working with windows

tech.net

III. évf. 11. szám
2003 november
1964 Ft

ISSN 15865185
9 771586 518005 11

Szerkesztőség:
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net

A szerkesztőség címe:
1062 Budapest, Andrássy út 62.
Telefon: 472-1214
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti a
NetAcademia Kft.
Terjesztési, előfizetési információ:
Telefon: 472-1214
terjesztes@netacademia.net
Megjelenik havonta,
ára: 1.344 Ft

Előfizethető megrendelőlevélben a szerkesztőségnél:
1062 Budapest, Andrássy út 62.
Fax: 472-1215
http://technet.netacademia.net/subs

Hirdetésfelvétel: **Szívós Éva**
Telefon: 472-1214
Fax: 472-1215
info@netacademia.net

Nyomdai előkészítés:
Ars Luna Bt.
Vezető: Dobák Ildikó

Nyomda:
AduPrint Kiadó és Nyomda Kft.
1033 Budapest,
Csikós utca 8.
Felelős vezető: **Tóth Béláné**

ISSN 1586-5185

SZÖVETSÉG az elektronikus aláírásért

Október elsején a HÍF székházában megalakult a **Magyar Elektronikus Aláírási Szövetség**. A szervezet, melyet elektronikus aláírási szakértők és informatikai szakemberek alakítottak, nyitott minden, az elektronikus aláírás ügyéért tenni akaró informatikus, jogász és üzletember előtt.

A szövetség honlapja a

http://groups.yahoo.com/group/measz/ címen található. A hírlevélre feliratkozni a measz-subscribe@yahoogroups.com címre küldött elektronikus levéllel lehet.

Digitális aláírás vs. elektronikus aláírás

A digitális aláírás egyes végtelenül tudatlan egyének szerint megegyezik az elektronikus aláírással. Még jó hogy vannak magasan képzett, okleveles elektronikus aláírás szakértők *(nem digitális aláírás szakértők)* akik tudják, hogy ez a két dolog nem egy és ugyanaz, s így ha a végtelenül tudatlan egyén digitális aláírást találna mondani, kijavítja őt elektronikus aláírásra, ha pedig elektronikus aláírást találna mondani, kijavítja digitális aláírásra. Mígnem esetleg tudatlan egyénünk – aki előbb-utóbb meglehetősen frusztrálttá válik ezen okoskodástól – a sokadik korrektúra után visszakérdez, hogy ugyan már magyarázzák el neki, mi az a digitális aláírás és mi az az elektronikus aláírás, meg egyáltalán mi a kettő közt a különbség.

A nyilvántartásba vett elektronikus aláírás szakértő erre hebeg-habog valamit, vagy előáll valamilyen fél lábon álló teóriával, melynek következtében tudatlan egyénünk tudatlansága csöppet sem enyhül. E teóriákból én is hallottam már párat. Az első az volt, hogy a digitális aláírás a digitális adathalmazt feldolgozó és digitális eredményt adó algoritmusokkal összefüggésben értelmezett *(mint amilyenek a nyilvános kulcsú eljárások)*, míg az elektronikus aláírás egy általánosabb fogalom, ami értelmezhető bármilyen a jövőben megjelenő elektronikus eljárásra, mely a digitális aláíráshoz hasonló biztonságot garantál. Azáltal pedig, hogy az EU irányelv és a nemzeti jogszabályok az elektronikus aláírás fogalmát használják, egy általánosabb szabályozást biztosítanak. Ennek köszönhetően, ha – mondjuk – 2011-ben megjelenik egy, – teszem azt – analóg adatokat feldolgozó analóg elektronikus aláírási eljárás, akkor majd nem kell újraszabni a jogszabályokat. Nem tudom mennyi volt igaz ebből az okoskodásból, én mérsékelten hittem neki mindaddig, míg nem hallottam újabbat.

Aztán jött az, hogy az elektronikus aláírás fogalmába tulajdonképpen belefér az is, ha a word doksi végére gépelem a nevem. Vagy ha odaillesztem beszkennelt kézi aláírásomat. Na erre se gondoltam korábban, de tény, hogy ezek is elektronikus úton készülnek, így miért ne nevezhetnénk őket elektronikus aláírásnak? Még jó, hogy ott volt talonban a digitális aláírás fogalma, s így ezek után mondhattuk azt, hogy a digitális aláírás az az elektronikus aláírás, ami nem tartozik ebbe a vicc kategóriába.

Vagyis a nyilvános kulcsú kriptográfián alapuló elektronikus aláírás ebben az értelemben is digitális aláírás. Eme értelmezés nem sokban tér el az előzőtől: e szerint is mondhatjuk azt, hogy a digitális aláírás az elektronikus aláírás részhalmaza. Ugyanakkor e második értelmezés szerint ez a részhalmaz erősebb *(nagyobb biztonságot garantál)* mint az elektronikus aláírás nagy halmaza, míg az előző értelmezés szerint mindkettő ugyanolyan biztonságot garantált.

A jogi terminológia ugyanakkor a digitális aláírás fogalmát nem használja, ő a hitelesség, sértetlenség és a letagadhatatlanság követelményeinek nagy biztonsággal megfelelő elektronikus aláírást „fokozott biztonságú elektronikus aláírás"-nak nevezi.

Aztán szerencsére találkoztam ennél jobb elmélettel is. E szerint a digitális aláírás az a bitsorozat, ami kijön a nyilvános kulcsú algoritmusból. Ez önmagában nehezen értelmezhető: mellé kell rakni azt az információt, hogy milyen algoritmussal készült; azt hogy, kinek a nyilvános kulcsával ellenőrizhető; hogy ez a kulcs *(tanúsítvány)* hol található; és így tovább számtalan értelmezési és kiegészítő adatot. Ezekkel az adatokkal együtt lesz a digitális aláírásból elektronikus aláírás.

Nekem ez utóbbi magyarázat tetszik a legjobban, valószínűleg újdonságértékénél fogva is. Egyik teória sem nyert azonban eddig olyan teret, ami segítségünkre lehetne a miheztartásban. A különböző programokban tipikusan digitális aláírásnak van magyarítva az eredetiben digital signature kifejezés. Mivel e programok legtöbbjét az USA-ban készítik, nem is várhatunk mást.

A kifinomult európaiak ennél sokkal jobban tudnak disztingválni: az európai szoftverek, a jogszabályoknak, szabványoknak és ajánlásoknak megfelelően inkább az electronic signature kifejezést használják, míg a digital signature fogalmat a speciális esetekre tartják fenn. A jól megfigyelhető kulturális szakadék mellett vagy ellenére a szakemberek legtöbbször szinonimaként használják a két kifejezést. Magam is így teszek, s csak akkor figyelek oda ennél jobban, ha valóban meg akarok különböztetni két, valamiért különböző dolgot.

Az elektronikus aláírás feltétlen divatosabb, így annak, aki haladni akar a korral, ezt tudom javasolni. Konzervatívabb egyének maradhatnak nyugodtan a digitális aláírásnál is, ami egyfajta nemes tapasztalatot s ódivatúságot egyaránt sugallhat. Annak pedig, aki egyszerűen csak el szeretne igazodni, mindössze azt tudom javasolni, hogy figyeljen a szövegkörnyezetre *(és használjon azt, amit akar – a szerk.)*.

Almási János
Almasi-Janos@dbrt.hu

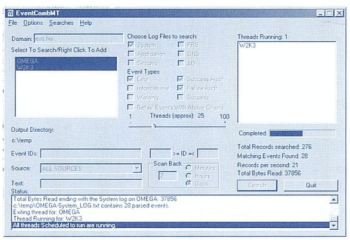
2003. 11. szám

Ne légy rendszergazda!

Secondary Logon Service

Mai „férges" világunkban már nem engedhetjük meg magunknak, hogy rendszergazdaként bejelentkezve tengessük mindennapjainkat. Hisz egy vírus, vagy egy féreg pontosan annyit tehet, mint a „hívója", vagy gazdaprocessze. Ha a „hívó" rendszergazdai jogokkal rendelkezik, akkor a gonosz kód is. Ez ellen találták ki a Secondary Logon Service-t és a RunAs-t. Megpróbáltam lehetőleg minden funkciót elvégezni RunAs-zel. Sikerült, bár elég mélyen bele kellett ásni szegény operációs rendszerbe! Kínomban még a NetSH parancsot is használtam.

4. oldal



Gyógyszeresdoboz+

Windows Server 2003 Resource Kit Tools

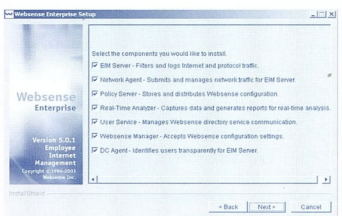
Van másik? Van, de azért ez egyelőre nagyon más*(ik)*. Per pillanat a „do more with less" jelmondat szó szerint érvényesül az új gyári segédprogram gyűjteményének tekintetében.

8. oldal

Websense – Az Internetfelügyelő

Az internet adta korlátlan szabadság néha a munka rovására mehet, sőt a cég számítógépein tárolt adatok is veszélybe kerülhetnek, ha ellenőrizetlen a vállalati webforgalom.

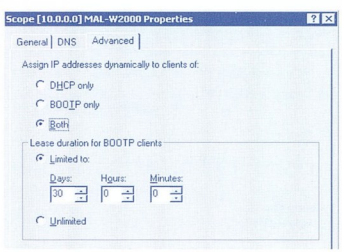
13. oldal



A DHCP rejtett szépségei – VI.

Azt már korábban is láthattuk, hogy a DHCP szolgáltatás nem áll önmagában, számos más komponenssel együttműködik. A DNS, az Active Directory, az RRAS és a DHCP viszonyát már tisztáztuk. A címkiosztó szerver azonban egyéb funkciókkal is szorosan integrált, sőt néha saját maga tartalmazza ezeket az alkalmazásokat. Ezúttal a BOOTP, a MADCAP, a RIS és a DHCP közti összefüggéseket vizsgáljuk.

16. oldal



Szolgáltatáskiesés – a százfejű szörny

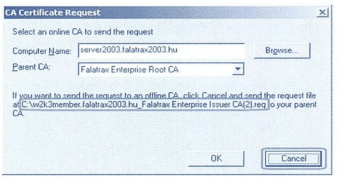
A Tech.net magazin szeptemberi számában egy nagyon érdekes cikket olvashattunk a Stratus cég ftServer megoldásáról, amely 99,999%-os rendelkezésre állást ígér. Sőt, nem csak ígéri, hanem be is tartja, hiszen a cég honlapján naponta újraszámolják a világon futó ftServerek összesített rendelkezésre állását, az pedig még ennél is jobb. Mindezt fürtszolgáltatás nélkül…

20. oldal

Tanúsítványkiadók a Windowsban

Qualified Subordination

A Qualified Subordination (kb. „minősített altanúsítványkiadók") fogalom egy új eszköz- és eljáráscsomag fedőneve a Windows Server 2003 Enterprise Edition nyílt kulcsú szolgáltatásai között. Segítségével a tanúsítványkiadók hierarchiájának ágaban szereplő altanúsítványkiadók működését minden eddiginél finomabban testreszabhatjuk.



23. oldal

Találd el a felhasználó igényeit!

Igényfelmérés, követelményanalízis

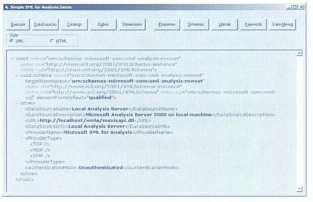
Hiába vagyunk a piac legnagyszerűbb fejlesztői, hiába találjuk ki a leghatékonyabb algoritmusokat, hiába dolgozunk a legmodernebb technológiákkal, ha egy dolgot nem tartunk szem előtt: a fő cél az ügyfelek elégedettsége.

