tesz mást, mint aláírogatja a kulcsokat. Olyan, mint egy közjegyző. Az ő informatikai neve:

Certificate Authority

Vagy röviden CA. A CA feladata nem más mint mások publikus kulcsainak hitelesítése, hogy ha kapunk egy publikus

kulcsot mely tökéletesen úgy tűnik, mintha Samukától származna, akkor biztosak lehessünk abban, hogy ez valóban azé, akié. Minden egyes kulcspárunk publikus tagját beküldjük a CA-nak, aki arra ráteszi saját digitális aláírását - hisz neki is van saját kulcspárja, amivel aláírogathat. Az aláírt publikus kulcs neve hitelesítési bizonyítvány.

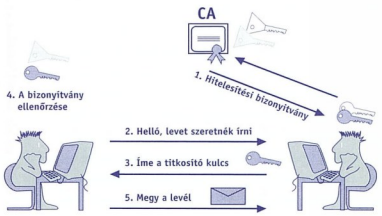


Egy hitelesítési bizonyítvány

Mit jelent ez a gyakorlatban? Azt, hogy az én publikus kulcsomat bárki elolvashatja, de a rendelkezésére álló idő alatt meg nem hamisíthatja, hisz ahhoz szüksége lenne a CA privát kulcsára. Minden megbízható CA úgy vigyáz a privát kulcsára, mint a szeme fényére, vagy talán még annál is jobban. Jobb helyeken *(pl. Verisign egyes komolyabb bizonyítványai)* a CA nemcsak hogy nem lóg az Interneten, de be sincs kapcsolva, sőt, komoly, akár fegyveres őr védelme alatt áll. Már csak azt kell biztosítani, hogy

1. amikor én beadom hitelesítésre a publikus kulcsomat, menet közben senki se cserélhesse le, és
2. senki se léphessen fel a nevemben kulcskérőként

E két szabály úgy teljesíthető a legkönnyebben, ha nem hálózaton, hanem más módon, például személyesen juttatom el a publikus kulcsomat a CA-hoz, ami egyben jó lehetőséget nyújt a második pont betartására, hisz ha másképp nem, a személyi igazolványommal hitelt érdemlően bizonyíthatom a testület előtt, hogy én én vagyok. *(De még a legjobbak sem tévedhetetlenek. Tavasszal a Verisign hitelesített két, szoftvereredetiség igazolására szolgáló publikus kulcsot a Microsoft nevére. A bökkenő csak az, hogy - amint az később bebizonyosodott - akik besétáltak az irodába, NEM álltak a Microsoft alkalmazásában. Így most van kint a világban két olyan hitelesítési bizonyítvány, mely semmisnek tekintendő.)* Amint rákerült a CA aláírása a publikus kulcsomra, már használhatom is. Íme a folyamat:



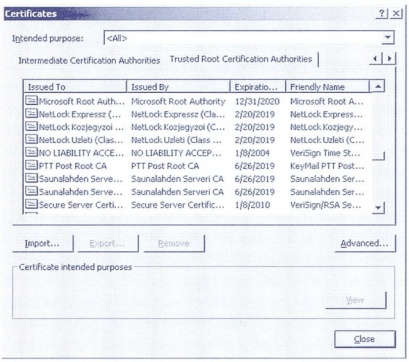
A kulcshitelesítés folyamata

A vindózom generál egy RSA kulcspárt, melynek publikus tagját beküldöm ①, és jó pénzért aláíratom egy közjegyzővel, s ezután ha valaki kéri a publikus kulcsomat, akkor nem pucéron adom oda neki, hanem a CA által aláírt, hitelesített példányt kapja ③. Henry ugyan továbbra is elkaphatja, de meghamisítani, lecserélni nem tudja, mert nem áll rendelkezésére a CA privát kulcsa. Maximum a saját kulcspárjának hitelesítésére bírja rávenni a CA-t, de azon a kulcson onnantól virítani fog, hogy „ez Hacker Henry publikus kulcsa". A feladó a megkapott bizonyítvány alapján, a CA publikus kulcsával elvégzi az aláírás valódiságának ellenőrzését ④, s ha minden rendben, akkor használatba is veszi. Ha eltérést tapasztal, sikít.

Kész. Azazhogy. Izé. Tessékmondani, hogy került a levélfeladó oldalára a CA publikus kulcsa? Talán csak nem hálózaton keresztül? Mert ha igen, azt Henry bizonyára le is cserélte menet közben. Hol itt a biztonság? Hol?

Trusted Root Certificate

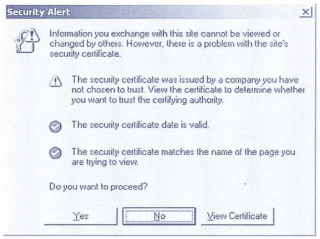
Hát ott, hogy a CA publikus kulcsa NEM hálózaton keresztül kerül a mi számítógépünkre. Hanem akkor hogyan? Vettük a boltban? Igen, igen. Az operációs rendszerrel együtt vettük a szoftverboltban, és CD-n jutott el hozzánk. Ha Henry képes lenne préselt CD lemezeken kicserélni az azon tárolt kulcsot, nagy-nagy bajban lennénk. De mivel erre nem képes, bizton állíthatjuk, hogy a CA-k publikus kulcsa sértetlenül jutott el hozzánk:



A megbízható CA-k listája a Windowsban. A telepítő CD-ről került oda valamennyi, így a magyar NetLock is!

Untrusted Root

Az előző oldalon látható bizonyítvány „Issued by" sora szerint a kibocsátó a NetAcademia CA. Tökéletes bizonyítványokat képes kibocsátani. Egy nagy hibája van: semelyik gyárilag telepített Windowsban sincs benn ennek a CA-nak a publikus kulcsa. Ez azt jelenti, hogy az innen kibocsátott tanúsítványok leellenőrzésére nem túl sokan képesek a világban. Ez végülis nem gond, mert egy ilyen tanúsítványból ki lehet szedni a benne lévő publikus kulcsot, éppen csak annak hitelességét nem tudjuk ellenőrizni – de attól még az egy jó RSA publikus kulcs. A benne lévő kulcs hamisítatlanságának leellenőrzésére csak azok a gépek lesznek képesek, amelyekre valamilyen módon eljuttatom a CA publikus kulcsát. Minden más helyen ez fogad:



Nem ellenőrizhető a bizonyítvány – de attól még működik!

Ilyenkor két választásunk van:

- Vagy elfogadjuk a bizonyítványt „as is", vagyis ellenőrizetlenül
- Vagy elvetjük, mint megbízhatatlant

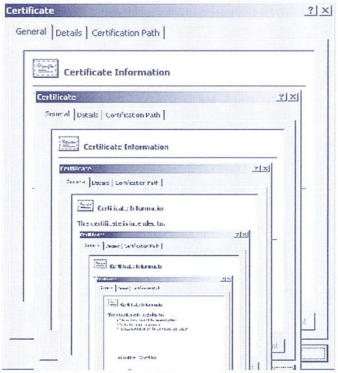
Az első lépésnek is lehet értelme. Ha csak az a gond, hogy nincs meg nálam a CA publikus kulcsa, és mondjuk egy weblapot kellene elérnem HTTPS://-en *(SSL)* keresztül, megfontolnám a jeszt, ha az SSL nem titkosítási okokból kell, hanem például tűzfal megkerülésére :-). *(Az SSL egyik kiváló tulajdonsága, hogy a titkosított webforgalomba a köztünk lévő tűzfal nem tud belekotyogni, hisz titkosított. Ezt a trükköt használhatjuk Exchange 2000 OWA használatára olyan tűzfalon keresztül, mely nem engedi át a WebDAV-ot. De ha titkosítom...!)*

CA hierarchia

Az eldöntendő igenen és nemen kívül lehet mégegy esélyünk: ha a CA nem árván áll a világban, hanem egy olyan CA hierarchia része, melynek gyökerét ismeri a Windows, akkor a probléma nem lép fel, mivel:

- Ha az idegen CA által kibocsátott bizonyítványt nem tudja ellenőrizni a Windows, megpróbálja magának a CA-nak a hitelesítését
- Ha a CA hitelesítése sem megy, megpróbálja a CA fölötti CA hitelesítését
- Ha az sem megy az afölöttit mindaddig, amíg el nem éri a CA hierarchia csúcsát – hisz annak publikus kulcsa ott virít telepítés óta a Windowsban.

Ez azt jelenti, hogy a fa bejárásához állandó hálózati kapcsolat kell? Szó sincs róla! Itt és most szögezzük le: a lokálisan meglévő Root Certificate-k miatt a Verisign *(és a többi root)* gépeit akár ki is lehet kapcsolni. Ha ez a rootra igaz, nyilván az egymásra épülő CA-alantasokra is igaz. A CA hierarchia leellenőrzése tipikusan lokálisan megtörténik, mivel *(jobbára)* minden bizonyítvány magával hordozza a szülei és dédszülei digitális aláírását is, valahogy így:



Egymást hitelesítő bizonyítványok láncolata

Ha elmúlik a bizalom

A bizonyítványokon alapuló azonosítás feltételez egy jó adag bizalmat a hitelesítőszervezet iránt. A CA-k általában rá is szolgálnak a bizalomra, és nem adják ki privát kulcsukat. Nem így a végfelhasználók! ők bizony elhagyják, publikussá teszik, eladják a velük „összenőtt" privát kulcsot, s ha ez kiderül, szükségessé válhat a bizonyítvány visszavonása. Erre való az úgynevezett Certification Revocation List, vagyis érvénytelenné vált bizonyítványok feketelistája. Minden aláírásellenőrzőnek elvileg kutya kötelessége lenne ellenőriznie, hogy a bizonyítvány a matematikai megfelelőségen túl megbízható-e még? Mivel azonban ez állandó hálózati forgalommal járna, a CRL ellenőrzése a munkaállomáson alapállapotban ki van kapcsolva.

A nyílt kulcsú rendszereket az X.509 szabvány írja le. A bizonyítványok kezelését pedig a PKCS #7-#10 szabványok. A következő alkalommal a PKCS #1 leírással, vagyis az RSA algoritmussal foglalkozunk!

Fóti Marcell
MCSE is.

Internet Security and Acceleration Server *(V. rész)*

Bővítmények *(Extensions)*

A bővítmények további funkciókkal gazdagítják az ISA Servert. Az ISA Server által kiszolgált kapcsolatok adatfolyamaihoz hozzáférve lehetővé teszik, hogy ezeken további szűréseket és megfigyeléseket végezzünk. A POP3 szűrő *(POP3 filter)* például nem meglepő módon arra szolgál, hogy kiszűrje a közzétett POP3 kiszolgáló ellen elkövetett buffer overrun támadásokat.

E havi cikkünkben az ISA Server-ben gyárilag megtalálható bővítményeket tekintjük át, természetesen nagyobb figyelmet fordítva azokra, melyek biztonsági szempontból a legfontosabbak. A bővítmények beállításait a megszokott ISA Server MMC-n keresztül érhetjük el a tömb, majd ezután az Extensions/Application filters kiválasztásával. Ha valaki esetleg további részletekre kíváncsi, vagy nem találja meg cikkünkben a kívánt segítséget, érdemes megtekinteni az ISA Server súgóját, ahol a bővítmények igazán elismerésre méltó részletességgel vannak ismertetve.

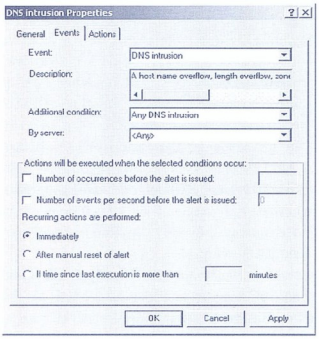
DNS behatolásészlelő szűrő *(DNS intrusion detection filter)*

A DNS intrusion detection filter elcsípi, majd elemzi a belső hálózat felé irányuló DNS forgalmat. Az alábbi négy forgalomtípus figyelését állíthatjuk be:

- DNS név túlcsordulás *(DNS hostname overflow)*. A DNS név túlcsordulás akkor történik, ha a DNS válasz egy névfeloldási kérésre túllép egy bizonyos meghatározott hosszúságot. Azokban az alkalmazásokban, melyek nem ellenőrzik a név hosszát, túlcsordulhatnak a belső pufferek, ezzel pedig lehetővé válhat tetszés szerinti parancsok végrehajtása a megcélzott számítógépen *(Ez itt a reklám helye: a NetAcademia security tanfolyamán bővebb ismeretek szerezhetők a buffer overrun-ról, és annak veszélyeiről)*.
- DNS hossz túlcsordulás *(DNS length overflow)*. Az IP címekre vonatkozó DNS válaszok tartalmaznak egy hossz mezőt, ami négy bájt hosszúságú lehet. Egy hibásan *(bizonyos szempontból mindegy, hogy véletlenül vagy szándékosan hibásan)* formázott DNS válasz következtében túlcsordulhatnak a belső pufferek, és lehetővé válhat tetszés szerinti parancsok végrehajtása a megcélzott számítógépen.
- DNS zone transfer from privileged ports *(1–1024)*, vagyis DNS zónaátvitel kérés az alsó portokról. Ez akkor történik, amikor egy rendszer olyan DNS ügyfélalkalmazást használ, amely a belső kiszolgálókról való zónaátvitelnél forrásportként egy alsó portot *(1-1024)* ad meg.
- DNS zone transfer from high ports *(1024 fölött)*. A fentihez teljesen hasonló, de ez a magas forrásportokat megadó kapcsolatokat figyeli.