27. oldal

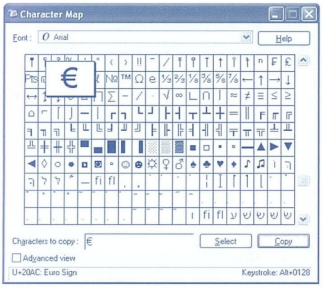
Kistestvére születik az ADO.NET-nek

XMLA és ADO MD .NET előzetes

Az MS SQL Server 2000 egyik újdonsága volt, hogy támogatta a relációs adatbázisaiban tárolt adatok elérését XML segítségével. Az ADO.NET-tel a relációs adatok és az XML fogalma még jobban összefonódott. De mi van az Analysis Serverben tárolt régi jó adatkockáinkkal és adatbányász modelljeinkkel? El tudjuk-e érni őket a .NET, vagy XML segítségével? Ezt fogjuk most megvizsgálni.



33. oldal



Reguláris kifejezések

Szövegfeldolgozás elmélet és gyakorlat regexekkel

A szövegekben keresés, egyes részek cseréje, kiemelése már az informatika kezdete óta foglalkoztatja a szakembereket. A regexek segítségével rendkívül kifejező módon tudunk részleteket megfogni egy hosszabb és nem szigorúan szabályos szövegből is. Ennek elméleti és gyakorlati kérdéseit boncolgatjuk a következő kilenc oldalban.

36. oldal

Ne légy rendszergazda!

Secondary Logon Service

Mai „férges" világunkban már nem engedhetjük meg magunknak, hogy rendszergazdaként bejelentkezve tengessük mindennapjainkat. Hisz egy vírus, vagy egy féreg pontosan annyit tehet, mint a „hívója", vagy gazdaprocessze. Ha a „hívó" rendszergazdai jogokkal rendelkezik, akkor a gonosz kód is. Ez ellen találták ki a Secondary Logon Service-t és a RunAs-t. Megpróbáltam lehetőleg minden funkciót elvégezni RunAs-zel. Sikerült, bár elég mélyen bele kellett ásni szegény operációs rendszerbe! Kínomban még a NetSH parancsot is használtam.

Október közepén készült az a féregvírus, amely a Windows Messengert támadja meg annak nyitott TCP portja mentén. Egy ügyes Buffer Overrun támadással a gonosz hacker saját kódját futtatja le a mit sem sejtő Messengerrel. A féreg kódja annak nevében fut, aki a Messengert futtatja.
Miért is van az, hogy egy gonosz kód *(vírus vagy féreg)* pontosan azokkal a jogosultságokkal fut, mint a gazdája? Ez a Windows processzindítási modelljének és az Access Tokennek köszönhető. Megfigyelték, hogy az egy és oszthatatlan WINWORD.EXE néha gyenge, mint a harmat *(ha Mari néni futtatja, nem tud beleírni a SYSTEM32 könyvtárba)*, máskor pedig olyan erős, hogy bármire képes *(a Rendszergazda által futtatva simán letörli a Windows bármelyik alkönyvtárát – ha abban nincs nyitott fájl)*.

Az Access Token

A különbség nyilvánvalóan nem a WINWORD.EXE-ben magában rejtezik, hanem az adott futtatás körülményeiben. Amikor bejelentkezünk a Windowsba, a rendszer hozzánk rendel egy leírótáblázatot, amiben a következő adatok vannak: Ki vagyok én? Mely csoportoknak vagyok tagja? Milyen rendszerszintű jogosultságaim vannak? Valahogy így:

Access Token
Személyes SID
Globális és univerzális csoport SID-ek
Lokális csoportok SID-je
Rendszerszintű jogosultságok

Az Access Token tartalma

A SID-ekről szerintem mindnyájan tudjuk, hogy miféle szerzetek, a rendszerszintű jogokra pedig mondok egy példát, hogy mindenki tudja, mire gondolok: Log On Locally.
A bejelentkezés után elindul az Explorer.EXE, és hozzárendelődik a mi Access Tokenünk. *(Ettől lesz az Explorer is pont olyan erős, mint mi magunk.)* Ezután valahányszor például fájlműveletet hajt végre az adott program, az operációs rendszer összeveti az Access Token tartalmát az adott objektum *(esetünkben fájl)* jogosultságlistájával *(Access Control List)*, ami felsorolja, hogy ki mit tehet az objektummal, valahogy így:

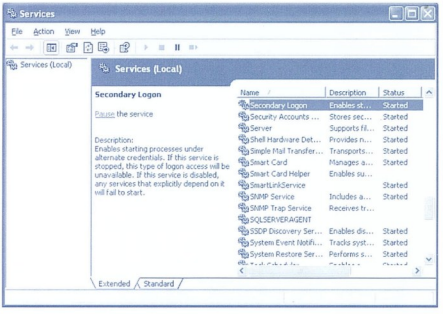
Access Control List	
SID 1	DENY Write
SID 2	DENY Full Control
SID 3	ALLOW Delete
SID 4	ALLOW Create

Egy jogosultséglista (ACL) belülről – leegyszerűsítve

Az Explorer.EXE akkor törölheti a kiszemelt fájlt, ha az Access Tokenben szerepel a SID 3 *(melynek tulajdonosa törlési joggal rendelkezik)*, de nem szerepel a SID 2 *(mert annak tulajdonosa teljesen el van tiltva a fájltól)*.

Az Acces Token öröklődése

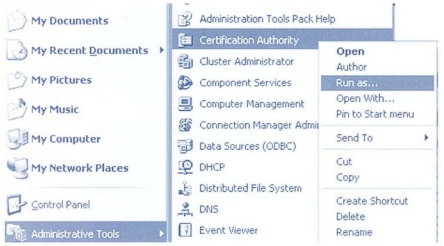
Később a felhasználó különböző alkalmazásokat indít, amelyek szintén az ő nevében futnak. Ennek az az oka, hogy a Start Menüből indított további alkalmazások gyermekei az Explorernek, ezért öröklik annak Access Tokenjét. Sőt, az örökség nemcsak apáról fiúra, hanem a teljes családfán végigfut. Ha bejelentkezés után indítunk egy Command Promptot, az gyermeke lesz az Explorer.EXE-nek. Ezután a parancssorból indítva egy programot, az a Command Promptnak gyermeke, az Explorernek pedig unokája lesz, de az öröklődés miatt ő is pontosan ugyanazzal az Access Tokennel fog rendelkezni, mint a nagyapja.
Ha a férgektől való félelmünkben sohasem jelentkezünk be rendszergazdaként, elvileg sohasem leszünk képesek „erős" alkalmazások indítására, ezáltal fontos funkciók ellátására sem. Hacsak az öröklődési lánc megbontására nincs valami módszer. De van. A Windows 2000 megjelenése óta lehetőség van arra, hogy bármilyen programot saját Access Tokennel bocsássunk útjára. Ezt a Secondary Logon Service nevű szolgáltatás teszi lehetővé.



A Secondary Logon Service teszi lehetővé az Access Token-váltást

Ez a szolgáltatás alapértelmezésben települ, és el is indul, így nincs más dolgunk, mint megkeresni a felhasználói felületen azokat az elemeket, amelyek révén használni is tudjuk. Ha

megcsiklandozzuk a rendszergazdai eszközöket az egér jobb gombjával a Start Menüben, némelyiken azonnal felbukkan a Run As... parancs. Vannak olyan eszközök is, amelyek viszont csak a SHIFT-jobbklikkre tárulkoznak fel. *(Nem sikerült megállapítanom, hogy mi a különbség oka. Van olyan MMC-konzol, ami sima jobbklikkre is RunAs-zel reagál, de van olyan is, amelyik nem.)* Az alábbi ábrán a Certification Authority eszközt próbálom más felhasználó nevében elindítani:



A Run As... menüponttal más felhasználó nevében indíthatunk alkalmazásokat

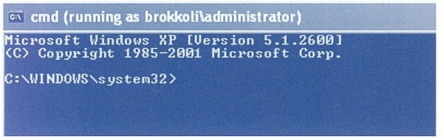
Az Administrative Tools menüpontban a létező rendszergazdai eszközöknek csak a töredékét találjuk meg, a többi nincs „előmelegítve", hanem be kell tölteni *(snap-in)* egy MMC konzolba. Ha szükségünk van egy „erős", ámde üres MMC konzolra, a parancssori RunAs segítségével indíthatunk egyet. Rögtön mutatom a szintaxist, de nem MMC-t fogok indítani, hanem egy Command Promptot. Hogy miért? Mert az további gyermekeket tud szülni, és rájuk hagyományozza a saját, megváltoztatott Access Tokenjét! Így ha további rendszergazdai parancsokat kell kiadnunk, ez a parancssor visszavár minket!

Az „erős" Command Prompt

Az alábbi parancs egy olyan CMD.EXE-t indít, amelyik az Administrator nevében, annak Access Tokenjével fut:

```
runas /user:administrator cmd
```

Az eredmény egy olyan parancsablak, aminek a tetejéről leolvasható, hogy kinek a nevében fut:



Az „erős" Command Prompt ablaka (lásd az ablak fejlécében!)

Ha ebből az ablakból „szülünk" mondjuk MMC.EXE-t vagy akár WinMine.EXE-t, az Administratori erősségű lesz.
Úgy tűnik, minden a kezünkben van ahhoz, hogy soha többé ne kelljen rendszergazdaként bejelentkeznünk.

Stílusváltás!

Tegyünk két fogadalmat:

1. Mostantól a napi rutinfeladatok végzésére NEM a rendszergazdai fiókunkat fogjuk használni!
2. Ha egy program csak akkor működik rendesen, ha rendszergazdaként futtatjuk, akkor SEM jelentkezünk be így, hanem
   a. kinyomozzuk, mi a hiba oka *(RegMon, FileMon)*
   b. csak az adott programot futtatjuk RunAs-zel.

Az első fogadalom betartását az alábbi Login Script is elősegíti, amely három másodperc után kijelentkezteti azt a felhasználót, akihez hozzárendeltük. Aki nem érez magában elég lelki erőt az 1. fogadalom betartásához, adja be magának intravénásan ezt a Logon Scriptet:

```
option explicit
dim oWMI, oWin, oSh
set oSh=CreateObject("wscript.shell")
set oWMI=GetObject("winmgmts:").ExecQuery("Select
* from WIN32_OperatingSystem")
for each oWin in oWMI
        oSh.PopUp "Ezzel a felhasználóval nem
        szabad bejelentkezni!", 3
        oWin.Shutdown 0
next
```

Ennek az azonnali logoff scriptnek egyébként elég jelentős irodalma alakult ki a NetAcademia Tudástárban *(www.netacademia.net/tudastar)*, például kiderült, hogy Terminal Sessionben nem jól működik, mivel üres marad az oWMI kollekció. *(Egyelőre nem tudni, miért...)*
Természetesen teljes stílusváltás csak akkor lehetséges, ha minden rendszergazdai funkciót el tudunk látni RunAs segítségével. A célom az, hogy ezt bebizonyítsam, még akkor is, ha néha teljesen lehetetlennek látszik a feladat.

Az Explorer indítása RunAs-zel

Ha fájlokat szeretnénk másolni, könyvtárakat létrehozni, vagy egyszerűen csak valamilyen dokumentumot megkeresni, a legkézenfekvőbb megoldás az Explorer.EXE használata. Nosza, elő a RunAs parancsot, vagy a korábban elindított „erős" Command Promptot, és indítsuk el!

```
Explorer.EXE
```

Se kép, se hang. Nem indul. Valószínűleg bugos, mert ha indítás előtt kilőjük a felhasználó által indított Explorert *(amitől persze leesik a képernyőről a Start menü is)*, akkor elindul.
Másképpen is lehet magyarázni az Explorer.EXE szabotázsakcióját: ha elindulna, gyakorlatilag feleslegessé tenné az eddigi óvintézkedéseinket, minthogy egy „erős" Explorer egyenértékű lenne azzal, mint ha rendszergazdaként jelentkeznénk be. Tehát nem indul el, bár egy hibaüzenettel azért vígasztalhatna minket.
Fáljok másolására ott vannak a belső DOS parancsok *(COPY, MOVE)*, vagy ha feltétlenül egérmozdulatokkal szeretnénk elvégezni a feladatot, elindíthatjuk az Internet Explorert is *(IExplore.EXE, a Program Files\Internet Explorer könyvtárban található)*, amit ha ráirányítunk a C:\ „URL"-re, veszi a lapot, és átalakul Explorerré *(csak az automatikus frissítést nem végzi el)*. Igen ám, de ha az Internet Explorert indítjuk rendszergazdai erősséggel, még nagyobb hibát követünk el, mintha a sima Explorert indítottuk volna, hiszen elég egy kis figyelmetlenség,

és máris egy „erős" böngészővel mászkálunk a neten. Az eljárás tehát nagy önfegyelmet igényel. A legjobb, ha lemondunk a Windows beépített grafikus fájlkezelőiről és valami más eszközt használunk fájlkezelésre.

Vezérlőpult hívása RunAs-zel

Talán a legfontosabb feladat, hogy a Vezérlőpultot el tudjuk indítani rendszergazdai erősséggel. Ha ez nem sikerül, akár fel is adhatjuk a küzdelmet, mert nem leszünk képesek elvégezni a legelemibb feladatokat sem *(pl. IP-cím átállítása)*.
A SYSTEM32 könyvtárban kutatva feltűnik a CONTROL.EXE, amit ha felhasználóként futtatunk, valóban felhozza a Vezérlőpultot, telistele sok szép ikonocskával. Ugyanakkor ha RunAs-zel, vagy éppen az „erős" Command Promptunkból próbáljuk indítani – se kép, se hang!
Ez is bugos? Nem, ő csak fél egyedül. Csak akkor hajlandó elindulni, ha vele együtt kézenfogva elindul egy Explorer.EXE is. *(Ennek az az oka, hogy a Vezérlőpult sem más, mint egy Explorer-mappa.)*
Korábban azonban láttuk, hogy *(akarva vagy akaratlanul)* olyan Explorer.EXE-nk van, ami mindaddig nem hajlandó elindulni a Secondary Logon Service hatóköre alatt, amíg elődjét, a felhasználó által indított Explorert meg nem öljük. Testvérgyilkosság.
Tehát a Vezérlőpult elindításának lépései:

1. Task Managerrel kilőjük az Explorer.EXE-t
2. CONTROL.EXE

Ez utóbbi lépés rendszergazdai erősséggel elindítja az Explorert *(lesz újra Start Menü stb.)*, ezzel gyakorlatilag romba dönti eddigi erőfeszítéseink eredményeit – viszont elindul a Vezérlőpult is.
Miután elvégeztük a kitűzött feladatot, ki kell lőni az erős Explorert, és el kell ismét indítani Mari néni Explorerét.
Ez így használhatatlan. És ha nem figyelünk oda, nem is működik. Ugyanis az „erős" explorert csak egy „erős" Task Manager tudja kilőni.

„Erős" Task Manager indítása

Ha a hagyományos módszerek valamelyikével indítunk Task Managert *(például CTRL+ALT+DEL, Feladatkezelő)*, egy „gyenge" példányhoz jutunk, ami csak azokat a feladatokat tudja bezárni és kilőni, amik maguk is gyengék.
Indítsuk el azonban az „erős" Command Promptból, és máris ki tudja lökni az „erős" Explorert!

```
Taskmgr.EXE
```

Egyedi Vezérlőpult ikonok futtatása

Ha a Vezérlőpult makacsul ragaszkodik egy „erős" Explorer jelenlétéhez, ne használjuk. Próbálkozzunk meg inkább az egyes Vezérlőpult-ikonok elindításával! Hiszen minden egyes ikon mögött egy önjáró, magában is futtatható programocska lakozik. *(Egészen pontosan fogalmazva: nem magában fut, hanem a RunDLL32.EXE futtatja őket.)* A Vezérlőpult programocskái a SYSTEM32 könyvtárban találhatók, .CPL kiterjesztéssel. Az alábbi táblázat összefoglalja, hogy melyik mire való:

CPL neve	Feladata
access.cpl	Kisegítő funkciók
appwiz.cpl	Programok telepítése varázsló
desk.cpl	Képernyőbeállítások
hdwwiz.cpl	Hardvertelepítő varázsló
inetcpl.cpl	Az Internet Explorer beállítópultja
intl.cpl	Területi beállítások
joy.cpl	Játékvezérlők
main.cpl	Egér és billentyűzet
mmsys.cpl	Multimédia
ncpa.cpl	Hálózati beállítások *(Explorer-alapú!)*
nusrmgr.cpl	Helyi felhasználók és csoportok
odbccp32.cpl	ODBC kapcsolatok
powercfg.cpl	Energiatakarékossági funkciók
slcpappl.cpl	Smart Link modemek
sysdm.cpl	A System ikon
telephon.cpl	Telefon és modem opciók
timedate.cpl	A rendszeróra beállítása

Ha például új programot szeretnénk telepíteni, írjuk be az „erős" Command Promptba, hogy

```
appwiz.cpl<Enter>
```

És máris előttünk a programtelepítő-varázsló. A billentyűzet és egér ikon érdekes jószág, ugyanis ha csak magában indítjuk el, a MAIN.CPL mindig az egérbeállításokat mutatja meg. A billentyűzet eléréséhez indítsuk így:

```
Main.cpl keyboard
```

Magyar Windows XP-n pedig így kell elindítani:

```
Main.cpl billentyűzet
```

És ezt most komolyan mondom. Kipróbáltam! A parancssori paramétere valóban „billentyűzet", hosszú ű-vel!

IP-cím átállítása

Akkor most lássuk a legfontosabb, leggyakrabban használt ikont, a hálózatokat! Szinte mindennapos feladat, hogy IP-címet, alapértelmezett átjárót, esetleg a DNS-kiszolgáló címét át kell állítanunk. Írjuk be az „erős" Command Promptba ezt:

```
Ncpa.cpl<Enter>
```

Majd várjuk a hatást! De a hatás elmarad. Se kép, se hang!
A rutinos RunAs-guru ilyenkor azonnal tudja, hogy itt megint az Explorer.EXE állja utunkat. És valóban! Ha kilőjük a felhasználó Explorerét, és kétszer *(sic!)* elindítjuk az ncpa.cpl-t, el is indulna rendben.
Hogy miért kell kétszer elindítani? Mert az első indításnál ezt a barátságtalan üzenetet kapjuk:

Kínlódunk az Explorer.EXE-vel

Ennek természetesen semmi értelme, mindössze azért vágják a fejünkhöz, mert elhagytuk a járt utat a járatlanért. De nem maga Bill Gates mondta, hogy ezentúl jobban ügyeljünk a biztonságos üzemeltetésre? Ilyenkor az az érzésem támad, hogy én vagyok e földgolyón a legelső személy, aki megfogadta Bill szavait. Még a saját programozói sem tették ezt!
Ami ennyire izzadságszagú, attól tartsuk távol magunkat. Hagyjuk a Vezérlőpultot, mert jól láthatóan rajtam kívül senki nem gondolja, hogy pont ezzel kellene IP-címet változtatni.

Hanem akkor hogyan?
Kézenfekvő elgondolásnak tűnik a Registry Editor használata, csak az a baj vele, hogy a regisztrációs adatbázisba közvetlenül belefirkált IP-címeket csak újraindítás után veszi figyelembe a rendszer. *(Elegendő lenne a hálózati kapcsolatot letiltani és újra engedélyezni, de akkor ugyanott vagyunk, mint az előbb: az ncpa.cpl használatára lenne szükség.)*
Nincs esetleg valami parancssori eszköz a hálózati kártya beállításainak babrálására? De van!

NetSH a megmentőnk

A NetSH parancs legalább olyan összetett és sokrétű, mint az NSLookup, és használni is hasonlóképpen két üzemmódban lehet:

- Ha paraméterek nélkül indítjuk, interaktív üzemmódban indul, parancssorába további speciális parancsokat lehet begépelni
- Ha a megfelelő paraméterekkel indítjuk, minden kérdezés nélkül lefut, és elvégzi a rábízott feladatot.

A számtalan alparancs és alprogram közül nekünk az INTERFACE alatti IP kontextus SET *(beállítási)* parancskészletére lesz szükségünk. Gépeljük be:

```
netsh interface ip set <Enter>
```

A következő beállítóparancsok léteznek:

```
Commands in this context:
set address - Sets the IP address or default
↳ gateway to the specified interface.
set dns - Sets DNS server mode and addresses.
set wins - Sets WINS server mode and addresses.
```

Tehát IP-címet, alapértelemzett átjárót, DNS- és WINS-kiszolgáló címet lehet vele beállítani. Kérdezzük csak meg tőle, hogyan kell például IP-címet változtatni:

```
netsh interface ip set address<Enter>
```

A több képernyőnyi válasz legalsó két sora két példát mutat a parancs használatára. Az első példa a Local Area Connection fedőnevű hálózati kártyát dinamikus IP-cím kérésére állítja be.

```
netsh interface ip set address
↳ name="Local Area Connection" source=dhcp
```

Egy picit túl van tupírozva ez a parancs, bőven elég lenne ennyi:

```
netsh interface ip set address Local dhcp
```

Ugyanis a NetSH töredékekből is felismeri, mire gondoltunk. Szinte már mesterséges intelligencia van benne *(nincs!)*, hisz a „Local" alapján kitalálta, hogy a Local Area Connection nevű kapcsolatra gondoltunk, míg a DHCP kulcsszóból arra következtetett, hogy DHCP-üzemmódot szeretnénk beállítani, nem pedig formázni a C:\ meghajtót, amire amúgy sem képes. Hogyan nézne ki az a parancs, ami a DNS-kiszolgáló címét is DHCP-forrásúra állítja? Így:

```
netsh interface ip set DNS Local dhcp
```

A második példa pedig kézi IP-címet állít be ugyanezen a kártyán:

```
netsh interface ip set address local static
↳ 10.0.0.9 255.0.0.0 10.0.0.1 1
```

Nem említettem, de a NetSH természetesen le is tudja kérdezni a jelenlegi állapotot a Show paranccsal:
netsh interface ip show address
A teljesség igényével felsorolom, mi mindent tud megmutatni a NetSH:

```
address, config, dns, icmp, interface, ipaddress,
ipnet, ipstats, joins, offload, tcpconn, tcpstats,
udpconn, udpstats, wins
```

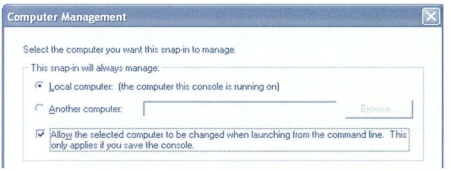
Ezek kivesézésére jelen cikkben nem térek ki.

MMC snap-inek

Még egy igen jelentős rendszergazdai „feladatgyűjtemény" van hátra, ez pedig az MMC snap-inek futtatása rendszergazdai jogkörrel.
Mi sem egyszerűbb: indítsunk egy „erős" MMC.EXE-t, és az összes snap-in erős lesz benne!
Mi a helyzet azonban akkor, ha egy távoli gépen kell „erős emberként" kotorásznunk az MMC snap-innel? Ez csak abban az esetben vezet sikerre, ha a másik gépen is rendszergazdák vagyunk. Ha az Administrator felhasználó neve és jelszava megegyezik a távoli gépen azzal, ami a helyi címtárban van, nincs probléma. Szintén nem ütközünk falakba, ha az „erős" Command Promptot egy tartományi rendszergazda nevében indítottuk, és a távoli gép is ugyanannak a tartománynak tagja, mint a mi számítógépünk.
Ha azonban ezek a feltételek nem teljesülnek, az MMC-konzol csendben „lepattan" a távoli gépről, új név+jelszó párost nem kér. Ismeretlen név+jelszó esetén viszont bolond lesz minket beengedni a távoli gép egy Disk Managerrel vagy hasonló célszerszámmal. Ilyen esetekben az alábbi módszert javasoljuk:

Építsünk ki rendszergazdai kapcsolatot távoli gépekkel!