Mindenképpen ajánlott a DNS intrusion detection filter használata, és annak beállítása, hogy mind a négyféle tevékenységet figyelje. A beállítások elvégzéséhez válasszuk ki a DNS intrusion detection filter objektumot, jobbklikk, majd válasszuk a Properties menüpontot.

Fontos tudni, hogy erre a szűrőre vonatkoznak a DNS intrusion alert settings-ben beállított riasztások. Ennek a riasztásnak bekapcsolva kell lennie *(persze a működéshez nem feltétlen szükséges, csak erősen ajánlott)*, és jó, ha olyan riasztást küld, ami igen gyorsan magára vonja az illetékes rendszergazda figyelmét. Alábbi ábránk az általunk javasolt beállítást tartalmazza. E beállítások következtében bármely DNS intrusion eseményre ISA Server-ünk elkezd riasztásokat küldeni. *(A beállításokat az alerts alatt a DNS intrusion alert tulajdonságlapjának (Properties) előcsalogatásával érhetjük el).*



A DNS Intrusion Detection riasztás beállításai

Van még néhány dolog, amit jó tudni a DNS intrusion detection filter-ről: Először is, ez a szűrő a belső hálózat felé irányuló forgalmat figyeli, az ISA Server-re irányuló gyanús kapcsolatok nem fognak riasztást előidézni. Másodszor, bár ez a filter igen hasznos tud lenni, ne alapozzuk erre teljesen a védelmünket. A Windows 2000 igen erősen a DNS-re támaszkodik. Ha egy támadó megszerzi belső hálózatunk zónafájlját/zónafájljait, az igen nagy előnyt jelent számára. Éppen ezért bizonyosodjunk meg róla, hogy a belső DNS kiszolgálók nem látszanak a külső hálózat felé. Ha mégis szükséges DNS adatokat a nyilvánosság elé tárni, tegyük ezt a DMZ-ben *(DMZ-ből)*, és csak azok az adatok legyenek elérhetők, melyek tényleg szükségesek a külső felhasználóknak. *(Ha minden igaz, következő havi cikkünkben épp ezt a témakört járjuk majd körül részletesebben)*

FTP access filter

Ez a bővítmény egy olyan állatfajta, aminek beállításával kevés teendőnk lesz, mivel összesen egyféle beállítási lehetőség van. Vagy bekapcsoljuk, vagy nem. Ez a szűrő továbbítja az FTP kéréseket a biztonságos címfordítás *(SecureNAT)* ügyfelektől a tűzfalszolgáltatáshoz. A szűrő dinamikusan megnyitja az FTP protokoll által igényelt másod-

lagos portokat, és elvégzi a NAT ügyfeleknek szükséges címfordítást. Általában ajánlott ezt a szűrőt bekapcsolva tartani *(ez az alapértelmezett állapota is)*.

H.323 protokollszűrő

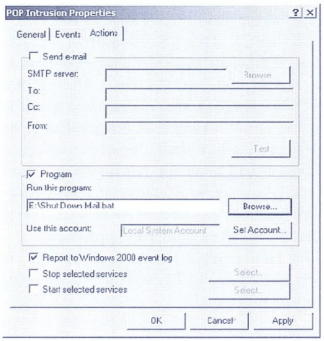
A H.323 protokollszűrő biztosítja a H.323 alkalmazások *(például NetMeeting)* számára, hogy működhessenek az ISA Serveren keresztül is. Ha a szűrő be van kapcsolva, megnyitja az ISA Server külső lábán az 1720-as portot, éppen ezért érdemes kikapcsolva tartani, ha nem használjuk a H.323 protokollt.

HTTP átirányítást végző szűrő *(HTTP redirector filter)*

Ez a szűrő annyiban hasonlít az FTP access filter-hez, hogy továbbítja a tűzfal és SecureNAT ügyfelektől érkező HTTP kéréseket a Web Proxy szolgáltatáshoz. Ez a szűrő szükséges az ISA alapvető funkcionalitásának biztosításához, ezért tartsuk bekapcsolva *(ez az alapértelmezett beállítás is)*. Ha követjük azt az ajánlást is, mely szerint mindig a Web Proxy ügyfelet használjuk a HTTP elérés biztosítására, kapcsoljuk be azt a HTTP redirector opciót, mely visszautasítja a tűzfal és SecureNAT ügyfelektől érkező HTTP kéréseket.

POP behatolás-észlelő szűrő *(POP intrusion detection filter)*

A Post Office Protocol *(POP)* behatolás-észlelő szűrő elkapja, és elemzi a belső hálózat felé irányuló POP forgalmat. Ez az alkalmazásszűrő a POP buffer overflow támadásokat figyeli, és észleli. Az ilyen támadás a DNS szűrőnél leírtakhoz hasonlóan történik, vagyis a támadó megpróbál buffer overflowt előidézni a kiszolgálón, és ha ez sikerült, szabad a pálya. A szűrő felhasználásának természetesen vannak korlátai *(ahogy az a behatolás-észlelő technológiáknál megszokott is)*. Ettől függetlenül, ha már POP3 kiszolgálót teszünk közé, mindenképpen ajánlott ennek a szűrőnek a használata *(ez az alapértelmezett beállítás is)*, és a hozzá tartozó riasztás DNS szűrőnél beállítotthoz hasonló beállítása is.



POP3 behatolásészlelés riasztás által végrehajtott cselekvés beállítása

Az ISA Server megszakítja a POP kapcsolatot a riasztás megtörténtekor. Ez viszont nem tartós! Vagyis a kedves támadó a kapcsolat megszakítása után azonnal újra tud kapcsolódni a közzétett POP kiszolgálóhoz. Ennek elkerülésére van lehetőségünk: a riasztást beállíthatjuk úgy is, hogy állítsa le a POP3 szolgáltatást, ha behatolási kísérletet észlelt. Ezt a POP intrusion alert Actions füle alatt tehetjük meg. Arra, hogy ezt beállítsuk-e, nincs kötelező érvényű szabály, de érdemes végiggondolni a következőket:

- Mennyire nélkülözhetetlen ez a szolgáltatás?
- Mennyi idő alatt lehet beállítani a szükséges védelmet, és újraindítani a szolgáltatást?
- Kit érint a szolgáltatás kiesése?

RPC szűrő *(RPC Filter)*

Az RPC szűrő, bármilyen hihetetlenül is hangozzék, RPC kiszolgálók *közzétételét* teszi lehetővé, így ezek is elérhetővé válhatnak a külső ügyfelek számára. A szűrő leggyakoribb alkalmazása az Exchange Serverek közzététele; de ha úgy döntünk, hogy Exchange kiszolgálónkat inkább az SMTP, POP3, és/vagy IMAP4 protokollok használatával tesszük közzé, ajánlott ezt a szűrőt kikapcsolni.

SMTP szűrő *(SMTP filter)*

A Simple Mail Transfer Protocol *(SMTP)* szűrő elcsípi az összes SMTP forgalmat, ami az ISA Server SMTP portjára *(25-ös port)* érkezik. A szűrőnél lehetséges olyan szűrési mechanizmusok beállítása, amelyek számos paraméter (például küldő, csatolt fájl típusa, csatolt fájl neve) alapján visszadobhatják a közzétett SMTP kiszolgálók felé tartó üzeneteket. A szűrő megvizsgálja az adott forgalmat, és csak akkor engedi tovább, ha a szabályok ezt megengedik. Az üzenetkezelés megfelelő szűrését üzembe állítani igazán szép feladat. Erre sem lehet igazán jó kötelező jellegű „receptet" adni, hiszen vannak olyan tényezők, melyek minden ISA Server-nél mások és mások *(például belső levelezőrendszer, vírusszűrés)*. Itt is érvényes az, ami mindenhol: ha elakadna valaki és a cikkben nem talál választ, használja bátran az ISA súgóját.

Fontos tudni, hogy a többi bővítménnyel ellentétben az SMTP szűrő alapértelmezésben nincs bekapcsolva. Ha ISA Server mögötti SMTP kiszolgálókat teszünk közzé, ajánlott bekapcsolni, és az alább leírtak szerint beállítani az SMTP szűrőt, de figyeljünk nagyon, amikor ezt tesszük, mert akár saját magunk is előidézhetünk szolgáltatás megtagadást *(DoS)*. Mielőtt nekilátunk egy éles ISA Serveren az SMTP szűrő beállításának, feltétlen olvassuk el az **[1]** knowledge base cikket.

Csatolt állományok *(Attachments)* fül

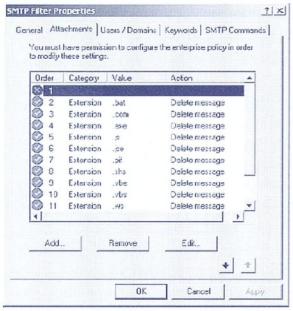
Az ez alatt a fül alatt található opciók azt a célt szolgálják, hogy a leveleket a csatolt állományok neve és kiterjesztése alapján szűrjük. Azoknak a csatolt fájloknak a blokkolása, melyek futtatható állományokra utalnak, igen hasznos lehet a vírusvédelem szempontjából, de mindenképpen figyelembe kell vennünk két dolgot.

1. Általában nem célszerű minden olyan csatolt fájlt blokkolni, ami veszélyes lehet, hiszen ezek között biztosan lesz olyan is, ami cégünk működéséhez szükséges *(például Word dokumentumok is tartalmazhatnak makrókat, mégsem kell ész nélkül eldobni őket. Legyen ez a vírusirtók feladata)*.
2. Az ISA Server csatolt állomány szűrőjében van egy ismert hiba. Az időszakos szabályok érvénytelenné válhatnak, ilyenkor egy szép piros x jelenik meg rajtuk, a szűrő viselkedése pedig előre nem látható módon megváltozhat. További információk erről a **[2]** címen találhatók.

A fentiek miatt javasolt, hogy ha használni akarjuk ezt a szűrőt, előbb teszteljük a majdani szabályokat, ezzel biztosítva, hogy az éles rendszerben az elvárásoknak megfelelően fognak működni.

Annak eldöntése, hogy mely csatolt állományokat kívánjuk blokkolni, sok tényezőn múlik, de azért van néhány olyan dolog, amit érdemes figyelembe venni. Én személy szerint például nehezen tudom elképzelni, hogy épeszű ember pif kiterjesztésű fájlt küldjön. Bezzeg a Sircam! Az alábbi lista tartalmazza a leggyakrabban előforduló futtatható állományokat, és szkriptfájlokat.

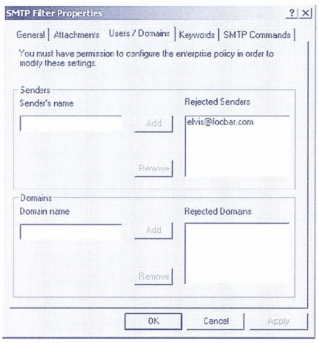
.bas, .hta, .msp, .url, .bat, .inf, .mst, .vb, .chm, .ins, .pif, .vbe, .cmd, .isp, .pl, .vbs, .com, .js, .reg, .ws, .cpl, .jse, .scr, .wsc, .crt, .lnk, .sct, .wsf, .exe, .msi, .shs, .wsh



Szűrés csatolt állomány neve/kiterjesztése alapján

Felhasználók/Tartományok *(Users/Domains)* fül

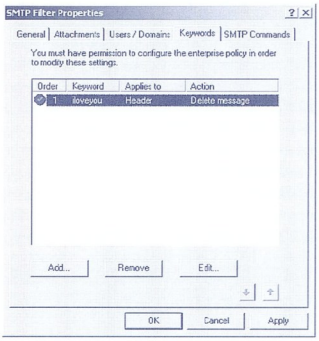
Ezek a beállítások lehetővé teszik, hogy adott felhasználótól *(például jozsi@hekkersz.hekk)*, vagy adott tartományból *(például mindenki a hekkersz.hekk tartományból)* érkező leveleket kiszűrjünk, így ezek a felhasználók nem tudnak az SMTP kiszolgálónkon keresztül levelet küldeni. Ennek elsődleges célja nyilván az, hogy a spamet, és a veszélyes tartalmakat terjesztőket eleve kizárjuk a levélforgalomból.



Szűrés felhasználók és tartományok szerint

Kulcsszavak *(Keywords)* fül

Ennek a segítségével tudjuk az üzeneteket a fejlécükben, vagy magában az üzenetben előforduló kulcsszó alapján szűrni. Ez segíthet például bizonyos vírusok terjedésének megakadályozásában *(például szűrés az ILOVEYOU szóra)*, de szűrhetünk akár arra a bizonyos s-el kezdődő, x-re végződő szóra is...