Ha az „erős" Command Promptból a helyes név+jelszó birtokában sikeresen rácsatlakozunk a távoli gép egyik adminisztrációs megosztására *(C$, ADMIN$ stb.)*, ezzel léket ütünk a távoli gép páncélján *(lesz egy rendszergazdai, „erős" kapcsolatunk)*. Ezen a lyukon keresztül azután a legvadabb MMC snap-inek is beférnek, hogy ott munkát, vagy éppen mérhetetlen pusztítást végezzenek. A csatlakozás után a snap-in felvételekor bátran állítsuk át az eszköz fókuszát a távoli gépre, úgy, ahogy ezt az alábbi ábra például a Computer Management esetén mutatja:



MMC snap-in futtatása távoli gépen

Ne felejtsük el bepipálni az alsó jelölőnégyzetet, különben később nem fog menni a távfelügyelet, amikor az elmentett MMC-konzolt *(.MSC fájl)* indítjuk RunAs-zel!

Fóti Marcell
marcellf@netacademia.net

Gyógyszeresdoboz+

Windows Server 2003 Resource Kit Tools

Van másik? Van, de azért ez egyelőre nagyon más*(ik)*. Per pillanat a „do more with less" jelmondat szó szerint érvényesül az új gyári segédprogram gyűjteményének tekintetében.

Mennyi? Ennyi?

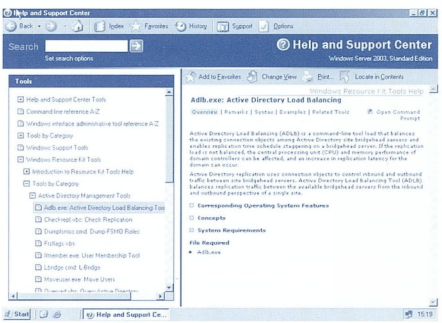
Úgy ahogy eddig megszoktuk, jelenleg nincs Resource Kit a friss Windows szerver változathoz. Persze kezdeti, illetve átmeneti időszakban vagyunk, még sok minden változhat, de innen a lelátóról úgy tűnik, hogy ez a helyzet nem is biztos, hogy sokat fog változni. Jelenleg egy mindenki számára szabadon letölthető *(a TechNet-ben sincs más)* 12,7 MB-os Windows Server 2003 Resource Kit Tools csomaggal rendelkezünk **[w2003reskit]** valamint a szintén letölthető Windows Server 2003 Deployment Kit-tel **[w2003deplkit]** és kész. A papíralapú változatról sincs túl sok biztos hír, csak annyi, hogy jövő év elejére talán kész lesz **[w2003rbook]** a 6586 oldal, amely az előzőhöz hasonlóan hét fejezetből áll és – úgy, ahogy a többi Windows Server 2003 könyvnél – kék színű futurisztikus barkácsszerszámok uralják a borítót.
Ezenkívül érdekes lehet még, hogy Mark Minasi tollából ugyanilyen címmel szintén megjelent egy könyv **[minasireskit]**, ami persze biztos jó, de nem ugyanaz. Viszont ha kész lesz az „igazi", akkor ezekből a könyvekből fog majd állni:

- Directory Services Guide
- Distributed Services Guide
- Internetworking Guide
- Networking Guide
- Server Management Guide
- Internet Information Services 6.0 Resource Guide
- Technical References. Provides the Registry Reference and the Performance Counters Reference for Windows Server 2003

Láthatóan sok változás ezekben sincs a Windows 2000 Server Resource Kithez képest, maximum a TCP/IP helyett megjelent a Directory Service Guide. Persze, ha úgy nézem, hogy magába a termékbe is beintegráltak jópár *(főleg parancssori)*, eddig csak más Resource Kit-ekben megforduló programocskát, meg aztán az IIS6-hoz is megjelent a különbejáratú Resource Kit **[iisreskit]**, és ott a Group Policy Management Console vagy a Windows System Resource Manager és a többi pl. innen **[w2003download]** letölthető ingyenes és teljes értékű eszköz, akkor azért sokat javul a kép. Az ebben a kis csomagban lévő majdnem 140 alkalmazás harmada teljesen új, *(ami tegyük a szívünkre a kezünket: nem is rossz arány a Windows 2000 Server Resource Kithez képest)* a többit újraírták, valamint a Trustworthy Computing keretein belüli biztonsági felülvizsgálat eredményeképpen is kivágtak jópár nem eléggé megfelelő alkalmazást, ami megintcsak nem baj, sőt.

Néhány észrevétel

Ha letöltöttük a csomagot *(rktools.exe)*, előtte vethetünk egy gyors pillantást a telepítési feltételekre, és konstatálhatjuk, hogy a Windows XP változatai alatt is működnek az alkalmazások *(hiszen egy család ez)*, de ettől függetlenül jónéhány természetesen Windows 2000 alatt is gond nélkül megy, illetve hogy semmi extra igény nincs, 30 MB szabad hely és ennyi. A telepítés next-next-finish típusú, a ResKit pedig szépen beleilleszkedik az XP féle Help and Support Center szín és design világába, úgy ahogy az alábbi képen is látható.



Színes, szagos, de azért kényelmes is az új felület

Egyesült a súgó és maga az alkalmazáslista, mely utóbbiban 17 kategória van *(plusz az ábécé szerinti lista és a szószedet)*. Nagyon jó ötlet volt az eszközök almenüjének egységes kialakítása a jobb oldali ablakban *(Overview, Remarks, Syntax, Examples, Related Tools)*, és az esetek nagy részében igen tartalmasak is az almenük. Viszont melegen ajánlom az „Open Command Prompt" hivatkozásra való kattintás elkerülését, mert ennek eredményeként a legtöbb esetben *(pl. a *.vbs szkripteknél)* jónéhány WSH ablakot kell „leokézni" míg megkapjuk a parancssort, ugyanis az aktuális parancsot „/?"-lel együtt nyitja meg, és ez ehhez a borzasztóan idegesítő fejleményhez vezet.
Ráadásul nem is a program mappájába teszi a parancssor promptját, hanem változó helyekre. Ezért ha többször is próbálgatni akarjuk a Resource Kit programokat, inkább a Start menüben szereplő Windows Resorce Kit Tools/Command Shell parancsikont használjuk.
De nézzük most már inkább a tartalmat, először az Active Directory Management Tools kategóriából.

QueryAD.vbs

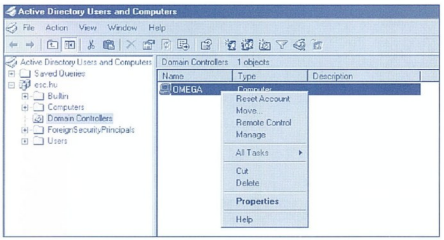
A GUI mellett van lehetőségünk immár parancssorból is keresni a címtárban, akár konkrét felhasználóra, gépre, szervezeti egységre, terjesztési listára vagy akár egy sémaattribútumra is. A keresésben maszkolhatunk is, illetve megadhatjuk, hogy melyik GC-re szabadítjuk rá a szkriptet. Az alábbi példa az s-sel kezdődő felhasználók nevét listázza a SERVER1 nevű GC közreműködésével.

```
cscript queryad.vbs -u "s*" SERVER1
```

RcontrolAD.exe

Egy távoli üzemeltetést megkönnyítő grafikus felületű segédprogramról van szó. Gyakorlatilag egy Remote Desktop kapcsolat *(amely szerveroldalához a Windows Server 2003-ban már nem kell külön Terminal Servert telepíteni, csak engedélyezni a bejövő kapcsolatokat)* amely az AD Users and Computers MMC-ből indul. Természetesen szükséges hozzá a Domain Admin jogkör is.

A kicsomagolás és a telepítés után kiderül, hogy elég egy darab – a tartományba tartozó – XP-n vagy Windows Server 2003-on feltelepíteni *(sőt egy erdőn belül többször nem is lehet mert az AD Users and Computers MMC modulját fogja kibővíteni)*, viszont ezután az összes fent említett operációs rendszerű gépet lehet ilyen módon távvezérelni, az összes olyan *(de csak szintén XP és W2K3)* gépről, amelynek %systemroot% mappájába bemásoljuk a csomag rcontrol.exe nevű alkotóelemét.

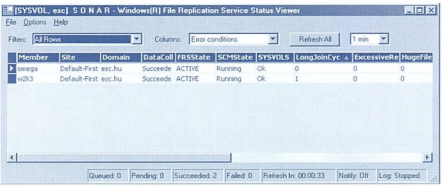


A távvezérelhető gép helyi menüje kibővül

Sonar.exe

Hangzatos neve hasonlóan kellemes és új programot takar *(prózaibb neve is van persze: FRS Status Viewer)*. A replikációs forgalom és a forgalmat prezentáló gépek ellenőrzésére, naplózására, replikációval kapcsolatos alapos statisztikák készítésére használhatjuk. Egy multimaster replikációs modellt megvalósító címtárban *(ilyen a w2k és w2k3 alapú címtár)* a replikáció alapfontosságú és megkerülhetetlen, az adathozzáférés, a hibatűrés, terheléselosztás vagy a sebesség miatt egyaránt, telephelyen belül vagy kívül is.

Szerencsére a replikációs folyamatok nagy része teljesen automatikus és minimális beavatkozást vagy optimalizálást igényel, de az ellenőrzésre és a hibakeresésre azért esetenként szükség lehet. Erre a célra szolgál ez a program. Természetesen a DC-kre célszerű telepíteni, de lehet XP vagy akár Windows 2000 Professional kliensről is használni a programot. Ami mindenképp szükséges, az a Microsoft .NET Framework valamelyik verziója. Ha viszont Windows 2000 Professionalen futtatjuk és a Windows 2000 Serverekkel kapcsolatos replikációt vennénk szemügyre, akkor szükség lesz még a Ntfrsapi.dll állományra is a DC %systemroot%\system32 mappájában. A program indítása után egy beállítópanelt kapunk, ahol a tartományt, a replikakészletet, és a frissítési időközt kell megjelölnünk vagy elfogadnunk a felajánlottakat. Rögtön megtekinthetjük az aktuális állapotot vagy akár be is tölthetünk egy már korábban lementett lekérdezést. Ezután jön a program fő ablaka, melyben táblázatos formában a replikációban résztvevő szerverek típusát vagy állapotának jellemzőit soronként illetve oszloponként *(és nagyon részletesen)* tudjuk megjeleníteni.



Az alsó géppel van egy kis probléma még...

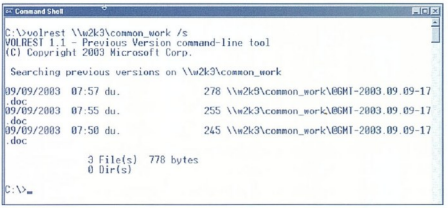
Ha a Disk and Data Management Tools-ban kutakodunk tovább, szembetűnik két eszköz, amelyeknél sokkal komplexebb és elterjedtebb megoldások is léteznek adatmentés témakörben, de egyszerűségük akár kulcsfontosságú is lehet. Ezek a CD illetve DVD író programok parancssorból: a **cdburn.exe** és a **dvdburn.exe**. Mindkettő csak image-t tud irni *(*.iso)*, de mindkettő használható lemeztörlésre is.

VolRest.exe

Az előzőeknél valószínűleg lényegesen többet használt eszköz lesz a Volrest.exe *(Shadow Copies for Shared Folders Restore Tool)*, amellyel a szerver megosztott mappáiban árnyékmásolatokat tudunk keresni, illetve vissza tudjuk állítani a törölt vagy felülírt állományok előző verzióit *(parancssorból)*. Miután egyszer már gondosan beállítottuk ezt a szolgáltatást *(jól odafigyelve szabad lemezterületre és az időzítésre)*, első körben keresünk a megosztásban:

```
Volrest \\w2k3\common_work /s /ad /as /ah
```

Az adott megosztásban árnyékmásolatokat keresünk tehát, méghozzá mappákat *(/ad)*, rendszerállományokat *(/as)*, és rejtett *(/ah)* állományokat is. Ha nem használjuk ezeket a kapcsolókat, ez utóbbi objektumok az alapértelmezés szerint szépen kimaradnak a keresésből.