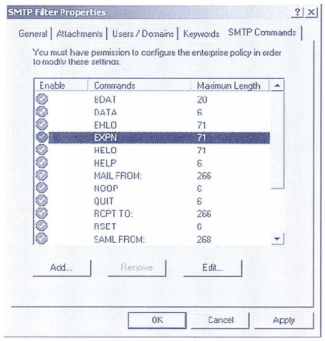


Szűrés kulcsszó alapján

SMTP parancsok *(SMTP Commands)* fül

Az SMTP commands fülön beállíthatjuk azokat a határértékeket, melyek az SMTP kiszolgáló nevében ügyködő ISA Server által elfogadott egyes parancsok maximális méretét határozzák meg. Ajánlott az alapbeállítások megtartása, de célszerű a következő parancsokat letiltani:

- EXPN. Az EXPN parancs megmutat egy disztribúciós listát, így a parancsot kiadó láthatja a listában szereplők címeit. Ez általában nem kívánatos.
- VRFY. A VRFY parancs arra használható, hogy a parancsot kiadó ellenőriztesse az adott postafiók létezését. Jó sok címet lehetne így megtudni. Irtsuk ki!
- TURN. A TURN parancs megfordítja az ügyfél és kiszolgáló szerepkörét. Ez általában olyan környezetekben használatos, ahol nem állandó kapcsolat *(vagy IP cím)* van. Ilyenkor a levelek például az Internetszolgáltató levelezőszerverére esnek be, majd a kapcsolat felépülésekor a cég levelezőszervere továbbítja a kimenő leveleket a szolgáltatóhoz, és kiadja a TURN parancsot, vagyis megkéri a szolgáltatónál levő kiszolgálót, hogy továbbítsa a nála várakozó befelé jövő leveleket. Ha valakinek így működik a levelezése, az természetesen ne tiltsa le ezt a parancsot.
- NOOP. A NOOP kicsit hasonlít a pinghez, hiszen egy válaszkódot kér a kiszolgálótól. Ez nélkülözhető, így áldozatul esik a paranoiának. :)



SMTP parancsok

SOCKS V4 szűrő *(SOCKS V4 filter)*

A SOCKS szűrő a SOCKS alkalmazásoktól érkező kéréseket a tűzfalszolgáltatáshoz továbbítja.

Az ISA Server ezután ellenőrzi a hozzáférési szabályokat, hogy eldöntse, hogy az ügyfélnek engedélyezett-e a külső hálózattal folytatott kommunikáció. A szűrő használata elsősorban a UNIX alkalmazásokkal való használhatósághoz szükséges. Alkalmazásának nincsenek biztonsági kockázatai, de jó tudni, hogy a SOCKS version 4 nem támogatja a felhasználó-azonosítást. A hozzáférésszabályozás SOCKS használata esetén ezért csak IP cím alapján valósítható meg.

Streaming Media szűrő *(Streaming Media Filter)*

A streaming media szűrő használatával javítható az Internetelérés sebessége, mert a szűrő képes az „élő adást" szétosztani.

Ez azt jelenti, hogy a szűrő az információt csak egyszer szerzi meg a külső hálózatról, majd elérhetővé teszi azt több belső ügyfél számára. Speciális biztonsági beállítások *(és kockázatok)* nem tartoznak a szűrőhöz.

Összefoglalás

Röviden az alábbi beállítások javasoltak az ISA Server bővítményeinél:

- Kapcsoljuk be a DNS behatolásészlelő szűrőt, és állítsuk be úgy, hogy figyelje mind a négyféle tevékenységet. Állítsuk be hozzá a megfelelő riasztást is.
- Ne feledjük, hogy a DNS behatolásészlelő szűrő használata nem helyettesítheti a körültekintő tervezést. Bizonyosodjunk meg róla, hogy a belső DNS kiszolgálókat nem lehet kívülről elérni.
- Kapcsoljuk ki a H.323 szűrőt, ha nincs szükségünk a H.323 protokollra.
- Ha követjük azt az ajánlást is, mely szerint mindig a Web Proxy ügyfelet használjuk a HTTP elérés biztosítására, kapcsoljuk be azt a HTTP redirector opciót, mely visszautasítja a tűzfal és SecureNAT ügyfelektől érkező HTTP kéréseket.
- Ha POP kiszolgálót teszünk közzé, kapcsoljuk be a POP behatolás-észlelő szűrőt.
- Ha lehet, kapcsoljuk ki az RPC szűrőt.
- Ha SMTP kiszolgálót teszünk közzé, kapcsoljuk be az SMTP szűrőt, és tiltsuk le az alábbi parancsok használatát: EXPN, VRFY, TURN, NOOP. Használjuk az üzenetek szűrésénél a csatolt állományok típusa és neve alapján történő szűrést is, ezzel sok veszélyes fájlt kiszűrhetünk. Állítsunk be bátran kulcsszó alapján végzett szűrést is! Ennek különösen egyes vírusok terjedésének megakadályozásában vehetjük nagy hasznát.
- Ne feledjük el azt sem, hogy szűrhetünk felhasználó és tartománynév szerint is, ami segíthet például a spam kiszűrésében.
- Ha SOCKS ügyfeleket is szeretnénk használni, kapcsoljuk be a SOCKS szűrőt, de ne feledjük, hogy ez esetben a hozzáférések szabályozása csak IP cím alapján történhet, felhasználók szerint nem.

Itt a vége, fuss el véle! A következő hónapban terveink szerint a biztonságos DNS környezet kialakítását fogjuk áttekinteni.

BP, MCSE

A cikkben szereplő URL-ek:
[1] http://support.microsoft.com/support/kb/articles/Q292/0/10.ASP
[2] http://support.microsoft.com/support/kb/articles/ Q292/0/14.ASP

ASP Suli (X. rész) Adatkezelés ADO-val

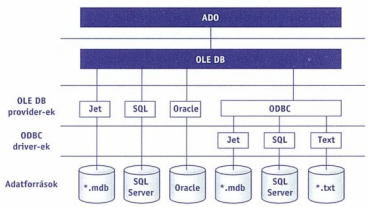
E havi számunkban az adatkezeléssel, az adatbázisok, adattárak elérésével foglalkozunk. Ebben az ActiveX Data Objects *(ADO)* objektumcsalád lesz segítségünkre, amely a DAO-t leváltva és az ODBC-t egyre inkább kiszorítva lett napjainkra az egyik leggyakrabban használt adatelérési felület. Az adatbáziskezelés *(sőt, még az ADO professzionális használata is)* olyan szerteágazó és részletes téma, hogy az nemcsak az újságot, de még egy teljes könyvet is megtöltene; éppen ezért a következő néhány oldalon az ADO alapjait, valamint az alapvető adatkezelési műveleteket mutatjuk be. Akit a téma részletesebben is érdekel, az látogassa meg a Weben az ADO objektummodell referenciáját **[2]**. A példaprogramok – mint mindig – az **[1]** címről letölthetők.

Mi az az ADO?

Az ADO egy COM objektummodell, ami az OLE DB nevű adatbáziskezelő komponens „tetején" működik. Az OLE DB bevezetésének célja az volt, hogy adatbázistól független, uniform adatkezelési felületet nyújtson a programozóknak. Az OLE DB-t és a különféle adatbázisokat providereknek nevezett szoftverkomponensek kapcsolják össze *(ezeket a providereket általában az egyes adatbáziskezelők gyártói készítik, de a Windows önmagában is tartalmaz néhányat)*. Az ADO tehát nem más, mint egy COM felület az OLE DB felett, ami lehetővé teszi, hogy azt minden ActiveX-kompatibilis nyelvből elérhessük *(Visual Basic-ből, ASP-ből, egyszerű VBScript és JScript scriptekből)* – míg az OLE DB-t szinte csak a C programozók használták.

Az uniformizált adatelérés ötlete nem új: a Windows évek óta tartalmazza az úgynevezett ODBC réteget. Az ODBC hasonló módon működik, mint az OLE DB: az adatforrások és az ODBC közötti kapcsolatot ott is külön „eszközmeghajtók", termékspecifikus ODBC driverek tartották fenn. Az ADO előnye, hogy az OLE DB providereknek köszönhetően az ODBC kikerülésével is elérhetjük az adatforrásokat – ugyanakkor létezik egy OLE DB provider, ami lehetővé teszi, hogy ADO-n keresztül ODBC adatforrásokat kezeljünk.

Az ADO, az OLE DB providerek és az ODBC driverek közötti viszonyt jól szemlélteti az alábbi ábra:



Az ADO, az OLE DB provider-ek, az ODBC driver-ek és az adatforrások viszonya

Egy szó mint száz: az ADO segítségével könnyen és azonos módon érhetjük el az adatforrásokat, akár közvetlen OLE DB, akár ODBC kapcsolaton keresztül. A használható adatforrások száma csak attól függ, hogy mihez találunk providert, esetleg ODBC drivert: manapság a tabulált szöveges fájltól az Access vagy SQL relációs adatbázisokon át akár egészen az Exchange Server levéltáráig sokmindenhez hozzáférhetünk. A Windows 2000-ben alapértelmezésben telepített OLE DB providerek:

- Microsoft Jet 4.0 OLE DB Provider – Access *(Jet, *.mdb)* adatbázisokhoz
- Microsoft OLE DB Provider for Indexing Service – az Indexing Service eléréséhez
- Microsoft OLE DB Provider for Internet Publishing – webtárak, FrontPage webek eléréséhez
- Microsoft OLE DB Provider for ODBC – ODBC adatforrásokhoz
- Microsoft OLE DB Provider for Oracle – Oracle adatbáziskezelőkhöz
- Microsoft OLE DB Provider for SQL Server – SQL Server adatbáziskezelőkhöz
- Microsoft OLE DB Simple Provider – egyszerű szövegfájlokhoz
- OLE DB Provider for Microsoft Directory Services – címtárszolgáltatások eléréséhez

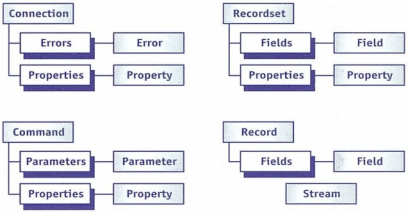
Ezeken kívül, ODBC segítségével elérhetjük az alábbiakat:

- Vesszővel elválasztott vagy tabulált szövegfájlok *(*.txt, *.csv, *.tab)*
- Access adatbázisok, Excel adattáblák *(*.mdb, *.xls)*
- dBase, Visual FoxPro, Paradox adatbázisok *(*.dbf, *.db)*
- SQL Server, Oracle adatbáziskezelők

További providerek természetesen külön telepíthetők *(újabbak jelennek meg például az SQL Server vagy az Exchange Server telepítésekor)*. A Microsoft Data Access Components *(MDAC)* legújabb verziója *(a cikk írásának pillanatában a 2.6)* a **[3]** címről tölthető le.

Az ADO objektummodell

Bemelegítésképpen lássuk az ADO objektumcsalád tagjait:



Az ADO objektummodellje (a kollekciókat árnyékkal jelöltük)

Magyarázat, egyelőre röviden, mert az egyes objektumokat majd használat közben fogjuk igazán megismerni:

- Connection objektum: az egyik legfontosabb objektum, egy adatbázissal való kapcsolatot reprezentál. A munkát általában a Connection objektum létrehozásával, majd az adatbázishoz való kapcsolódással kezdjük
- Recordset objektum: a másik legfontosabb objektum. A Recordsetet kezdetben úgy képzeljük el, mint egy eredménytáblát, amelyben soronként találhatók a rekordok. Minden rekordnak több mezője *(Field)* lehet *(extrém esetben akár minden rekordnak más és más)*, ezek a mezők a táblázat oszlopai.
- Field objektum, Fields kollekció: Egy rekord *(vagy Recordset)* egy mezője. A Field objektumokat általában a Fields kollekció segítségével érjük el.
- Record objektum: egy Recordset egy sora, vagy akár egy különálló rekord. A Record objektum is mezőkkel *(Fields)* rendelkezik.
- Command objektum: parancsok, beépített eljárások, lekérdezések futtatására használható objektum. Lényegesek a hívott eljárás bemenő paraméterei *(Parameters kollekció, Parameter objektumok)*; eredményként általában *(bár nem feltétlenül)* Recordset-et kapunk vissza.
- Stream objektum: régi ismerős; bináris és szöveges folyamok, adatok kezelésére, mozgatására használható objektum.
- Property objektum, Properties kollekció: az objektumoknak sok olyan jellemzőjük van, ami közvetlenül nincs kivezetve. Ezeket a jellemzőket név szerint, az adott objektum Properties kollekciójában található Property objektumokon keresztül érhetjük el.
- Error objektum, Errors kollekció: az adatkezelés közben felmerült hibákat tartalmazza.

ADO konstansok

Az ADO programozása során sokszor találkozunk konstansokkal. E konstansok értékének keresgélése sok-sok felesleges munkával jár. Szerencsére a Windows mind a VBScript-hez, mind a JScripthez tartalmaz megfelelő, beilleszthető fájlt, ami tartalmazza a konstansok definícióját. A fájlokat keressük a \Program Files\Common Files\System\ado mappában; a VBScript-hez használatos fájl az adovbs.inc, míg ha JScript-ben dolgozunk, az adojavas.inc-t másoljuk ki a munkamappába. Ha az ASP oldalak elején a következő paranccsal beillesztjük a megfelelő fájlt, a kódban már nyugodtan használhatjuk a konstansokat:

```
<!-- #INCLUDE FILE="adovbs.inc" -->
```

Ennek a megoldásnak az a hátránya, hogy használata esetén akaratlanul is megnöveljük az ASP oldalaink méretét. Ha ezt nem szeretnénk, a webalkalmazásunk global.asa fájljába típuskönyvtárként is felvehetjük az ADO-t:

```
<!-- METADATA TYPE="typelib" FILE="C:\Program
Files\Common Files\System\ado\msado15.dll" -->
```

Ezután a kódokban minden további nélkül használhatjuk az ADO konstansait. A példaprogramokban az egyszerűség kedvéért mindig a konstansok valódi értékét használjuk, és megjegyzésként feltüntetjük a konstans nevét.