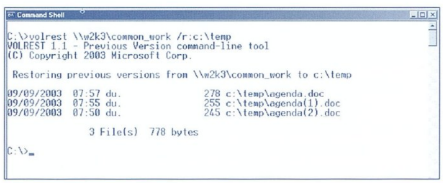


A keresés eredménye

Az itt most nem használt **/b** kapcsolóval egy egyszerűsített formátumú eredményt kapnánk.
Látható, hogy három változat is van ebben a megosztásban a példa agenda.doc fájlból, két módosított és egy törölt *(ez már nem látható, ezt csak úgy mondom)*. Ezután, hátradőlve, láblógatva, kávéval a kezünkben visszaállítunk. Jelen esetben szeretnénk mindent, és természetesen célszerűen mindig egy másik mappába *(mert jelenleg is ugyanitt lehetne az állomány egy későbbi verziója ugyanilyen névvel)*.

```
volrest \\w2k3\common_work\ /r:C:\Temp
```

Amennyiben a **/e** kapcsolót is használnánk, az üres mappák is visszaállításra kerülnének. Jelen helyzetben – mivel több azonos állománynév is van –, egy újabb dilemma lép fel, érthető módon valahogyan át kell nevezni a visszaállított objektumokat. Ez alapesetben zárójeles sorszámozással oldódik meg, ezt a következő ábrán lehet követni.

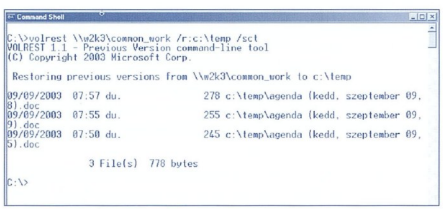


A visszaállítás boldog pillanatai

Ha ugyanezt a parancsot így adjuk ki:

```
volrest \\w2k3\common_work /r:C:\Temp /sct
```

akkor az árnyékmásolat időbélyege kerül az állománynévbe, egyszerűsítve a sorrendiség kiderítését.



A visszaállítás még boldogabb pillanatai

Ha többféle állománytípusunk lenne, maszkolhatnánk is aszerint, hogy mely típusokat szeretnénk visszaállítani.

Ehhez a témakörhöz tartozik még a **VolPerf.exe** *(Shadow Copy Performance Counters)* amely telepítése után lehetségessé válik az árnyékmásolatokkal kapcsolatos csokornyi mennyiségű teljesítményszámláló használata. Ami azért jó dolog, mert grafikusan ábrázolva kissé átláthatóbb lesz nagyobb mennyiség árnyékmásolat esetén az üzemeltetés, mint az erre a célra szolgáló, beépített **Vssadmin.exe** nevű parancssori alkalmazással.

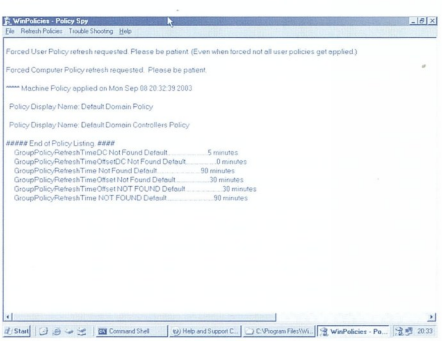
Admx.exe

Áttértünk a Group Policy Management Tools csoportra, ahol ez a segédprogram *(ADM File Parser a becsületes neve)* hasznos űrt tölt be. Két sablonállományt tudunk vele összehasonlítani, valamint egy tetszőleges sablonállományt tudunk szövegállományba lementeni. De nem ám a sima Export List, mint a Csoportházirend MMC-ben *(amely gyakorlatilag a helyi menüben csak a helyet foglalja)*, hanem ténylegesen, kulcsokkal, beállításokkal, akár magyarázatokkal együttes exportot jelent. Nyilván nem helyettesíti, hanem kiegészíti a GPMC *(Group Policy Management Console)* nagyszerű lehetőségeit, mert hiszen ott is csak import van. Abban az esetben is jól jöhet a program, ha saját sablont gyártunk, hiszen egy kiexportált gyári sablon mintaként szolgálhat.

Ebben a csoportban összefuthatunk még az **Inetesc.adm** sablonnal is, ami az alapból igen szigorú IE Enhanced Security Configuration központi beállítását segíti elő.

WinPolicies.exe

Policy Spy néven is emlegeti a ResKit ezt a szintén új és feletébb hasznos segédprogramot, amely sok praktikus funkciót fog össze egy csokorba. Komplexitiását az is mutatja, hogy külön súgója van *(winpolicies.chm)* és használható Windows 2000 operációs rendszereken is. Meglátásom szerint annak lesz igazán praktikus, aki tesztelni vagy megjavítani szeretné a Csoportházirendet, vagy esetleg kipróbálni a lehetőségeit, mert olyan ellenőrző funkciói vannak, mint pl. a Tálcába beépülés és automatikus üzenetek megjelenítése házirend változás esetén *(File/Notifiy when policy applied)*. Külön menüje van a házirendfrissítéseknek *(Refresh Policies)* ahol a Windows 2000 esetén szükséges secedit, vagy az XP/Windows 2003-hoz passzoló gpupdate parancs is kiadható. Van egy bőséges hibakeresési menüje is *(érdekes módon Trouble Shooting-nak hívják, így külön)*, ahol az érintőleges naplóállományokat nyithatjuk meg, vagy akár mi is indíthatunk egy új naplóállományt, mondjuk egy házirendmódosítás kipróbálása miatt.



Frissítési időközök dumpja a registryből

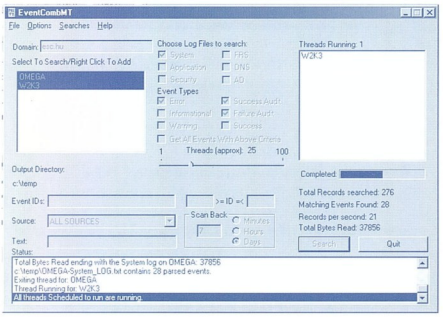
Lehetőségünk van hosszas turkálás helyett egy kattintással a registryből kinyerni a lényeges kulcsok tartalmát, pl. a frissítési időközöket, vagy akár a különböző házirendágak módosításának történetét *(Dump Machine History, Dump User History)*. Kérhetünk egy listát a rendszerben lévő összes fel-

használó/csoport SID-jéről a manuális ellenőrzéshez és még sorolhatnám az apró kis finomságokat. Ami kissé furcsa volt - és le sem tudtam tesztelni ezért - az az, hogy nem derült ki, mit takar vajon a gyakorlatban a Verify All Policies funkció. Mert azt ugyan már a menüpontban közli, hogy ez egy hosszú folyamat, de azt nem, hogy ehhez 5-ből 5 alkalommal a program szó nélküli bezárása szükséges ☺.

EventCombMT.exe

Az EventComb egy grafikus felületen működő eszköz, amely a tartományunkban lévő tartományvezérlők és tagkiszolgálók eseménynaplóiban tud széleskörűen keresni. Tökéletesen működik Windows 2000 alatt is.

A program egyszerre több gép *(mondjuk az összes DC, vagy az összes GC, stb.)* naplóállományait is át tudja fésülni az általunk beállított kritériumok alapján. A feltételek szűkíthetők naplótípusra *(System, Security, FRS, stb.)* valamint eseménytípusokra is *(Error, Warning, Success Audit, stb.)*. Ezenkívül lehet konkrét eseményazonosítóra is keresni *(vagy akár tól-ig módon, pl. 1000-1100-ig)*, és beállíthatjuk azt az időtartamot is *(napban, órában, vagy percben)*, ameddig visszafelé figyelembe veszi a naplóállományokat. Van lehetőség az esemény forrásának pontos definiálására *(pl. atapi, LsaSrv vagy éppen Kerberos)* és végül a korona: tetszőleges sztringre is kereshetünk. A keresés eredményét alapból a C:\Temp mappába egy szövegállományba menti.



Munkában az EventComb

A trükkös rész

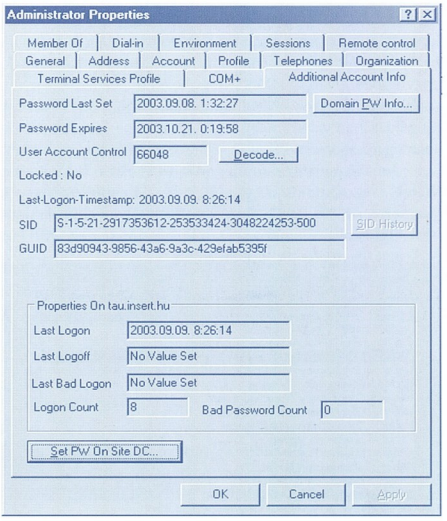
Ha nem konkrétan keresek egy már korábbról ismert programot, valószínűleg sosem vettem volna észre, hogy a felsorolt eszközökön kívül még létezik egypár a ResKit által „Individual Tool"-nak hívott programocska, jól elbújva. Ugyanis ezek az eszközök szerepelnek a közös mappában, de nincsenek kategorizálva és csak a Release Notesben van róluk szó. Nem tudom, miért van ez így, de a lényeg, hogy megvannak, mert van közöttük néhány igen hasznos alkalmazás.

Acctinfo.dll

Ha bemásoljuk a %systemroot%\system32 mappába és a következő paranccsal beregisztráljuk ezt a dll-t,

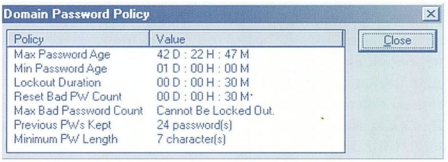
```
regsvr32 c:\windows\system32\acctinfo.dll
```

egy plusz fül *(Additional Account Info)* fog megjelenni minden felhasználó tulajdonságlapján az Active Directory Users and Computers MMC-ben. Ez az új fül tartalmazza pl. a felhasználó legutóbbi jelszóváltoztatásának, a jelszó lejáratának vagy a legutóbbi be- és kilépésének az időpontját.



Egy pillantással is lehet sokat látni

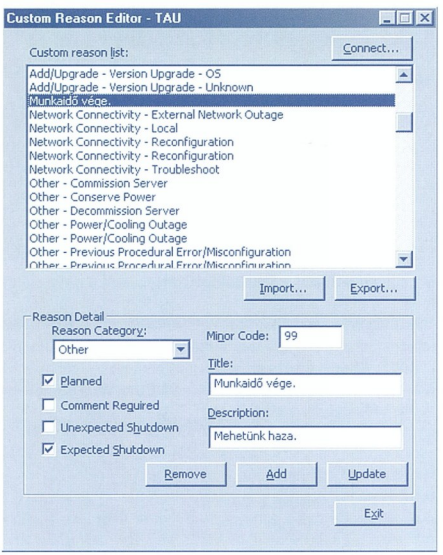
Tudnunk kell viszont, hogy ha több szerveren is használni szeretnénk ezt a funkciót, akkor mindegyikre be kell másolnunk, illetve regisztráltatnunk az említett állományt. A fentiek mellett még jópár adat összegyűjthető a felhasználóról, de talán az ismerős dolgok mellett a User Account Control mező az ami az érdekes. Ez az érték felel meg az AD-ben beállítható olyan egyedi jellemzők számszerű összegének, mint pl. a kötelező SmartCard használat a belépésnél, a fiókletiltás érvénye, vagy pl. az, hogy a jelszóváltoztatás engedélyezett-e *(ezeket ugyenezen a panelen az Account fülön, az Account options-nál pipálgathatjuk be egyesével)*. Így aztán, ha valaki ezen a panelen kicsit gyakorol a Decode gomb segítségével, ránézésre meg tudja majd mondani hogy ha a userAccountControl értéke 512 vagy esetleg 8192642 akkor az milyen következményekkel jár felhasználóra nézve. ☺ A panel jobb felső sarkában még egy hasznos dolgot vehetünk észre, ez pedig a Domain Password Policy információk.



Gyors tájékozódás a jelszóházirendben

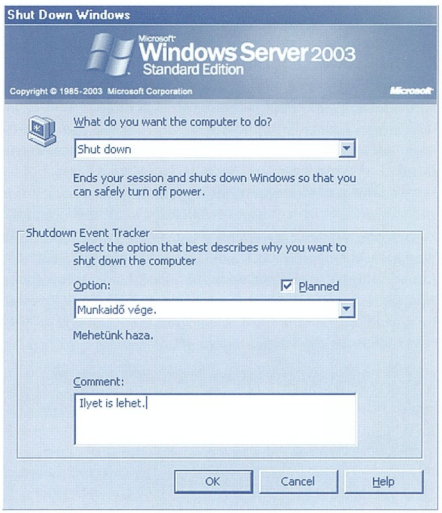
Custreasonedit.exe

Ugye mindenki lefagyott egy pillanatra, mikor először akart újraindítani vagy leállítani egy Windows Server 2003-at? Igen, én is azt gondolom, hogy a Shutdown Event Tracker elsőre meglepő volt, azóta idegesítő. Szerencsére van hozzá a Csoportházirendben opció, így le lehet tiltani a megjelenését. De ha mélyebben belegondolunk, van azért értelme ennek az új szolgáltatásnak, különösen a szervereken. Hiszen távollétünkben is bekövetkezhet újraindítás, amelyet nem biztos, hogy észreveszünk; vagy éppen elvárják tőlünk, hogy jelentést/statisztikát nyújtsunk a leállásokról, illetve újraindításokról. Ezen feladatok megoldásában valószínűleg segít rajtunk ez a szolgáltatás, hiszen egyrészt a nem kívánt újraindítások után rögvest feldobja az értesítő panelt az arra jogosult felhasználó belépésekor, illetve szorgalmasan gyűjti a bejegyzéseket a %systemroot%\System32\Logfiles\Shutdown mappában lévő XML állományba.



Szerkesztőmunka a helyi gépen (a Connecttel viszont más géphez is kapcsolódhatunk)

Ha tényleg szükség van erre a szolgáltatásra, és nem elégszünk meg az eredetileg definiált okokkal *(amit a leállítás panel listájában látunk)*, íme itt egy eszköz ennek a listának a testreszabására: a Custom Reason Editor. Beismerem, elsőre ez a bővítés/csere is elég perverz dolognak tűnik, de mégis, elképzelhető, hogy nagyon jól jön ez a lehetőség.
Az alkalmazás beüzemelése két lépésből áll. Először megkeressük a Samplereasons.reg állományt a Windows Resource Kits Tools mappából, és hozzáadjuk a registryhez a tartalmát pl. egy dupla kattintással. Ezzel a régi leállítás listánkat teljesen lecseréltük egy lényegesen bővebbre, amelyet még tovább szerkeszthetünk ezzel a parancssorból indítható, de a grafikus felületen működő alkalmazással, amelyet ezért célszerű interaktív módban elindítani *(a **/i** kapcsolóval)*.
Az eszköz azért parancssorban is tud egy-két extrát, pl. export/import funkcióval rendelkezik és a **/L** kapcsolóval indítva gyönyörűen kilistázza a már említett mappából a korábbi leállítással vagy újraindítással kapcsolatos eseményeket.



Íme az eredmény a megjelenésben

Lockoutstatus.exe

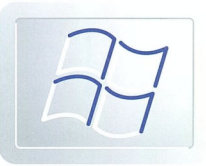
A végére maradt Account Lockout Status az előző eszközhöz hasonlóan egy hibrid képződmény: parancssorban kell elindítanunk de a GUI-n látjuk az eredményt. Ez az alkalmazás azokat a felhasználókat listázza ki, akiket a vonatkozó házirend alapján kizárt a rendszer. Nem szükséges több példányban telepítenünk a programot, mert a tartományon belüli összes tartományvezérlőről összegyűjti ezeket az információkat. A lekérdezni kívánt felhasználóra hivatkoznunk kell a parancsban, pl. így:

```
lockoutstatus /u:testdomain\gipszj
```

A program az eredményben tételesen felsorolja a megvizsgált DC-ket, a kizárás tényét *(ha ez a helyzet)* és ezenkívül az elrontott jelszavak számát is.

Gál Tamás
MCSE: Security
gtamas@tjszki.hu

Websense – Az Internetfelügyelő



Az internet adta korlátlan szabadság néha a munka rovására mehet, sőt a cég számítógépein tárolt adatok is veszélybe kerülhetnek, ha ellenőrizetlen a vállalati webforgalom.

Ma már nem kérdés, hogy a vállalati számítógépeken szükséges-e Internethozzáférés vagy sem. Számtalan elvégzett elemzés, és kutatás kimutatta, hogy jóval hatékonyabb a munkavégzés, ha a dolgozó azonnal hozzáfér a világhálóhoz és a keresett adatot rövid időn belül eléri a saját gépén. A kérdés csak az, hogy a neten való keresgélés közben esetlegesen elkalandozó figyelem nem hátráltatja-e a munkavégzést? A válasz sajnos: De igen!

A munka helyett file-cserélgető hálózaton MP3-at vadászó kolléga, illetve a legfrissebb játékokat, filmeket letöltő társa nem éppen tevékenyen járul hozzá a cég éves termeléséhez. A vállalati vezetés szempontjából az elfoglalt, túlterhelt sávszélesség, illetve mostanában a számítástechnikai bűnesetek középpontjába került file-cserélgetős ügyek miatt járhat kellemetlen következményekkel az ellenőrizetlen webforgalom.

Éves jelentések készítésekor felmerülhet a kérdés: „Hogyan alakult az idén a vállalati Internetezés?" Vagy akár kérhetik tőlünk rendszergazdáktól, hogy statisztikai adatokkal indokoljuk az Internet-elérés fejlesztésére elkölthető pénz létjogosultságát. Mert ha a nem csekély összegért bekötött műholdas kapcsolaton keresztül csak a filmletöltés lett gyorsabb, hamar lemondhatunk a cégvezetés jövőbeni támogatásáról...

Visszatekintés – Hogyan volt? Mivé lett?

Azt hiszem, minden kedves olvasóban él még az emlék, amikor „régen", a hazai hálózatok őskorában nagyméha véletlenül elérhető volt egy-egy gépen a csigalassúságú „modemes" Internet. Nos, ott semmiféle védelemről, szűrésről, netán tűzfalról szó sem volt. Ez utóbbi fogalom is maximum az építészetben volt akkoriban ismerős. A gépek *(igen nagy dolog volt a többes szám)* mindegyike külső „éles" IP címet kapott, és a cég hálózatából igencsak széles kapuk nyíltak a külvilágra. Drága volt és elérhetetlenül misztikus.

Valami borzongással vegyes áhítattal figyeltük a bennfentes kolléga billentyűkön zongorázó ujjait, amikor is az első ártatlan statikus weboldal megjelent a villódzó képernyőn. Erről jut eszembe a régi számítástechnikus szóvicc is:

– CGA monitor?
– Nem, saját!

Ugye milyen furcsa, de egyben kellemes érzés visszagondolni ezekre az időkre? Én ekkortájt döntöttem el, hogy ez lesz a szakmám. A rendszergazdai pozíciókban akkoriban dolgozó kivételes mérnökökre úgy gondoltam, mint ötévesen a pilótákra vagy mozdonyvezetőkre: Istenként tiszteltem őket. Az őskorban, amikor a gépek, a hálózat és a teljes infrastruktúra teljesítménye a maihoz képest elenyésző volt is, jó érzés volt látni a mi kis öreg gépünkön a „Világhálót" megtestesítő kezdetleges oldalakat, és persze a folyton ott tevékenykedő, érthetetlen szavakat használó fura figurát: a rendszergazdát.

Ki emlékszik például olyan fogalmakra mint Gopher, Finger, Usenet, Arpanet, Fidonet vagy Wais, MUD, MOO, Archie netán BBS, Bitnet? Néhányról ma is hallani még, ám sokuk feledésbe merült az idők folyamán. Ne felejtsük el viszont, hogy már a korai neten is lehetett információkat keresni, levelezni, olvasgatni – mai szemmel nézve nevetségesen alacsony szinten, de lehetett! És ez volt a fontos: A lehetőség. Mert ennek a felismerése indította el azt a folyamatot, ami mára megszülte ezen fogalmakat, hogy „információs-szupersztráda", meg „információs-társadalom", stb. De születtek egyéb új fogalmak is, mert ugye a jó és a rossz mindig kézenfogva járnak. Így lett: Vírus, Trójai, Worm, SpyWare, Hacker, Cracker és még sokan mások, hogy csak egy párat említsek a sok közül, amitől szépen lombosodott az egykori kis fácska. Mostanra már elég nagy és terebélyes lett ahhoz, hogy a biztonságtechnika szükségszerűen szorosan összeforrjon a számítástechnikával. Ma már libabőrös lesz az összes hozzáértő szakember attól a gondolattól, hogy a céges hálózatban levő gépek külső IP címeket kapjanak tűzfal nélkül. Állítom, hogy komolyabb károsodás nélkül maximum 10 percet működhetne így egy mai hálózat. Mert sajnos ott tartunk, hogy a károkozás is automatikus, mintha egy láthatatlan király egy géphadsereget vonultatna fel ellenünk és számítógépeink ellen. A megtámadott-legyőzött gép átáll az ellenséges csapatba és onnantól ő is az információk pusztításában vállal szerepet. Paradox módon ember alkotta valamennyiüket.

> Ha azt a bizonyos 80-as számú téglát kivesszük a tűzfalból, egy halom ártó szándékú dolgot is beengedünk, akaratlanul is...

Persze filozofálgatással nem sokra megyünk, cselekedni kell! Az első és legfontosabb lépés egy jó tűzfal és egy megfelelő tudással bíró hálózati szakember. Együtt szinte minden káros dolgot kívül tudnak tartani, védve a belső szeparált hálózatot. Ám a folyamatos, gyors munkavégzés biztosítása érdekében kénytelen-kelletlen neki is muszáj időről-időre lyukakat ütnie a tűzfalon. Ha az Internet – és ezen belül a World Wide Web – nyújtotta előnyöket beeresztjük a céges falak mögé, számolnunk kell egyéb új, a biztonságot veszélyeztető tényezővel is. Ha azt a bizonyos 80-as számú téglát kivesszük a tűzfalból, egy halom ártó szándékú dolgot is beengedünk, akaratlanul is. Annyi mindent bele lehet csomagolni már a HTTP-be, hogy a lehetőségek száma végtelen, csak a programozók állhatatosságán és kitartásán múlik az egész. Jusson eszünkbe az

Exchange 2003 szerver és legújabb kliensének speciális webes képessége, ami – ha szeretnénk – az RPC-t bújtatja bele a HTTP-be. Van tehát kiemelkedően jó oldala is a webalapú fejlesztéseknek, de nemcsak a jó szándékú programozók „dolgoznak" a lehetséges újításokon.

A megoldás: Websense

Az előbbiekben ismertetett problémákra jó megoldást kínál a **[websense]**-es címen elérhető termék. Az átmenőforgalom alapú ellenőrzést megvalósító eszköz alkalmas a webforgalom szűrésére és a hozzáférések aprólékos szintű beállítására. Egyes oldalakat blokkolhatunk, illetve engedhetünk a legkülönfélébb szinteken: például az egész vállalati hálózaton, vagy csak egyes gépekre, vagy csak egyes userek-re, csoportokra nézve. Még időalapú korlátozás is lehetséges, azaz ami nem érhető el munkaidőben, munka után lehet nézegetni.
Az alkalmazott módszer hasonlatos ahhoz, amiről egy korábbi cikkemben *(tech.net 2003. augusztus: SPAM, anti-SPAM)* a BrightMail által használt metódusról már írtam. A rendszer lényege, hogy egy adatbázisban gyűlnek a gyártó cég által nemkívánatosnak mondott tartalmú oldalak, domain-ek és ezt az adatbázist használja, töltögeti szorgalmasan az általunk telepített Websense Enterprise. A Windows NT illetve W2k szerver alapú változat mellett létezik Sun Solaris és Linux operációs rendszerhez készített verzió is. A Windows 2003 szerverre fejlesztett változat a cikk születésének idején az ígéret stádiumában állt.