Kapcsolódás az adatbázishoz

A cikk során az aspsuli.mdb Access *(Jet)* adatbázist használjuk, ami letölthető az **[1]** címről. Ez az adatbázis az SQL Server-ben jól ismert Northwind adatbázis néhány érdekes tábláját tartalmazza. Aki rendelkezik SQL Server-rel, az a kapcsolódási fájlokat átállítva akár közvetlenül az SQL Server-ben is dolgozhat. A példaprogramok feltételezik, hogy az adatbázis- és konfigurációs fájlok a C:\db\ könyvtár alatt találhatók. A példaprogramok letöltési helyén lévő db könyvtárban található fájlokat ide másoljuk majd be.

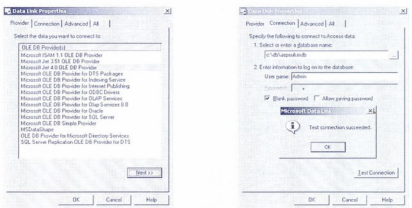
Az adatbázisok eléréséhez magának az adatbázisnak nem kell *(sőt, nem is szabad)* a webes területen lennie. Az adatbázishoz megfelelő jogosultságokat kell definiálnunk *(ha pl. az aspsuli.mdb-hez az anonymous internetes felhasználónak nincs írásjoga, nem fogja tudni módosítani a tartalmát, sem pedig új adatokat felvenni bele)*; ezeknek a hozzáférési jogosultságoknak viszont semmi keresnivalójuk olyan helyen, ahol a webes ügyfél közvetlenül elérhetné őket.

Ha már tudjuk, milyen adatbázishoz szeretnénk kapcsolódni, az ADO-nak háromféle módon adhatjuk meg:

- Az úgynevezett Connection String segítségével. A Connection String egy szöveges változó, amely minden, a kapcsolódáshoz szükséges információt *(adatbázis helye, felhasználónév, jelszó, stb.)* magában hordoz.
- Data Link File segítségével. A Data Link File tulajdonképpen egy fájlba mentett Connection String, azonban van két nagyon nagy előnye: egyrészt, a kódba nem kell magát a Connection Stringet beleírni, csak a Data Link File nevét; így a működő rendszer alatt egyetlen mozdulattal, a fájl átírásával kicserélhetjük az adatbázist. Másrészt, a Data Link File ügyes konfigurációs varázslót tartalmaz, így nem nekünk kell a Connection String előállításával és szintaxisával bajlódnunk.
- ODBC adatforrások, DSN-ek segítségével. Ha az ODBC Manager-ben definiáltuk a megfelelő System Data Source Name-t (DNS-t), az ADO-nak elég azt átadnunk a kapcsolódáshoz.

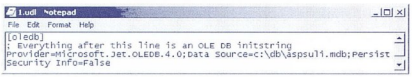
A Data Link File

A Data Link File – mint mondtam – nem más, mint egy szövegfájlba mentett Connection String. Előnye viszont, hogy a Connection String-et nem kézzel kell előállítani. Próbáljuk ki: hozzunk létre egy üres szövegfájlt, majd nevezzük át .udl kiterjesztésűre! Az ikonja megváltozik, és ha kettőt kattintunk rá, megjelenik a Data Link Properties dialógus:



A Data Link dialógus

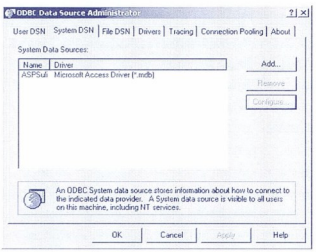
A Provider oldalon kiválaszthatjuk, hogy melyik OLE DB providert szeretnénk a kapcsolathoz használni. Ezután a Next gombra kattintva megjelenik a Connection fül, aminek tartalma az előzőleg választott provider típusától függ *(hiszen providerenként más és más adatokat kell megadnunk)*. A példaprogramban a Jet 4.0 *(Access)* provider-t választottuk, majd a következő oldalon megadtuk a használni kívánt adatbázis nevét *(c:\db\aspsuli.mdb)*. Ha a Test Connection gombra kattintunk, az eszköz megpróbálja a megadott adatok alapján felépíteni a kapcsolatot, és értesít az eredményről.
Ne felejtsük el, hogy a Data Link fájlt felhasználóként hozzuk létre. Amikor az adatbázist ASP oldalból szeretnénk használni, az anonymous Internetfelhasználó nevében tevékenykedünk – előfordulhat, hogy a Data Link létrehozásakor minden jónak tűnik, és ASP alól mégsem működik. Ilyenkor valószínűleg arról van szó, hogy az anonymous Internet felhasználó (IUSR_<gépnév> illetve most már inkább a Web Anonymous Users csoport) nem rendelkezik a megfelelő jogosultságokkal az adatbázis eléréséhez.
Nyissuk meg NotePaddel az előbb létrehozott .udf fájlt:



A Data Link File csak egy szövegfájlba mentett Connection String (a szöveg kijelölt része)

ODBC kapcsolat létrehozása

Ha ODBC kapcsolatot szeretnénk használni, úgynevezett System Data Source Name-et *(System DNS-t)* kell létrehoznunk. Ehhez a vezérlőpulthan keressük meg a Data Sources *(ODBC)* ikont *(az Administrative Tools alatt bujkál)*, és indítsuk el az eszközt:



Ha ODBC kapcsolatot szeretnénk használni, System DNS-t hozzunk létre

Az Add... gombra kattintva megjelenő varázslóban válasszuk ki a használni kívánt driver-t *(a fenti példában a Microsoft Access Driver-t)*, majd a Finish-re kattintva töltsük ki a további mezőket. A Data Source Name mezőben adjunk a kapcsolatnak egy egyedi nevet *(ezzel fogunk hivatkozni rá)*, majd válasszuk ki, melyik adatbázist szeretnénk használni.

A Connection objektum

Bár az adatbázis-kapcsolatot nem feltétlenül kell a Connection objektum segítségével előre létrehozni *(egy-egy Recordset megnyitásához vagy Command objektum használatához az ADO képes azt menet közben is létrehozni)*, mindenképpen hatékonyabb, ha előre megtesszük azt. Első lépésként hozzuk létre a Connection objektumot *(conn1.asp)*:

```
Set oConn =
  Server.CreateObject("ADODB.Connection")
```

Az objektum létrehozása után első dolgunk az adatbázis-kapcsolat megnyitása lesz. Ehhez az objektum .Open() metódusát használhatjuk. A metódus első paramétere határozza meg a megnyitni kívánt kapcsolatot, és lehet:

- egy Data Link File neve *(„File Name=" előtag után)*:

```
oConn.Open "File Name=C:\db\aspsuli.udl"
```

- ODBC System DSN neve („DSN=" előtag után):

```
oConn.Open "DSN=AspSuli"
```

- ... és persze lehet maga a komplett Connection String is, például:

```
oConn.Open "Provider=Microsoft.Jet.OLEDB.4.0;
  Data Source=C:\db\aspsuli.mdb;
  Persist Security Info=False"
```

Az .Open() metódus ezen kívül még három paramétert kaphat, ezek megadása azonban nem kötelező. Sorrendben:

- UserID – a kapcsolathoz használt felhasználónév, ha itt megadjuk, az felülbírálja a Connection String-ben definiált értéket
- Password – a kapcsolathoz használt jelszó, ha itt megadjuk, az felülbírálja a Connection String-ben definiált értéket
- Options – ha értékét adAsyncConnect-re *(16)* állítjuk, a kapcsolódás aszinkron menne végbe; a kapcsolódás befejezésekor pedig a ConnectComplete esemény következne be. Ezt a funkciót ASP-ben *(események hiányában)* azonban nem használhatjuk.

Ha a kapcsolat megnyitása nem sikerül, hibát kapunk vissza, amit kezelhetünk. Emellett bármikor ellenőrizhetjük a Connection objektum .State jellemzőjét is; értéke az alábbi konstansok valamelyike lehet:

Konstans	Érték	Jelentés
adStateClosed	0	A kapcsolat zárva
adStateOpen	1	A kapcsolat nyitva
adStateConnecting	2	Kapcsolódás folyamatban
adStateExecuting	4	Parancs végrehajtása folyamatban
adStateFetching	8	Eredmények visszaadása folyamatban

A kapcsolat lezárása

A munka befejeztekor a Connection objektum .Close() metódusának meghívásakor zárjuk le a kapcsolatot. A valóságban a kapcsolat azonban nem szakad meg: mivel az adatbázishoz való kapcsolódás az egyik leglassabb művelet, az OLE DB bizonyos ideig tárolja a kapcsolatokat, hátha ezidő alatt érkezik egy újabb kérés, ami pontosan ugyanazt az adatbázist, pontosan ugyanolyan paraméterekkel

szeretné elérni *(ez egyébként elég gyakori eset)*. Ilyenkor az OLE DB nem épít fel új kapcsolatot, hanem a meglévő, korábban már lezárt példányt használja fel újra. Fontos tudni viszont, hogy az újrafelhasználás nem egyenlő a párhuzamos felhasználással, tehát az OLE DB egy kapcsolatot addig nem oszt ki másnak, míg az nyitva van – helyette kénytelen újat létrehozni. Éppen ezért próbáljuk a Connection objektumot a lehető legkésőbb megnyitni, és a munka végeztével a lehető leghamarabb bezárni.

A Recordset objektum

Az adatokat legtöbbször az adatbázis egy részének táblázatos nézetén keresztül kezeljük. Ez a táblázatszerű adathalmaz nálunk egy Recordset objektum képében jelenik meg, valahogy így:

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit
1	Chai	1	1	10 boxes x 20 bags
2	Chang	1	1	24 - 12 oz bottles
3	Aniseed Syrup	1	2	12 - 550 ml bottles
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	Chef Anton's Gumbo Mix	2	2	36 boxes
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.
10	Ikura	4	8	12 - 200 ml jars
11	Queso Cabrales	5	4	1 kg pkg.
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.
13	Konbu	6	8	2 kg box

A Recordset objektum táblázatos formában

A Recordset tehát sok-sok rekordot *(sort)* tartalmaz, és minden rekord több mezőből *(oszlop)* áll. Az aktuális rekordot a kurzor jelzi *(az ábrán látható nyilat is felfoghatjuk kurzornak)*. A kurzort a Recordseten belül elmozdíthatjuk; a műveleteket *(írás-olvasás)* pedig az aktuális *(kurzor által mutatott)* rekordon végezzük.

A Recordset megnyitásakor definiálnunk kell, milyen kurzort szeretnénk majd használni. A kurzorokat két kategória szerint csoportosíthatjuk: a kurzor típusa és a kurzor helye szerint. Típus szerint az alábbiak lehetnek:

- Static *(adOpenStatic, 3)*: Statikus kurzor esetén a rekordokról másolat készül, és a Recordsetben már csak a másolaton dolgozunk. Az adatbázisban történt változások menet közben nem jelennek meg a Recordsetben, és az adatok nem is módosíthatók. A kurzort előre és visszafele is mozgathatjuk.
- Forward Only *(adOpenForwardOnly, 0)*: Az így létrehozott Recordset hasonló a statikushoz, azzal a különbséggel, hogy itt a kurzor csak előre mozoghat. Ez az alapértelmezett kurzortípus.
- Dynamic *(adOpenDynamic, 2)*: A dinamikus kurzorral létrehozott Recordsetben minden, a létrehozása óta végrehajtott változás is látható, a kurzort mindkét irányba mozgathatjuk. Ez a legdinamikusabb, és egyben „legköltségesebb" kurzortípus
- KeySet *(adOpenKeyset, 1)*: A keyset kurzor hasonló a dinamikus kurzorhoz, azzal a különbséggel, hogy az így létrehozott Recordsetben a többi felhasználó által a Recordset létrehozása óta törölt vagy létrehozott rekordok nem láthatók *(a meglévő rekordokban történt változások igen)*.

A fenti értékeket a Connection illetve a Recordset objektum .CursorType jellemzőjeként állíthatjuk be, illetve a Recordset objektum .Open() metódusának paramétereként adhatjuk át. Az alapértelmezés – mint már említettük – a forward-only kurzor; az adatok egyszeri legyűjtéséhez tökéletesen elég. Ugyanakkor, ha az adatokban módosítanunk kell, a forward-only kurzor már kevés.

A kurzorok másik fontos tulajdonsága a kurzor helye. Nevezetesen, hogy a kurzort a kiszolgáló vagy pedig az ügyfél kezeli-e. Alapértelmezés szerint *(és ha azt az adatbáziskezelő provider-e is támogatja)*, a .CursorLocation jellemző értéke adUseServer *(2)*, azaz a kurzor kiszolgálóoldali. Ha azt szeretnénk, hogy a teljes Recordset az ügyfélnél tárolódjon – vigyázzunk, ennek adatforgalmi folyományai vannak –, a kurzort a klienshez küldhetjük az adUseClient *(3)* értékkel.

Mielőtt végre elkezdenénk használni a Recordsetet, még egy dolgot ki kell tárgyalni, az pedig a rekordok zárolása *(Locking)*. Ahhoz, hogy ugyanazt az adatot két felhasználó egyidőben ne tudja módosítani, a módosítás idejére zárolni kell. Ez a zároló mechanizmus természetesen létezik az OLE DB-ben, de többféle módon is működhet, attól függően, hogy a .LockType jellemzőnek milyen értéket adunk:

- adLockReadOnly *(1)* – csak olvasható Recordset, így biztosan nem lesz szükség a rekordok zárolására. Ez az alapértelmezés, tehát ha az adatokat módosítani szeretnénk, a zárolás típusát meg kell változtatnunk.
- adLockPessimistic *(2)* – pesszimista zárolás; a provider a rekord bármelyik mezőjének módosításakor zárolja a rekordot
- adLockOptimistic *(3)* – optimista zárolás; a provider csak a módosítások érvényre juttatásakor (az .Update() metódus idejére) zárolja a rekordot
- adLockBatchOptimistic *(4)* – kötegelt optimista zárolás; a provider csak a változtatások kötegelt érvényesítésekor *(az .UpdateBatch() metódus alatt)* zárolja a rekordokat.