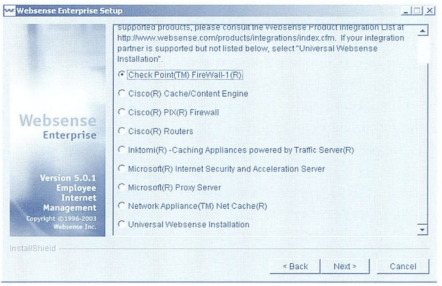
A Websense software-csomag

A cég weboldaláról **[websense]** letöltve a demo verziót, a következő opciókat választhatjuk önálló csomagokban:

- Websense Enterprise
- DC Agent
- Language Pack
- Network Agent
- Reporter
- Websense Explorer
- Websense Manager

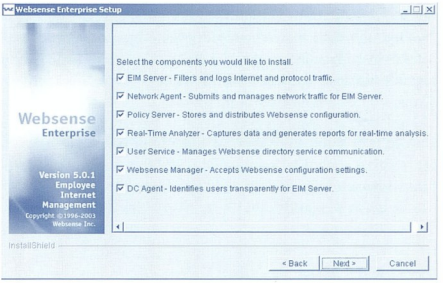
Nagyon hamar kiderül, hogy egy igen összetett és bonyolult feladatokra felkészített termékkel állunk szemben. Első látásra ijesztőnek tűnhet a sok-sok funkció, beállítás. De ne keseredjünk el, a második pillantásra is az. A fentebb felsorolt összetevők együttes mérete 500 MB körül van és ezek még csak az install-anyagok! Telepítés után – a választott komponensek függvényében – ennél kevesebb hely is elég lehet, de az alkalmazott szűrési módszerek tekintetében érdemes újragondolni a gép hardverelemeit. Memória szempontjából például már az installer is jelzi igényeit, hogy a későbbiekben ebből majd sok kell!
Az Enterprise install eleve tartalmazza a felsorolt összetevők közül az önálló működéshez szükséges összes lényeges elemet, és ezek egy általános telepítés után kb. 200 MB-ot fogyasztanak el a merevlemezről. Elsőre nekem kicsit zavaró volt, hogy külön-külön tölthetők le az egyes programrészek, mert azt hittem, hogy majd egyszerre kell mind a teljes működéshez. Ez nem is áll távol a valóságtól, csak szerencsére az Enterprise-ban minden benne van, ami ténylegesen szükséges. A többi csak azért érhető el külön, ha esetleg a funkciókat elválasztva szeretnénk, több gépen elosztva telepíteni, vagy önmagában kipróbálni.

A programcsomag teljes, mélyreható ismertetése túlmutat a cikk keretein, inkább csak kedvcsinálónak csemegézek a funkciók erdejében. Így a részletes termékismertetőtől is eltekintenék most, hiszen ez elérhető a gyártó weboldalán. A Websense fő tevékenysége természetesen a Web-szűrés, az összes többi a kapcsolódó, kiegészítőktől zsúfolt bedolgozó, segítő modulok sokasága. Mint a felhasználói igényeket maximálisan figyelembevevő alkalmazás, a Websense is igazodik a már meglévő megoldásokhoz. Ha a vállalati hálózaton van már egy tűzfal – ez ugye nagyon valószínű –, vagy proxy esetleg ISA szerver, a Websense képes ezekkel együttműködve végezni feladatát. Az alábbi képen látható, hogy a legtöbb napjainkban használt hálózati technológiával képes összedolgozni.



A Websense együttműködési listája.

Talán a képen nem jól látszik, de az installer felhívja a figyelmet, hogy az elérhető megoldás-szállítók felsorolása nem teljes, a többi gyártóhoz a további kompatibilitási lista a **[webs_integr]** címen érhető el. Ahhoz, hogy a lehető legtöbbet kihozzuk a program képességeiből, természetesen együtt kell működnie a címtárral. Erre a DC Agent nevű komponens segítségével képes. A védendő hálózat méreteitől, illetve az átmenő webes forgalom nagyságától és a már alkalmazott egyéb védelmi infrastruktúrától függően a modulokat szétválaszthatjuk, külön szerverekre tehetjük. A következő kép az Enterprise install részelemeit mutatja.



A Websense Enterprise és komponensei.

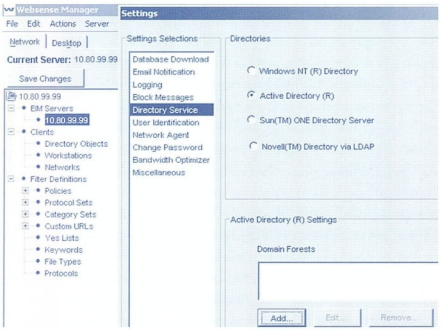
Az egyik alprogram, a Real-Time Analyzer például igen hasznos, mert használatával naprakész reportokat készíthetünk. Ez persze nemcsak a vállalatvezetésnek jó, hanem saját magunknak is, amikor finomhangoljuk a beállításokat. Meg fo-

gunk ám lepődni, hogy mi minden jön be a hálózatra anélkül, hogy tudnánk róla. Ezt egy életszagú példával szemléltetném: Nálunk a hálózatban a hotbar.com domain felől érkezett egy ártatlannak tűnő kis Outlook kiegészítés. Az említett weboldalra tévedve szinte azonnal települni akaró software az Outlook arculatát hivatott kissé „feldobni". Színes templateeket, gombokat, grafikákat helyezhetünk el vele a szokványos levelezésben. Kérdés, hogy ennek mi értelme van, azon kívül, hogy erősen erőforrásigényes az így születő felcicomázott levelek megjelenítése? Természetesen ezekre semmi szükség, de a legtöbb felhasználónak nagyon tetszik.

Az ily módon „feljavított" e-mailekben már ott hirdeti a program önmagát és egy link segítségével buzdítja a többi felhasználót, hogy ők is lépjenek be az Outlookot lassító összetevők önkéntes telepítőinek népes táborába. Az arcátlanság csúcsa, hogy a weboldal még egy hamis Microsoft Authorized Partner logót is mutogat, a gyanakvóbb felhasználóknak. Egy futó Spyware detector jelezte, hogy az ártatlannak tűnő install során mi minden települt volna a gépre, ha hagyjuk: backdoor, remote information-collector, stb. Eme utóbbi eszköz tevékenységéről csak sejtéseim vannak, de még így sem szeretném, ha az Outlookban nekem ilyenem lenne...

Nos, ez a „webfertőzés" és sok más társa simán elkerülhető a Websense alkalmazásával. A reportok elkészültével például igencsak szemben tőlő lehet az összes gépről egy bizonyos irányba tartó hatalmas háttér-webtevékenység, amit a kis ártalmatlannak tűnő programocskák generálnak. Az említett példa csak egy a sok közül, aminek veszélyeire baráti elbeszélgetés során nem lehet a felhasználókat felkészíteni. Az ilyesmit kívül kell tartani a belső hálózatból. Láthatjuk, hogy ez a problémakör tűzfallal csak nehezen kezelhető, hiszen állandóan változó, dinamikusan „fejlődő" web-es támadási módszerek jelennek meg napról-napra. Ha ezt állandó szabályrendszerszerkesztéssel szeretnénk manuálisan elhárítani, igencsak sok dolgunk lesz.

Ezért is képes a program az összes ismert, „nagy" tűzfalgyártó termékével együttműködni. A következő képről az is kiderül, hogy a címtárak közül sem vagyunk pusztán az Active Directoryra ítélve, hanem más neves gyártók hasonló megoldásaival is bátran kihasználhatjuk a Websense nyújtotta előnyöket.



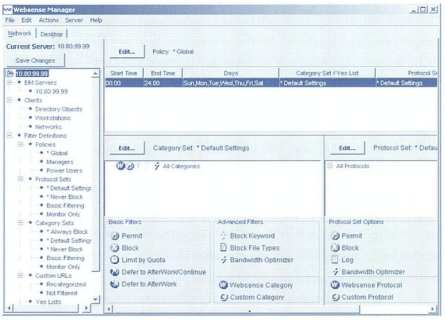
A Websense által ismert címtárfajták.

A webes támadások többsége a felhasználók jóhiszeműségére, illetve a „sok kicsi sokra megy" elvére alapozva próbálja meg kicsikarni azt a bizonyos Yes-t a gép előtt ülőkből. Gondolok itt az Explorerben időnként fel-felbukkanó kis figyelmeztető ablakokra, ahol nekünk kell kézzel engedélyezni vagy tiltani egy-egy oldalról érkező hasznos vagy káros programkódot. A véletlenül megtalált gonosz szándékú oldal kíséretében automatikusan felbukkant tíz-húsz popup ablakot zárogatva óhatatlan, hogy az átlagos felhasználó itt-ott el ne fogadjon egy kétes tanúsítvánnyal rendelkező oldalt.

Legrosszabb esetben az ablak alján található „Always trust" kezdetű mondat élén egy pipával egyetemben... Ezzel a problémával szemben sem elég, ha kioktatjuk a felhasználókat az ilyenkor szokásos helyes tennivalókról. A potenciálisan veszélyes oldalakat egyszerűen ki kell zárni a hálózatból, kiküszöbölve ezzel a véletlen károkozás esélyét is.

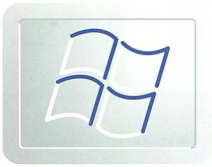
A biztonságos web-elérés megvalósításán túl egyéb problémákkal is szembesülhetünk. Például rengeteg olyan kérdés merülhet fel a felsőbb IT vezetés felől, hogy meséljük csak el, hogy a dolgozók mit nézegettek a legtöbbet a cég számítógépein? Vagy az újonnan kifejlesztett méregdrága információs weboldal beváltotta-e a hozzá fűzött reményeket? Ilyen és ehhez hasonló kérdésekre a forgalom figyelése és részletes naplózása nélkül biztosan nem tudunk érdemi választ adni. De ha van Websense, máris két legyet üthetünk egy csapásra. Egy eszközzel megoldható a többszintű központosított szabályzás, naplózás, kiértékelés.

Ha most újból visszagondolunk a régi időkre – mint ahogy azt a cikk során már egyszer megtettem – és összehasonlítjuk a pár évvel ezelőtti Internetezési szokásokat a maiakkal: ugye manapság már szükség van a nagyobb hálózatok webforgalmának kontrolljára? Mint ahogyan szükség van a tűzfalra is és a vírusvédelemre is. Ezek nélkül a mi feladataink megsokszorozódnának, elvéve az időt a hasznos munkától. Az ismertetett példánál maradva: nekünk kell majd odamenni a felhasználó gépéhez és kézzel leirtani róla a Hotbar által feltelepített ártalmas programokat. De ilyen ám a Gator és a Kazaa is. Ha ezek ismeretlenül csengenek, látogassuk meg a **[spyware]** címet. Végezetül megmutatnám a Websense Manager arcát, van itt minden, ami kellhet. Próbálják ki, hiszen csak a káros weboldalakat veszíthetik vele.



A Websense Manager kezelőfelülete.

Füzesi Szabolcs
fuzesisz@osi.hu
MCSA



A DHCP rejtett szépségei – VI.

Azt már korábban is láthattuk, hogy a DHCP szolgáltatás nem áll önmagában, számos más komponenssel együttműködik. A DNS, az Active Directory, az RRAS és a DHCP viszonyát már tisztáztuk. A címkiosztó szerver azonban egyéb funkciókkal is szorosan integrált, sőt néha saját maga tartalmazza ezeket az alkalmazásokat. Ezúttal a BOOTP, a MADCAP, a RIS és a DHCP közti összefüggéseket vizsgáljuk.