Recordset létrehozása, megnyitása

Recordsetet létrehozhatunk önmagában, majd csatlakoztathatjuk egy adatbáziskapcsolathoz *(vagy hagyhatjuk a memóriában, sokszor tökéletes tárolóeszköz)*; illetve Recordsethez jutunk akkor is, ha a Connection vagy a Command objektum .Execute() metódusát meghívjuk. Vesézzük ki először az első esetet: egy újonnan létrehozott Recordset objektumot hozzárendelünk egy már meglévő adatbáziskapcsolathoz, majd abból megnyitunk egy táblát *(rs1.asp)*:

```
Set oConn =
  Server.CreateObject("ADODB.Connection")
oConn.Open "File Name=C:\db\aspsuli.udl"

Set oRS = Server.CreateObject("ADODB.Recordset")
oRS.Open "Products", oConn, , , 2 'adCmdTable

While Not oRS.EOF
 Response.Write oRS("ProductName") & "<BR>"
 oRS.MoveNext
WEnd

oRS.Close
oConn.Close
```

A Recordset Open metódusának paraméterei a következők:

- A végrehajtandó parancs *(SQL kifejezés, tábla vagy tárolt eljárás neve, Command objektum, stb.)*
- Az aktív adatbáziskapcsolat Connection objektuma
- A használni kívánt kurzortípus *(ha nem adjuk meg akkor adOpenForwardOnly)*
- A használni kívánt zárolási stratégia *(ha nem adjuk meg, akkor adLockReadOnly)*
- Az első paraméterként átadott adat értelmezésére, valamint végrehajtására vonatkozó adatok, például:

Konstans	Érték	Jelentés: az első paraméter...
adCmdText	1	SQL kifejezés vagy tárolt eljárás neve
adCmdTable	2	Tábla neve
adCmdStoredProc	4	Tárolt eljárás neve
adCmdUnknown	8	Ismeretlen *(alapértelmezés; az ADO fogja eldönteni, hogyan értelmezi)*
adCmdFile	256	Fájlba mentett Recordset objektum fájlneve

A példában egy alapértelmezett, csak olvasható Recordsetet nyitottunk a Products táblára.

Navigálás a Recordsetben

Két fontos jellemző az elejére:

- .BOF – értéke True, ha a kurzor a Recordset első rekordja előtt áll
- .EOF – értéke True, ha a kurzor a Recordset utolsó rekordja után áll

Ha a Recordset .BOF és .EOF jellemzőinek értékei egyaránt True, a Recordset üres. A kurzor mozgatása:

- .MoveNext() – a kurzor a következő rekordra lép
- .MovePrevious() – a kurzor az előző rekordra lép
- .MoveFirst() – a kurzor az első rekordra lép
- .MoveLast() – a kurzor az utolsó rekordra lép

Lássuk csak mégegyszer az előző példa közepét:

```
While Not oRS.EOF
 Response.Write oRS("ProductName") & "<BR>"
 oRS.MoveNext
WEnd
```

Egy dolog nem ismert még: az aktuális rekord adott mezőjét – például – az oRS*(mezőnév)* hívással érhetjük el. Minden másnak ismerősnek kell lennie: egy ciklust formálunk mindaddig, amíg a Recordset kurzora az utolsó utáni pozícióra nem lép. A kurzor folyamatos mozgatását a cikluson belül kiadott .MoveNext() hívások biztosítják.

Tipikus hiba, hogy a cikluson belül elfelejtjük a .MoveNext() metódust meghívni. Az eredmény ilyenkor egy végtelen ciklus (hiszen a kurzor sosem fog a Recordset végéig elérni); rajta segíteni csak az adott webalkalmazás – rosszabb esetben az IIS szolgáltatás – újraindításával lehet!

A Fields kollekció

A Recordset mezőit a fenti közvetlen hivatkozásnál okosabban is elérhetjük, ha kihasználjuk a Fields kollekció illetve a Field objektum nyújtotta előnyöket. Az rs2.asp egy SQL kifejezést futtatva visszakapott Recordset mezőinek néhány adatát, majd a rekordok minden egyes mezőjét listázza nekünk.

```
For Each oField In oRS.Fields
  Response.Write "<BR>Nev: " & oField.Name
  Response.Write "<BR>Tipus: " & oField.Type
  Response.Write "<BR>Meret (definialt): " &
↳  oField.DefinedSize
  Response.Write "<BR>Meret (pillanatnyi): " &
↳  oField.ActualSize
  Response.Write "<BR>"
Next
```

A rekord mezőit jelképező Field objektumnak sok jellemzője van, a neve *(.Name)* és értéke *(.Value)* mellett a legfontosabb talán mégis a típus *(.Type)*. Ez lehet például *(a teljes listát lásd:* **[3]***)*:

Konstans	Érték	Jelentés
adSmallInt	2	Egész szám *(kétbájtos)*
adInteger	3	Egész szám *(négybájtos)*
adCurrency	6	Pénzbeli érték
adBoolean	11	Logikai érték *(igaz/hamis)*
adDate	7	Dátum
adVarWChar	202	Unicode szöveges változó
adLongVarWChar	203	Hosszú Unicode szöveges érték *(megjegyzés, note)*
adBinary	128	Bináris érték
adLongVarBinary	205	Hosszú bináris érték

A listcats.asp és a listemps.asp további lehetőségeket mutat be. Egyrészt, láthatjuk, hogy az adatok megjelenítésének módját függővé kell tenni az adatok típusától:

```
Select Case oField.Type
  Case 6: ' adCurrency (penz)
    Response.Write FormatCurrency(oField.Value )
  Case 7: ' adDate (datum)
    Response.Write FormatDateTime( oField.Value )
  Case 205: 'adLongVarBinary (kep)
    Response.Write "<IMG src=""photo.asp?
↳    t=Categories&f=Picture&" &
↳    "i=CategoryID&id=" & oRS("CategoryID") &
↳    """>"
  Case Else: Response.Write oField.Value
End Select
```

A pénzügyi adatokat a VBScript FormatCurrency(), a dátumértékeket a FormatDateTime() függvénye segítségével formázzuk meg. A bináris adat ebben az adatbázisban egy fényképet jelent, ezért ennek a megjelenítéséhez létrehozunk egy speciális URL-t, ami a photo.asp oldalt meghívva átadja annak a használt tábla és mezők nevét, valamint annak azonosítóját. A photo.asp pedig az átvett értékek alapján kimenetén tiszta bináris adatot ad vissza, és ehhez segítségül hívja az ADODB.Stream objektumot is:

```
Set oSt = Server.CreateObject("ADODB.Stream")
oSt.Type = 1 'adTypeBinary
oSt.Open
```

```
oSt.Write oRS( sPhotoField )
oSt.Position = 78 ' A kep tartalma 78.-tol kezd.
Response.ContentType = "image/bmp"
Response.BinaryWrite oSt.Read
```

A példában megnyitunk egy Stream objektumot, majd kiolvassuk a fényképet tartalmazó mezőből a bináris adatot. Az adatbázis jellegzetessége, hogy ez a bináris adat egy 77 bájtos fejlécet is tartalmaz, ami nem tartozik a képhez, ezért nem kell visszaadni *(ezért pozícionáljuk a Stream objektumot a 78. bájtra)*. Ezután a Response objektum ContentType jellemzőjét *(a válasz MIME típusát)* beállítjuk „image/bmp"-re, majd binárisan elküldjük az ügyfélnek a Stream objektum tartalmát.

Adatok módosítása

Ha adatokat kiolvasni már tudunk, itt az ideje, hogy módosítsunk is. A példaadatbázis terméktáblájában *(Products)* a UnitPrice mező tartalmazza a termék egységárát. Hozzunk létre egy olyan oldalt, ami a 73-as azonosítójú *(ProductID)* termék *(egyébként vörös kaviár)* pillanatnyi egységárát 10%-kal megnöveli! A teljes kód a kaviar.asp példaprogramban található.

```
Set oConn =
↳ Server.CreateObject("ADODB.Connection")
oConn.Open "File Name=C:\db\aspsuli.udl"
Set oRS = Server.CreateObject("ADODB.Recordset")
oRS.Source = "SELECT * FROM Products
↳ WHERE ProductID=73"
oRS.ActiveConnection = oConn
oRS.CursorType = 2 ' adOpenDynamic
oRS.LockType = 3   ' adLockOptimistic
oRS.Open , , , , 1 ' adCmdText
Response.Write "Előtte: " &
↳ FormatCurrency( oRS.Fields("UnitPrice") )
oRS.Fields("UnitPrice") =
↳ oRS.Fields("UnitPrice") * 1.1
oRS.Update
oRS.Requery 1 ' adCmdText
Response.Write " -- Utána: " &
↳ FormatCurrency( oRS.Fields("UnitPrice") )
oRS.Close
oConn.Close
```

A példaprogram első részét is érdemes megfigyelni. Látható, hogy a Recordset megnyitása előtt a legtöbb jellemzőt egyedileg is beállíthatjuk, és az .Open() metódusnak csak az utolsó paraméterét kell átadnunk *(vagy azt sem)*. Figyeljük meg, hogy dinamikus kurzort nyitottunk, és optimista zárolási stratégiát választottunk. Ezután kiírjuk a termék aktuális árát. Itt most az adatokhoz nem közvetlen, hanem a Fields kollekción keresztül fértünk hozzá (ez a „szebb" írásmód).
Az adatok módosítása egyszerű: a kívánt mező*(k)* értékét írjuk felül, majd a módosítások érvényre juttatásához hívjuk meg a Recordset Update metódusát.
A változások elvileg azonnal megjelennek a mi Recordsetünkben is *(hiszen dinamikus kurzort használtunk)*, de a demonstráció kedvéért mi a .Requery() metódus segítségével újra lekérdezzük, frissítjük a Recordset tartalmát. A .Requery() metódus egyetlen paramétere az .Open()-nél is megadott lekérdezés típusa.

Rekordok hozzáadása és törlése

Meglévő rekordok adatait már tudjuk módosítani, most nézzük meg, hogyan lehet új rekordokat létrehozni, illetve meglévő rekordokat törölni. A regionadd.asp a régiók táblázatába (Region) felveszi Európát, 5-ös azonosítóval, a regiondel.asp pedig törli azt.
A regionadd.asp lényeges része:

```
oRS.AddNew
oRS.Fields("RegionID") = 5
oRS.Fields("RegionDescription") = "Europe"
oRS.Update
```

Az egyetlen újdonság *(amellett, hogy most nem SQL lekérdezés eredményét, hanem a komplett táblát nyitottuk meg)*, a Recordset .AddNew() metódusában van. Paraméterként átadhatnánk neki az újonnan létrehozott rekordhoz tartozó mezők listáját, illetve azok értékeit, de ha nem tesszük, az új rekord örökli a tábla mezőit. Az .AddNew() hívása után a kurzor az újonnan létrehozott rekordra áll, az új rekord mezőinek értékét pedig az előbb ismertetett módon állíthatjuk be. Figyeljük meg, hogy bárhol áll a kurzor, az új rekord mindig a tábla végén jelenik meg; valamint hogy a kód többszöri futtatásával újabb és újabb Európa sorok születnek *(mert az adatbázisban nincs kikötve, hogy a rekordoknak egyedieknek kell lenniük)*.
A regiondel.asp törli az *(első)* Európa sort:

```
oRS.Open "SELECT * FROM Region WHERE
↳ RegionID = 5", oConn, 2, 3, 1
  ' adOpenDynamic, adLockOptimistic, adCmdText
If oRS.BOF And oRS.EOF Then
  Response.Write "Nincs ilyen rekord.<BR>"
Else
  oRS.Delete
End If
oRS.Close
```

Lépésenként: lekérdezzük az összes 5-ös azonosítójú sort *(Európa)*. Ha van ilyen, meghívjuk a Recordset .Delete() metódusát. Végül, bezárjuk a Recordset-et.
A .Delete() metódust paraméter nélkül használva az aktuális rekordot törli *(az .Update() hívására nincs szükség)*. Ha ez így nem megfelelő, a metódus paramétereként meghatározhatjuk, hogy mely rekordokat szeretnénk törölni – erről *(és az ADO további elemeiről)* a következő számban lesz szó.

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net/download/asp/10
[2] http://msdn.microsoft.com/library/en-us/ado270/htm/mdmscadoapireference.asp
[3] http://msdn.microsoft.com/library/en-us/ado270/htm/mdcstdatatypeenum.asp

Windows automatikus kilépés

Windows automatikus kilépés

K: *Az lenne a kérdésem, meg lehet-e oldani NT, és W95 alatt hogy automatikusan kilogoljon a hálózatból, és visszamenjen a bejelentkező képernyőhöz, mintha a 'bezárja az összes programot, és belép más néven' -t választom a kilépésnél. Van erre valami program, amivel el lehet ezt érni? Gondoltam, hogy átnevezem .scr-nek, és beteszem képernyővédőnek.*

V: A kérdés arra az alapvető fontosságú biztonsági feladatra utal, hogy hogyan léptessük ki az inaktív felhasználókat, mielőtt más használná a jogosultságaikat.

A megoldást a Microsoft Windows NT Server 4.0 Resource Kit tartalmazza. A képernyővédő gondolat helyes elképzelés, valóban a reskitben megtalálható WinExit program: egy speciális képernyővédő. A képernyővédő meghatározott idő letelte után szabályosan kilépteti a bejelentkezett felhasználót. Egyéb tulajdonságai más képernyőkímélőkhöz hasonlóan a Vezérlőpult->Képernyő->Képernyőkímélő menüben állíthatóak be és tesztelhetőek.

Ha azokat a programokat is be kell zárni, melyek nem mentett adatokat tartalmaznak, be kell jelölni a „Force application termination" opciót.

Forrás : NetAcademia Windows 2000 Lista

MS Exchange: Törölt levelek visszaállítása

K: *Egy felhasználó törölte a levelei egy részét, ezt szeretném a jövőben elkerülni. A kérdésem a következő: Hogyan lehet beállítani az Exchange Servert úgy, hogy néhány napig a levelek visszaállíthatóak legyenek valahogy? (Láttam, hogy az Outlookban van egy „Recover deleted item" menü, de ez sajnos szürke).*

V: Ahhoz, hogy a véglegesen letörölt levelek egyáltalán visszaállíthatóak legyenek, szükség van az Exchange beépített funkciójára, mely a Deleted items *(törölt elemek)*, tehát az egyszer letörölt levelek végleges törlése után is lehetőséget biztosít meghatározott ideig a levelek visszaállítására.

Azt, hogy az Exchange mennyi ideig tegye visszaállíthatóvá a leveleket, a „Deleted Mail Message Retention Time" pont alatt lehet beállítani.

Ha egy meghatározott mailboxra vagy könyvtárra kell beállítani, akkor az Exchange Server adminisztrációs programjában az adott mailbox vagy a public folder megnyitásával kezdhetjük. Ki kell választani a Limits fület és ott a „Use This Value *(Days)*" mezőben a visszaállíthatóságot lehet beállítani napokban.

Ha egy private vagy public IS -re kell beállítani, akkor az Exchange Server adminisztrációs programjában Server containert alatt az adott private vagy a public IS kiválasztásával kezdhetjük. Ki kell választani a General fület és ott a „Deleted item retention time *(days)*" mezőben a visszaállíthatóságot lehet beállítani napokban.

A levelek nem minden esetben visszaállíthatóak a fenti védelem ellenére sem. Ha a leveleket a következő „HARD DELETE" módon töröljük, akkor nem maradnak meg:

- Törlés SHIFT+DEL-lel
- IMAP vagy más olyan kliens törölte a levelet, ami nem helyezi be a Deleted Items könyvtárba
- A törlést egy offline dolgozó felhasználó végezte, aki szinkronizálás előtt letörölte a levelet a Deleted Items-ből

Az alapbeállítás szerint csak a Deleted Items könyvtárra van aktiválva a „Deleted Item Recovery". Ha az Inbox, Outbox stb. mappákra is használni akarjuk, a következő Registry beállítás szükséges a munkaállomásokon:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Exchange\
  Client\Options
DumpsterAlwaysOn=1
```

Ez választ ad az eredeti kérdésre is, miért volt „szürke" a Deleted Items Recovery.

Forrás : NetAcademia Exchange Lista

XMLgessünk *(VII. rész)* Webszolgák és Weburak

Két számmal korábban láttuk, miért van szükség a SOAP-ra *(Simple Object Access Protocol)*: ha szeretnénk barátságban látni a különböző gyártóktól származó rendszereket, és új alapokra helyezni az Internetes alkalmazások kommunikációját. Választ kaptunk a miértre. Az előző számban megnéztünk egy konkrét SOAP kiszolgáló- és ügyfélimplementációt, a Microsoft SOAP Toolkit-re építve *(az COM alapú volt, emberek!)*. Láttunk egyféle hogyant. Most pedig beletekintünk a jövőbe, és megnézzük, hogy mire jó ez az egész SOAP mizéria *(ezúttal szigorúan .NET alapokon)*. Hogyan tudja egy hétköznapi alkalmazásfejlesztő felhasználni az új technológiákat? Mi az a Webszolga? *(Webszolga: a Webservice magyarításával jó sokat küzdöttünk. A szerviz nem jó szó, mert oda az autónkat visszük. A service, szolgálat, szolgasors, szervusz tő kézenfekvő félreforditásával jött a Webszolga szó. Bocsánat érte.)* Hogyan kell írni és használni? A válasz itt lapul a következő oldalakon.

Mi az a Webszolga?

A Webszolga nem más, mint olyan program, amely más programok számára nyújt szolgáltatásokat szabványos internetes protokollokon keresztül. Egy aktív weblap, például a www.netacademia.net emberek számára nyújt szolgáltatásokat, információkat. Don Box szavaival élve, ha a fogyasztó szénalapú *(ember, kutya)*, akkor az nem Webszolga, viszont ha az információk fogyasztója sziliciumalapú *(számítógép program)*, akkor az egy Webszolga.

A honlapok kiszolgálója HTTP szerver szokott lenni, ennyiben áll a hasonlóság, mert a Webszolgák mögött is legtöbbször egy felokosított webszerver áll. Ennek ellenére a Webszolga nem egy honlap! A webszerver és a HTTP protokoll használata csak azért kézenfekvő, mert mindkettő a mai Internet gerincét képezi, és a legtöbb helyen a tűzfalak kíméletesen bánnak a HTTP csomagokkal - átengedik őket. Emellett az előző részekben már megnéztük, hogy a HTTP protokollra könnyen ráültethetünk valamilyen paraméterkódolási réteget, így elég egyszerűen megvalósítható egy távoli funkció meghívása, azaz egy Webszolga felépítése.

Eddig olyan jól *(?)* megvoltunk a COM komponenseinkkel, amelyek bizonyos szolgáltatásokat nyújtottak az alkalmazásaink számára. Mégis,

Mi változott?

A világ. Egyre több olyan szolgáltatás működik és lesz a világban, amelynek a belső logikája számunkra érdektelen vagy nem nyilvános, de szükségünk lesz rá az alkalmazásunkban. Például írunk egy üzleti webalkalmazást, ami bankkártyás megrendeléseket vesz fel. Érdemes lenne minél korábban megtudni, hogy az adott bankkártya érvényes-e, létezik-e, satöbbi. Ez nem megy saját kútfőből, kell egy külső szervezet, akinek a rendelkezésére állnak ezek az infók, és hajlandó azokat könnyen elérhetően a rendelkezésünkre bocsátani. Készítünk egy alkalmazást, ami egy listában megjeleníti a magyar névnapokat és azok dátumait. Honnan vesszük a listát? Letöltjük az Internetről, begyűrjük egy adatbázisba, és onnan listázzuk ki az alkalmazás számára? És ki fogja frissíteni az adatbázist? Mi van, ha a Dzsoki szép „magyar" név mellé felveszik a Dzsonit is április elsejei névnappal *(már van gazdája)*? Kellene valami módszer, hogy a központi nyilvántartó *(anyakönyvi hivatal? Nem tudom.)* könnyen felhasználható módon publikálhatná az adatokat, így az azt felhasználó alkalmazások például hetente frissíthetnék az adatbázisukat.

Szeretnénk minden felhasználó asztalán *(desktop)* kijelezni a holnapi időjárást. Kivágjuk a megfelelő részt valamelyik időjárásjelentő honlapból? Lehetséges, csak a sok rizsa közül nehéz lesz kiszedni azt a pár sornyi értékes infót.

Miért adnak vissza sokszor badarságot a webes keresők? Mert ők is eltévednek a púderben. A honlapokat embereknek tervezték, akik a sok mellébeszélés között észreveszik a villogó szekszreklámot, és rákattintanak, mert tudják, hogy ott lesz valami csemege. A természetes intelligencia segít megfejteni a weboldalak tartalmát, és mi akkor is megtaláljuk a kívánt információt, ha átstrukturálják a lapot, ezzel szemben a szemétdombot túrogató programok sokszor eltévednek.

Egy szónak is száz a vége: XML. Írjuk le a Webszolgánk által publikált szolgáltatásokat mindenféle felesleges körítés nélkül oly módon, hogy azt a fogyasztó programok is könnyedén, szisztematikusan tudják értelmezni. Nem emberek, programok. Az embereknek keretek kellenek, meg színes képek, de higgyék el, egy programot nem fogja lenyűgözni a Pléhboy *(ismerjük, az Özból)* címlapja. Ha egy olyan programot szeretnénk írni, amely csinos lányok képeit jeleníti meg a desktopon a fenti lap képarchívumából, akkor az a program sokkal jobban örül egy, a képet reprezentáló bájtfolyamnak, mint egy csinos HTML lapnak *(kinek a pap, kinek a paplan...)*.

A lényeg, hogy a gépek számára írt szolgáltatásokat más módon gondolkodva kell megközelíteni. Nem honlapokban, hanem metódusokban, nem linkekben és tartalomjegyzékekben, hanem interfészekben, szolgáltatásokban és végpontokban kell gondolkodnunk.

A mai világban a rövid fejlesztési időknek köszönhetően nem fogunk nekiállni írni például egy JPeg -> BMP kódolót, mert annyi ideig tartana, mint az egész projekt ideje. Nem írjuk meg, mert veszünk egy komponenst, amiben sok programozó nagyon sok munkával jól megírta azt, amit mi csak fáradságos munkával vagy sehogy nem tudtunk volna megtenni. Persze, pénzbe kerül, de például egy 1000$-os komponens *(~ 300e Ft)* árán maximum 3 napig lehet foglalkoztatni egy komoly fejlesztőt. Ennyi idő alatt eljut a diszkrét szinusz transzformáció megírásáig *(rosszabb esetben még meg sem értette :)*. De a JPeg kódolás még egy kicsit odébb van...

A Webszolgák világában szintén szolgáltatásokat veszünk, éppúgy, mint például egy COM komponens esetén. Csak míg az utóbbi esetben egy halott dolgot veszünk, aminek a való világról az az utolsó lenyomata, hogy kirángatták a prog-

ramozó körmei közül. Ezzel szemben a Webszolgák mögött állhatnak élő, pillanatról-pillanatra változó adatbázisok, mérési eredmények, valódi, élő adatok. Minden egyes újabb hívás más és más információkat adhat vissza. Azért ez egy picit több, mint egy COM, CORBA, ... komponens, nem?

A Webszolga technológia a Microsofté?

Nem. Meg az XML sem. Sokszor tesznek fel hasonló kérdéseket a hallgatók az XML tanfolyamokon. Az XML-ben benne van a Microsoft keze. Miben nincs? Ennek ellenére nem kisajátította azt, hanem más cégekkel együttműködve megpróbáltak megegyezni egy egységes szabványhalmazban. A megegyezési folyamatot a World Wide Web konzorcium vezényli le, ahol leül a sok haragos ember, mindenféle cégek képviselői és magánemberek, és megpróbálják kifürkészni a Világméretű Pókháló jövőbeli alakulását. Pontosabban ők a pókok, akik szövögetik a hálót, mindig újabb és újabb hálókat szőve.

Amint változik a világ igénye, úgy újabb és újabb protokollokkal, szabványokkal igyekeznek megtámogatni az elektronikus társadalmat. Újító lépés volt a HTTP protokoll kifejlesztése a 90-es évek elején. Azóta is jól állja a sarat, egyegy alkalmazás *(SOAP, WebDAV)* még további sok évre biztosítja a fennmaradását.

A Webszolgák esetén egyértelműen ott áll a háttérben a Microsoft. Ám *(a Microsoft is változik)* a Webszolga technológia nem egy zárt, céges szabvány, mint a COM vagy a JAVA nyelv. A specifikáció nem írja elő, hogy a háttérben Windows 2000 Servernek kell állni, sőt, egyre több Webszolga megvalósítás létezik például Linux és Apache alá. Bili bá nagy üzletet lát benne, ezért sok fejlesztő dolgozik azon Redmondban, hogy Microsoft.NET alatt minél egyszerűbben lehessen Webszolgákat fejleszteni. De ez nem azt jelenti, hogy a .NET lesz az egyetlen platform a Webszolga fejlesztésekhez, azonban fel kell kötni a gatyát a konkurenciának...

A gépek és a politika birodalma

Némi filozofálgatás után tekintsük meg a Webszolgák technikai oldalát, és ismerkedjünk meg a főszereplőkkel!

Gondolkodjunk el, milyen követelményeknek kell eleget tenni egy Webszolgákat támogató infrastruktúrának. Legalacsonyabb szinten kell egy olyan protokoll, ami támogatja a kérés-válasz alapú üzenetközvetítést. Erre tökéletes a HTTP, az okokról már korábban beszéltem.

Jó lenne erre egy olyan réteg, ami az üzenetközvetítőt meglovagolva képes távoli eljáráshívásokat támogatni, elintézve a paraméterek kódolását és hívások logikáját. Erre lett kitalálva új-régi ismerősünk, a SOAP.

A SOAP hívások előkészítéséhez szükség van egy olyan formális leírásra, ami leírja egy Webszolga által nyújtott szolgáltatásokat, azaz a rajta található metódusokat, azok paramétereit és visszatérési értékét. Ez kell a szolgáltatást igénybe venni kívánó fejlesztőknek, a fejlesztői környezeteknek. Lehetne word doksi, de mint mondtam, a Webszolga nem MS technológia. Valamilyen gyártófüggetlen megoldás kellene, amelyet könnyen értelmeznek a számítógépek, de a humánumnak sem törik bele a bicskája. Mi szólnának az XML-hez? A formális szolgáltatásleírást Web Services Description Language-dzsel *(WSDL)* valósítjuk meg. Ez egy XML fájl, ami pontosan definiálja a Webszolgán meghívható metódusokat és azok szerkezetét.

Egy kiszolgáló sokféle Webszolgát tartalmazhat, ezek leírására létrehozhatunk egyfajta tartalomjegyzéket, amelyet Discovery információnak hívunk. Például a szolgáltató gyökerébe rögtön felveszünk egy ilyen Discovery fájlt, ami leírja az összes általunk nyújtott Webszolgát.

Honnan tudom, hogy hol találok például egy időjárásjelentést szolgáltató Webszolgát? Cipészt vagy hastáncost a szakmai telefonkönyvekben keresünk. Kellene valami hasonló a Webszolgákhoz is. A Microsoft javaslata az elektronikus telefonkönyvre a UDDI szolgáltatás *(Universal Description, Discovery, and Integration)*. Ez nem más, mint egy nagy kereső, amiben WSDL fájlok laknak, és a szolgáltatást nyújtó cég mindegyikhez kitölti a potenciális ügyfelek számára fontos és érdekes információkat: az én Webszolgám ingyen kiszámítja a Nap és a Hold pillanatnyi távolságát. Az enyém hívásonként 1000 Ft-ért nem csinál semmit, de szeretnék gazdag lenni, ezért hívja meg mindenki, és adja meg a hitelkártya számát. Ezekből a plusz információkból megkeressük a számunkra szimpatikus szolgáltatást, és a visszakapott WSDL fájl alapján készen állunk arra, hogy nekilássunk azok használatához.

A WSDL még csak egy Note, azaz feljegyzés a World Wide Web konzorciumnál. Sajnos a gyártók nehezen akarnak megegyezni közös formátumban, egyik sem akar engedni. De előbb utóbb lesz egy közös WSDL nyelv. Addig is a fogyasztó alkalmazásoknak *(tipikusan fejlesztőeszközök)* fel kell készülni a lehető legtöbb WSDL nyelvjárás fogadására. De ez nem a mi gondunk.

A UDDI kérdés még nehezebb. A világ soha nem bízott meg a Microsoftban. Én egyik cégben sem bízom meg. Az UDDI indítvány – „Webszolga telefonkönyv" mögött az MS áll. Engem nem érdekel, hogy az MS mit csinál az általam publikált információkkal, hisz azokat azért teszem közzé, azért regisztrálom be hozzájuk, hogy minél többen láthassák, és így minél többen megvegyék a Webszolgámat. Valamilyen központi adatbázisra szükségünk lesz, amely nyílván elosztott lesz, és több cég kezében lesz. Hogy kinek a megoldása fog győzni? Ez a jövő zenéje.

Egy működő Webszolga

.NET-ben Webszolgát írni majdnem annyira egyszerű, mint egy közönséges osztályt fabrikálni. Vegyük az alábbi egyszerű osztályt:

```
using System;

public class MathService
{
    public float Add(float a, float b)
    {
        return a + b;
    }
}
```

Ez egy C# nyelven leírt osztály, melynek neve MathService, és egy metódusa van, Add, mely két float típusú szám összegét képezi. Ezt az osztályt már le lehet fordítani DLL-lé, és az így kapott ún. assembly-t már fel lehet használni más programokban. Lokálisan. Ott tartunk, ahol a COM tartott *(nem DCOM, COM)*. Publikáljuk ezt ki a nagyvilágnak, mint egy Webszolgát! Mi sem egyszerűbb:

```
using System;
using System.Web.Services;

public class MathService : WebService
{
    [WebMethod]
    public float Add(float a, float b)
    {
        return a + b;
    }
}
```

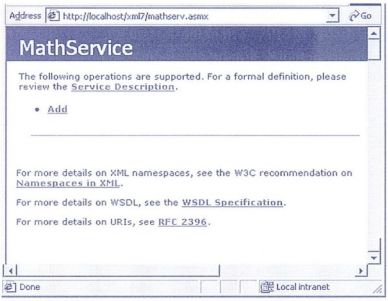
A dőlt betűvel jelölt részek a kiegészítések. A MathService osztályunkat a WebService nevű osztályból származtattuk örökléssel. Ezt jelzi a : a két osztálynév között. Ez azt jelenti, hogy az osztályunk tudni fog minden olyan szolgáltatást, amit a WebService osztály is tud. Persze, hogy ott van megírva az összes implementációs részlet. Az említett osztály a System.Web.Services névtérben található, ezért használtam a using System.Web.Services; sort. E nélkül is működne az alkalmazásunk, csak akkor mindig ki kellene írni a teljes nevet, ami kényelmetlen lenne:

```
public class MathService :
System.Web.Services.WebService
```

A Webszolgaként is hívható metódusainkat meg kell jelölnünk a [WebMethod] attribútummal. Alapban nem lesznek a metódusaink publikálva, hisz sokszor rengeteg metódust használunk egy osztályban, de csak néhányat szeretnénk a nyilvánosság rendelkezésére bocsátani.
Már csak kellene valaki, aki éjjel-nappal figyel, és várja, hogy valaki meghívja az Add metódusunkat. Ilyen feladatra kiváló egy webkiszolgáló, kivált, hogy a szervizünket HTTP protokollon szeretnénk elérni. Azért, hogy a webkiszolgáló tudja, hogy az előbb összeállított osztályt Webszolgaként akarjuk üzemeltetni, rakjuk azt bele egy .asmx kiterjesztésű állományba, és írjuk az elejébe az alábbi fejlécet:

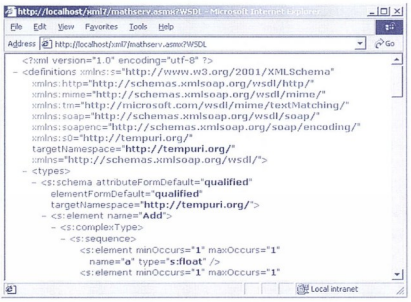
```
<%@ WebService Language="C#"
    Class="MathService" %>
//Innen jön az előbbi osztály definíciója.
using System;
...
```

Már csak egy lépés van hátra: másoljuk fel az asmx fájlunkat egy olyan IIS5 webkönyvtárba, amit mások is elérhetnek, és amire fel van telepítve a .NET Framework futtató környezet *(egy 18 MByte-os .exe a telepítő)*.



Egy Webszolga alapértelmezett arca

Nálam az xml7 virtuális könyvtár mögött található meg a Webszolga. Nézzünk rá böngészővel, mintha egy honlap lenne: A képről kivágtam a blablát, csak a lényeg van rajta. Az első link a szervizünk formális WSDL leírására navigál el, mindjárt megnézzük közelebbről. Alatta pöttyel megjelölve van kilistázva egyetlen metódusunk. Ha lenne több, akkor mind itt lennének felsorolva. Alul láthatóak linkek a World Wide Web konzorcium honlapjára, a különböző felhasznált technológiákra. Hangsúlyozom, hogy a linkek nem a Microsoft címére mutatnak, hanem a w3c honlapjára.
Lássuk, mi tárul fel az első link *(Service Description)* mögött!



A WSDL fájl tartalma

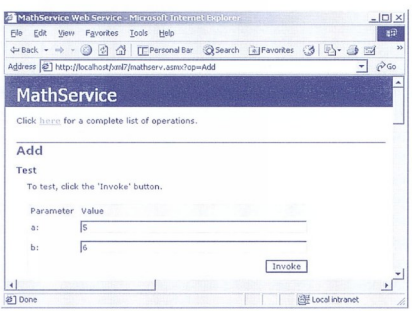
Egy WSDL dokumentumot kapunk vissza, amely a Webszolga URL-ünk WSDL paraméterezésével generálódott. Honnan ez a szép leírás? Nos, az ASP.NET futtató fogja a publikált osztályunkat, és az ún. Reflection képességek felhasználásával felderíti annak szerkezetét. Hasonló ez ahhoz, ahogy egy program fel tudja deríteni egy COM Type Library-ből a COM komponens szerkezetét, csak .NET-ben sokkal részletesebb leírás készül a típusokról. A felderített szerkezetből pedig generál egy WSDL fájlt, ahogy az a fenti képen is látszik. A .NET rengeteg helyen kihasználja a gazdag típusleírásból fakadó Reflection lehetőségeket, egy nagycsomó kézi faragástól kímélve meg minket.
Ragadjunk ki egy részletet a leírásból:

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
... >
<types>
  <s:schema ...>

    <s:element name="Add">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1"
              name="a" type="s:float" />
          <s:element minOccurs="1" maxOccurs="1"
              name="b" type="s:float" />
        </s:sequence>
      </s:complexType>
    </s:element>

    <s:element name="AddResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1"
              name="AddResult" type="s:float" />
        </s:sequence>
      </s:complexType>
    </s:element>

  </s:schema>
</types>
```

Látható, hogy a szolgáltatás hívását az Add nevű XML elem írja le, látható a két összeadandó paraméter is, amelyek típusa float. Szemfülesebbek észrevehetik, hogy ez az XML darabka nem más, mint egy XML Schema dokumentum. Miért találjanak fel valamit, ami már ki van találva? Az AddResponse a hívásra adott választ formalizálja, amiben csak egy float visszatérési érték utazik.

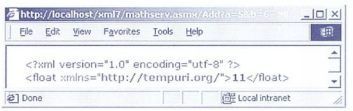
A másik oldal

Kétféleképpen is munkára bírhatjuk Webszolgánkat. A leíró lapon *(kettővel ezelőtti kép)* az Add linkre kattintva kapunk egy tesztlapot, amely segítségével tesztelhetjük a metódusainkat.



Még ki is próbálhatjuk a Webszolgát programozás nélkül!

Beírjuk az összeadandó paramétereket, és az Invoke gomb megnyomására meghívódik a szolga Add metódusa.
Mit kapunk válaszul? Természetesen XML tartalmat:



5+6=11. Remek. A válasz névterét természetesen be lehet állítani a szervizben, mi most az alapértelmezett tempuri.org-ot használjuk.
Hogyan utaztak a paraméterek a szolgához? Látható, hogy a metódus neve után ott van a két összeadandó hagyományos URL kódolással. Mivel a paraméterek az URL-ben utaznak, rájöhetünk, hogy a háttérben egy HTTP GET metódushívás történt. Hol itt a SOAP? Sehol. Bár meg lehet hívni így is Webszolgákat, nem ez az ajánlott módszer. Hogyan fogunk lekódolni például egy tömböt vagy egy összetett objektumot? Legyen ez a SOAP gondja. Igen ám, de kellene egy olyan osztály vagy komponens, amely képes olyan SOAP csomag összeállítására, amelyet a Webszolgánk közvetlenül képes fogyasztani.
A .NET keretrendszer számtalan SOAP-hoz kapcsolódó osztályt bocsát a rendelkezésünkre. A

```
System.Web.Services.Protocols.
  SoapHttpClientProtocol
```

például szinte mindent elintéz helyettünk, ha egy Webszolgát szeretnénk használni SOAP-on keresztül. Nekiállhatunk kézzel is az osztály használatának, azonban ez nem lenne túl produktív.

Jöjjenek a sémafordítók *(schema compiler)*!

A WSDL precíz leírást ad a Webszolgán elérhető metódusokról. A SOAP egyértelmű kódolási szabályokat ad a paraméterek XML-lé alakítására. A SoapHttpClientProtocol osztály képes SOAP csomagok előállítására, és a Webszolga által visszaküldött SOAP csomag értelmezésére. Ezek után már csak elő kell állítani egy olyan osztályt, ami a SoapHttpClientProtocol-ból származik örökléssel, és a segítségével egyszerűen meg lehet hívni Webszolgákat, egyszerűen úgy, mintha azok a saját gépen lakó objektumok lennének. Az ilyen átjátszó objektumokat általában Proxynak hívják.
Azaz egy olyan megoldás a célunk, ahol nem SOAP csomagokat dobálunk, hanem létrehozunk egy objektumot az ügyfélalkalmazásban, és annak a metódusait meghívva a funkcionalitás nem a saját gépünkön fut le, hanem egy másik gép Webszolgájában. Ehhez kellene egy olyan eljárás vagy program, ami a WSDL leírás alapján automatikusan legenerál egy SoapHttpClientProtocol-ra épülő proxyosztályt. Másképpen fogalmazva egy olyan program, ami átjárást biztosít a WSDL-ben található XML Schema és a programozott objektumok világa között. Ezeket a szoftvereket hívjuk sémafordítóknak.
A .NET keretrendszerrel kapunk egy WSDL.EXE-t, ami pontosan egy ilyen sémafordító. Paraméterül megadunk neki egy elérési utat (lehet fájlelérési út és URL is), ahol egy WSDL dokumentum található, és megadunk egy célnyelvet, amily nyelven fog létrejönni a proxyosztály. Generáljunk mondjuk egy C# nyelvű proxyt az előbbi Webszolgánk leírásából:

```
wsdl /language:c#
  http://localhost/xml7/mathserv.asmx?WSDL
```

Kimenetként kapunk egy MathService.cs C# állományt, Webszolgánk proxyosztályának forráskódját.
Kukkantsunk bele:

```
//
// This source code was auto-generated by wsdl,
// Version=1.0.2914.16.
//
...
public class MathService :
  System.Web.Services.Protocols.
  SoapHttpClientProtocol {
  public System.Single
  Add(System.Single a, System.Single b) {
    object[] results =
    this.Invoke("Add", new object[] {a, b});
    return ((System.Single)(results[0]));
  }
...
}
```

Látható, hogy a proxynak van Add metódusa, éppúgy, mint a Webszolgánknak, ráadásul pontosan olyan szerkezetben, mint ahogyan azt létrehoztuk. Azaz vár két lebegőpontos paramétert, és visszaad egy lebegőpontos eredményt.
Ez a metódus a Webszolga szinkron hívására való. Ez azt jelenti, hogy az Add metódus meghívása után addig nem kapjuk vissza a vezérlést, amíg az meg nem hívta a Webszolgát, és vissza nem kapta a választ. Mivel hálózatos és különösen internetes környezetben gyakori a több másodperces késleltetés, a szinkron hívások sokszor másodpercekre lefagyasztanák a felhasználói felületet, és az alkalmazásunkat *(joggal)* utálnák a felhasználók. A probléma kivédésére a proxy-ban megvannak az aszinkron híváshoz szükséges metódusok is.
Ez azt jelenti, hogy kapunk egy BeginAdd nevű metódust is, amelynek paraméterezése hasonló az Add-éhoz, csak van neki néhány egyéb paramétere is. Többek között megadhatunk egy objektumot plusz paraméterként, amelynek egyik metódusát visszahívja a keretrendszer, ha az aszinkron hívás eredménye megérkezett. Azaz egyszerűen meghívjuk a BeginAdd-ot, átadjuk neki az összeadandó paramétereket, és megadunk egy saját visszahívandó osztályt. A BeginAdd azonnal visszatér, de a háttérben *(egy másik szálon)* elindítja a Webszolga kommunikációt. Amikor az eredmények megérkeznek, meghívódik a visszahívandó osztályunk egy metódusa, és mi ebben például kiíratjuk az eredményt a felhasználónak. Ez a megoldás nyilván desktop ügyfélalkalmazásoknak jó, egy webalkalmazás nem nagyon tud profitálni az aszinkron lehetőségekből.
Az aszinkron hívások témaköre nagyon jól ki van dolgozva a .NET-ben, meg fog érni egy teljes cikket a tárgyalása.

Egyszerű ügyfélalkalmazás

Ezek után írjunk egy konzolalapú ügyfélalkalmazást, ami a generált proxyosztályon keresztül hívja meg a Webszolgánkat. A program nagyon egyszerű lesz. Létre kell hozni a proxyobjektumból egy példányt, és meg kell hívni az Add metódusát:

```
using System;

class WebszolgaTeszt
{
  public static void Main()
  {
    MathService m = new MathService();
    float r = m.Add(5, 6);
    Console.WriteLine(r);
  }
}
```

Ahhoz, hogy a hivatkozott MathService osztályt tudjuk használni, a WSDL.EXE által generált proxy osztályt le kell fordítanunk egy DLL assembly-vé *(mert abban van a MathService implementálva)*:

```
csc /target:library MathService.cs
```

Most, hogy megvan a proxy DLL, már csak a saját ügyfélalkalmazásunkat kell lefordítatni:

```
csc test.cs /r:mathservice.dll
```

A /r kapcsolóból tudja a fordító, hogy a MathService osztály a mathservice.dll-ben található.
Fordítás után miénk a test.exe, már csak futtatnunk kell:

Ott a válasz, 11, amit már a Webszolgánk hívásával számoltattunk ki. Ilyen pofonegyszerű használni az új Webszolga technológiát!
Ezt a példát és egy jóval összetettebbet is kipróbálhat és letölthet a kedves olvasó, ha ellátogat a **[1]** címre.

Zárszó

Ha már a nyakunkon a .NET, a következő számban a .NET XML lehetőségeivel fogunk foglalkozni, amelyek megismerése sokkal közelebb fog vinni bennünket a Webszolgák lelkivilágának megismeréséhez is. Amennyiben sikerült megmozgatnom a kedves olvasók fantáziáját, kérem ne habozzanak megírni nekem, milyen Webszolgákat látnának szívesen a NetAcademia webszerverén. A népszerű vagy praktikus ötleteket megvalósítom, és forrásukkal együtt kirakom a webünkre. Kíváncsian várom a javaslatokat!

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:
[1]: Letölthető kódok
http://technet.netacademia.net/download/xml

A titok nyitja

Az „e" előtaggal ellátott műszaki megoldások *(például e-mail, e-commerce, e-cash stb.)* közül egyesek gyakorlatilag teljesen, mások pedig egyre inkább elképzelhetetlenek a kriptográfia nélkül – részben a felhasználók személyes érdeke, részben pedig a szabályozás terjeszkedése miatt. A bankszámlaszámok, PIN kódok, jelszavak, fontos üzenetek titokban tartása nem igényel különösebb magyarázatot. A tipikus felhasználó azonban egy tekintetben megváltoztathatatlan: nem komálja a macerát. Ahhoz, hogy a kriptográfia eredményeit használja is, lehetőleg ne kelljen a megszokottnál többet kattintania. Hát nem? Hát de!

A kriptográfia alapja az algebra és az aritmetika. Az RSA *(Rivest-Shamir-Adleman)* algoritmus például két nagyon nagy prímszám szorzatán alapul: n=p*q. *(Ennél azért slightly bonyolultabb... Majd decemberben megírom – a szerk.)*

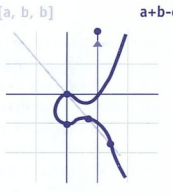
A Microsoft kutatói *(saját bevallásuk szerint)* naponta imádkoznak egy valódi egyirányú függvény létezésének bebizonyosodásáért. Ez érthető is: ha fellebbentik a fátylat a faktorizálás rejtélyéről, akkor a kriptográfiának, és ezzel együtt az összes, ezen alapuló üzletágnak befellegzett. Magától értetődően nem csak imádkoznak, hanem javítani próbálják a kódolás, a dekódolás és a titkosítókulcs előállításának sebességét, hatékonyságát és biztonságát. Ha megtalálják a titok nyitját – legyen ez köszönhető akár a Mindenhatónak, akár a logikának, – a kriptográfia nagy mérföldkőhöz, ha nem egyenesen a végállomásához érkezik.

Az elméleti kriptográfia célja az, hogy az X-en végrehajtott matematikai műveletek segítségével előállított Y-ból X-et ne lehessen egyértelműen, pontosabban egyáltalán ne lehessen meghatározni. Venkatesan kutató szerint a kriptográfia olyan rejtvények előállítása, amelyeket legalább az elkövetkező 20 évben nem fognak megoldani. Eddigi tapasztalatai alapján szkeptikus a végső megoldást illetően. Szerinte az egyirányúnak hitt algoritmusokról előbb vagy utóbb mindig kiderül az ellenkezője. Ettől való félelmében, Montgomery *(az osztást szorzásokkal helyettesítő, róla elnevezett algoritmus feltalálója)* a faktorizálás, mint fő veszélyforrás, különféle módszereivel próbálkozik. Egy 155 jegyű számot nemrégiben mindössze négy hónap alatt sikerült összetevőkre bontaniuk egy szobáraváló számítógéppel. A processzorok sebessége jelenleg olyan ütemben nő, hogy a mindössze szó használata az előbbi mondatban teljes mértékben tudatos volt. A kísérlet rámutatott, hogy a 155 számjegynek megfelelő 512 bit már kevés. Jöhet az 1024 bit és a 310 számjegy, ami az elődjénél hozzávetőlegesen egymilliószor nehezebbé teszi a feltörését. A számítógépek teljesítményének prognosztizálható növekedésén túl további veszélyek leselkednek ránk. Hadd említsek itt kettőt. A SETI-vel betört a köztudatba is az Interneten elosztott feladatvégrehajtás. Ez már olyan változatban is létezik, amikor a felhasználónak fogalma sincs arról, hogy segédkezet nyújt egy számításban. Hány szobáraváló gép dolgozik ilyenkor? Jó sok! A kriptológusok egy másik nagy félelme a kvantumszámítógép, amely forradalmasíthatja a számítástechnikát. A hagyományos gépek memóriája 0-k és 1-ek sorozata, továbbá a számoknak csak egy adott halmazán végezhetünk egyidejűleg műveleteket. A kvantumgépek memóriája ellenben egy kvantumállapot, amely több szám szuperpozíciójából jön létre. Képes párhuzamosan akár az összes számon matematikai műveleteket elvégezni, illetve a köztük lévő interferencia révén újabb eredményeket szolgáltatni. Emiatt aztán egy hasonló méretű, konvencionális felépítésű gépnél sokkal nagyobb számítási sebesség is elérhető. A megvalósítást célzó elképzelésekben nincs hiány – más kérdés, hogy a gyakorlat hol is tart valójában, és hogyan reagál az iparág egy ilyen innováció megjelenésére *(hadd passzoljam haza a labdát azzal, hogy dr. Watsonnak pár hónappal ezelőtt a Byte magazinban megjelent, burkolt célzásoktól sem mentes cikkére utaljak :-)*.

A kulcsok hosszának növelésével végülis a feltörés ideje könnyedén eljuttatható az irracionálitás határára, az online alkalmazások azonban ezt az időfonzat miatt egyáltalán nem tolerálják. Ön például mennyit lenne hajlandó egy internetes vásárlásnál arra várni, hogy a számlaszáma biztonságosan eljusson az eladó kiszolgálójára? Én annál is kevesebbet!

A modern számítógépek a nyilvános kulcsú titkosításoknál manapság használatos 128 jegyű számon a másodperc századrésze alatt elvégzik a szükséges műveleteket. Első ránézésre azt mondhatnánk, ez aztán a sebesség, de ha arra gondolunk, hogy mindez egy olyan kiszolgálón történik, amely percenként több ezer felhasználó kérését teljesíti, akkor az eredmény hirtelen más fényben tündököl. A gyakorlati kriptográfia célja ennek megfelelően a sebesség és a biztonság közötti, ésszerű kompromisszum megtalálása.

Montgomery, Lauter és Venkatesan egy ígéretes megoldáson is dolgoznak. A módszer az elliptikus görbéken alapul, amelyek egyszerű, köbös függvények. Az ábrán pl. az $y^2 = x^3 - 3x - 5$ elliptikus függvény látható, koordinátarendszerben ábrázolva. Az a és b pontokat összekötve egyértelműen megkapjuk c-t, de csupán c-ből az a-ra és a b-re nem következtethetünk. A lehetséges megoldások száma elméletileg végtelen, de a véges számábrázolás a megoldáshalmazt is korlátossá redukálja. Az ilyen egyenletek felhasználásával nyilvános kulcsú titkosítási rendszerek hozhatók létre. Ez a technika, ugyanakkora kulcsot feltételezve, biztonságosabb az RSA-nal és a Diffie-Hellman módszernél is, továbbá kevesebb időre van szükség a kódoláshoz és a dekódoláshoz. De talán még ezeknél az eredményeknél is fontosabb, hogy a jelenleg ismert algoritmusokkal még az RSA-nál is lassabban lehet feltörni.



A jelek szerint van még remény!

Zacco

VAN ZSÁKUNKBAN MINDEN JÓ...

Ünnepi akciók a SZÁMALK Továbbképzésnél

MCP vizsga a csizmában...

Szerezze meg Ön is a Microsoft termékspecialista (MCP) minősítést! Amennyiben **2001. december 6-ig** jelentkezik bármelyik Microsoft vizsgára hivatalos Prometric vizsgaközpontunkba, akkor most az MCP vizsgát 22.000 Ft helyett akciós, **5.000 Ft-os** áron veheti igénybe. (Az akció idején belül egy személy egy darab kedvezményes árú vizsgát vehet igénybe.)

Tanuljon karosszékből Microsoft Press kiadványok

Minden tisztelt vásárlónknak, aki legkésőbb **2001. december 17-ig** hivatalos Microsoft Press szakkönyveket rendel, **ajándékba adjuk** a magyar nyelvű *Windows Millennium Edition lépésről-lépésre* c. kiadványt. (Az akció a készlet erejéig tart)

Tanuljon jövőre is nálunk érdemes!

A SZÁMALK Továbbképzés a jövő évben is **változatlan áron** kínálja tanfolyamait. **Szerezzen eladható tudást!** Kedvezményes konstrukciójú *hivatalos Microsoft Windows 2000 rendszermérnök képzésünk* **2002. január 28-tól** veszi kezdetét. Ha még ebben az évben regisztrálja magát hivatalos Microsoft tanfolyamunk valamelyikére, **5.000 Ft értékű Microsoft Press könyvvásárlási utalványt** ajándékozunk Önnek.

Tanfolyami ízelítőnk:

2152 Implementing Windows 2000 Professional and Server	január 28 – február 1.
2272 Implementing Microsoft Windows XP Professional	január 28 – február 1.
2072 Administering Microsoft SQL Server 2000 Databases	február 25 – március 1
1572 Implementing Microsoft Exchange Server 2000	február 14–18.
2159 Implementing Microsoft ISA Server 2000	február 25–26.
2373 Visual Basic.NET for Developers	február 18–22.

További részleteket és információt akcióinkról, tanfolyamainkról internetes oldalunkon talál, vagy, kérjük, keresse telefonon szervezőinket.

Minőség, egyéneknek kedvező konstrukciók, cégeknek partneri kedvezmények!

Microsoft CERTIFIED Technical Education Center

SZÁMALK TOVÁBBKÉPZÉS

SZÁMALK

☎ 203-0304/3050 **www.szamalk.hu/tisza** 1115 Budapest, Etele út 68.