BOOTP – a félig lenyelt előd

A DHCP szolgáltatás nem volt előzmények nélküli, isteni fejből kipattant szikra. 1985-ben Bill Croft a Stanford egyetemről és John Gilmore a Sun Microsystemtől készítettek egy szabványtervet, amely RFC 951-es számmal és „Bootstrap Protocol *(BOOTP)*" címmel vonult be a történelembe. Az ő megoldásuk sok tekintetben hasonlóan működött, mint a mai DHCP kiszolgálók, bár nem pontosan ugyanúgy. Az eltérések az alkotók indítékaiból, az előttük álló feladatokból fakadtak. A fenti két úriember a lemez nélküli munkaállomások elindulási folyamatát szerette volna automatizálni. Így a BOOTP szabvány egy kétfázisú indítást definiált. *(Az RFC az elsőt írta le részletesen.)*

Az első fázisban a munkaállomások UDP 67 *(szerver)* és UDP 68 *(kliens)* portokon keresztül kommunikálnak. A szerver egy IP címet biztosít a lemez nélküli gépnek, továbbá néhány paramétert, amelyet ők gyártói kiegészítéseknek *(vendor extensions)* neveztek. Azt az információt, hogy a munkaállomás melyik kiszolgálóról, milyen nevű boot állományt tölthet le az induláshoz szintén a BOOTP kiszolgáló adja meg. A második fázisban a lemez nélküli rendszer TFTP protokollon keresztül felveszi a kapcsolatot a korábban megadott szerverrel, lemásolja a boot állományt, majd azt betöltve elindítja a szükséges operációs rendszert.

Tulajdonképpen az RFC szerzői oldották meg a tyúk-tojás problémát, amit mi korábban paradoxonként emlegettünk, vagyis ők javasolták a szórt üzenetek alkalmazását. Láthatjuk, hogy az első fázis igen-igen hasonlít a DHCP működési elveihez. Központi címkezelésről van szó mindkét esetben, ugyanazokat az UDP kapukat használják a BOOTP és DHCP kiszolgálók, kliensek, továbbá az üzenetváltásnál a csomagszerkezet is megegyezik, egy kivételtől eltekintve: a BOOTP csomagok csak 64 oktetnyi helyet biztosítanak a gyártói kiegészítéseknek, míg a DHCP 312 oktetet értelmez, amelyben a különböző opciók utaznak. *(A két fogalom – opció és gyártói kiegészítés – gyakorlatilag ugyanazt takarja.)*

Ezek a hasonlóságok arra késztették a DHCP kiszolgálót megalkotó programozókat, hogy lehetővé tegyék a BOOTP ügyfelek támogatását. A Microsoft a Windows NT 4.0 SP2 óta támogatja az idősebb testvér klienseit.

A hasonlóságok mellett szólni kell a meglévő különbségekről is. Jó kiindulási alap a szállítói kiegészítések felsorolása.

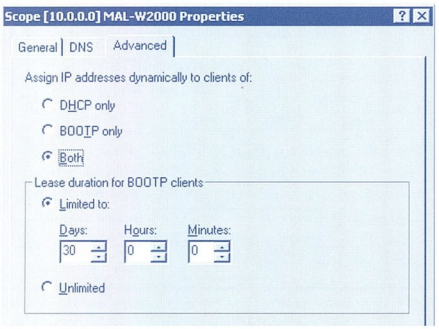
Kód	Leírás
1	Subnet Mask
3	Router
4	Time Server
5	Name Server
9	LPR Server
12	Computer Name
15	Domain Name
17	Root Path
42	NTP Servers
44	WINS Server
45	NetBIOS over TCP/IP Datagram Distribution Server
46	NetBIOS over TCP/IP Node Type
47	NetBIOS over TCP/IP Scope
48	X Window System Font Server
49	X Window System Display Manager
69	SMTP Server
70	POP3 Server

A fenti lista lényegesen rövidebb, mint a DHCP opciók teljes táblázata, ráadásul néhány nagyon fontos paraméter is hiányzik. Nincs sehol szó bérletidőről *(51-es opció)* megújítási időről *(renewal time, 58-as opció, rebind time 59-es opció)*. Ezek szerint egy BOOTP kliens nem újítja meg a bérletét? Nos, nem újítja meg, mivel az IP cím kérését nem bérletviszonynak fogja fel. S íme, ez a legnagyobb különbség egy DHCP és egy BOOTP ügyfél között. Egy BOOTP kliens kér ugyan IP címet a rendszerinduláskor, s minden újabb induláskor is, ám onnantól a címet saját magáénak érzi. Nincs további kapcsolata a BOOTP kiszolgálóval! Hogyan lehet így címeket kezelni? Hiszen az egyszer kiadott címről azután már nem rendelkezhetünk, nem vonhatjuk vissza.

A BOOTP világ másképp működik, mint a DHCP. A címek kiadása ugyanis a szabvány szerint egy táblázat alapján történik, amelyben előzőleg rögzítik a leendő ügyfél azonosító adatait, többek között a MAC címét is. Ha analógiát keresünk, a DHCP lefoglalt címeihez hasonlíthatjuk az eljárást – a bérletidőtől eltekintve. Vagyis szó sincs ellenőrizetlen folyamatokról, épp ellenkezőleg. A BOOTP világ nem is kaphat címet olyan állomás, amelyről a rendszergazda nem tud. Kizárólag a BOOTP táblázatba felvett hostok nyernek bebocsátást a hálózatba.

No persze ez nem is olyan kényelmes megoldás, mint a DHCP, mivel előzetesen be kell gyűjteni legalább a MAC címeket, a boot állományneveket, a TFTP szerver címeket, kézzel frissíteni a táblát stb., stb. Mai szemmel nézve meglehetősen fura, a szabvány megalkotásakor azonban még nem kellett naponta tucatjával telepíteni és kivonni gépeket a hálózatból, vagyis a fenti megoldás is kielégítő volt. Arról nem beszélve, hogy a DHCP-hez képest a BOOTP világ még biztonságosabbnak is tekinthető, hiszen kizárólag a rendszergazdák explicit engedélye után kaphat egy host címet, ami azért erőteljesebb kontroll, mint egy scope aktiválása. A DHCP biztonságá-

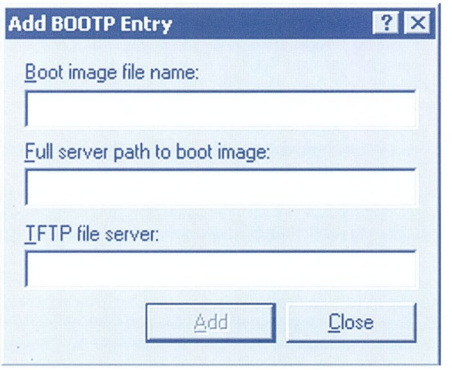
ról azonban később még lesz szó, most nézzük, hogyan is fest a konkrét BOOTP implementáció.
A Windows NT 4.0-ban a támogatás azt jelentette, hogy a szerver elfogadta a BOOTP csomagokat, majd a lefoglalt címeket *(reserved addresses)* kezdte el vizsgálni. Amennyiben talált olyan MAC-címet, amely a csomagban szereplő ügyféllel megegyezett, a bérletet kiosztotta. Amikor tehát korábban a lefoglalt címek működéséhez hasonlítottuk a BOOTP megoldásait, egyáltalán nem jártunk távol az igazságtól, mert a DHCP BOOTP „emulációja" épp ezt a lehetőséget használja ki.
A Windows 2000-ben továbbfejlesztették a BOOTP támogatást, és a fenti megoldás megtartása mellett bevezették a „dinamikus BOOTP ügyfelek" fogalmát. Ez azt jelenti, hogy már nem kell a lefoglalt címek közé felvenni előre a BOOTP ügyfeleket, azok éppen úgy egy aktív bérlettartományból kapnak címet, mint a közönséges ügyfelek. Csupán két dologra kell figyelni. A bérlettartomány tulajdonságai párbeszédpanelen az „Advanced" lapon engedélyezni kell a BOOTP ügyfelek dinamikus kiszolgálását, továbbá meg kell határozni, hogy mennyi időre kapják meg kiadott IP-címet.



A BOOTP ügyfeleknek 30 napra szól az alapértelmezett bérlet

A 30 napos bérletidő eléggé veszélyes dolog. Attól, hogy egy DHCP kiszolgáló DHCP ügyfélként szeretné kezelni a BOOTP klienseket, a szabvány még nem változik meg. Márpedig a BOOTP ügyfelek nem fogják megújítani a bérletüket. Ha tehát 30 napon túl sem indítjuk újra őket, a DHCP kiszolgáló másnak is kiadhatja a címüket. Bölcs rendszergazda tehát ezt az értéket legalább 100 napra, vagy még inkább 365 napra módosítja. Manapság ugyanis a BOOTP ügyfelek eléggé sokáig képesek üzemelni egyhuzamban. A dinamikus támogatásnak van még egy fontos feltétele: az ügyfelek csak IP címet kérjenek. Ha szükségük van a boot állomány nevére, TFTP szerver címére stb., akkor a dinamikus BOOTP ügyfél mint megoldás nem jöhet szóba. Valamilyen módon ugyanis hozzá kell rendelni a fenti adatokat is a klienshez. Marad a korábban felvillantott „reserved address" alapú megoldás. Felvesszük a BOOTP ügyfeleket a lefoglalt címek közé – megadva a MAC címét, nevét stb. és bejelölve, hogy BOOTP kérés érkezik majd.
Ezután be kell állítani két speciális opciót, a 66 és 67 számút. Az első TFTP kiszolgáló nevét, a második a bootfájl nevét tartalmazza.

Végül a szerver tulajdonságai lapon bekapcsolhatjuk a BOOTP tábla mappát. Ha ezt megtettük, lehetőségünk van három adattal ellátni a BOOTP ügyfeleket. Az első és a harmadik már ismerős, itt azonban megadhatjuk még a teljes elérési utat is a boot állományhoz. A klienshez felvett értékek alapján a tábla bejegyzéshez egyértelműen hozzárendelhető a BOOTP ügyfél is. Ugye nem bonyolult?
A Windows 2000 ennél tovább nem megy. A súgóban azt a fura mondatot olvashatjuk, hogy a TFTP szervernek egy harmadik gyártótól kell származnia, mert a Windows 2000 nem rendelkezik ilyen kiszolgálóval.



A BOOTP szabványt követő párbeszédpanel

A fenti állítás egy kicsit sántít, hiszen a RIS kiszolgálóhoz dukál egy TFTP is. A pontos megfogalmazás tehát az lehetne, hogy a Windows 2000-et nem szállították olyan TFTP kiszolgálóval, amely a BOOTP ügyfeleket is kiszolgálná. *(Kár, hiszen a PXE kliensek nagyon közeli rokonai a BOOTP-t használó állomásoknak.)*
A BOOTP szolgáltatásról összességében annyit mondhatunk, hogy a szabvány hű követése, ám nem teljes megoldás. Amennyiben azonban az IP-cím kiosztást tekintjük feladatnak, úgy a DHCP kiszolgáló megfelelő támogatást nyújt a BOOTP ügyfeleknek is.

MADCAP

Ha a BOOTP szabvány a múlt, akkor a Multicast Address Dynamic Client Allocation Protocol *(MADCAP)* egy kicsit talán a jövő. Nem azért, mintha a multicast szabvány nem lenne legalább olyan idős, mint némely mai középiskolás *(1986-ban jelent meg az RFC 988, amelyben először definiálták)*, hanem inkább azért, mert a rá épülő alkalmazások, mint a videó-konferencia, azonnali üzenetküldés, e-learning megoldások, stb. még csak most kezdik meg hódító útjukat.
A MADCAP megértéséhez minimális szinten érteni kell a multicast világot is. Az már közismert, hogy a gépeknek egyedi IP címmel kell rendelkezniük, amelyek A, B, C osztályba sorolhatók, esetleg osztály nélküliek. Léteznek azonban D osztályú címek is, ezeket onnan lehet felismerni, hogy a IP cím első négy bitje 1110, vagyis egy multicast cím a 224.0.0.0 és 239.255.255.255 tartományba esik. A multicast címekre a következő szabályok vonatkoznak:

1. Egy állomásnak rendelkeznie kell legalább egy unicast címmel, mielőtt egy multicast címet kap.
2. Egy multicast címet több állomás is megkaphat – tulajdonképpen ez az értelme. Egyedi címeknél ez tilos. A szabvány szerint az azonos multicast címmel rendelkező állomások halmazát nevezik multicast csoportnak, amelynek nulla vagy annál több tagja lehet. Egy csoport nem csak egy logikai alhálózatból fogadhat tagokat, és a hálózat tudja, hogy mely állomások tagok, és azok hol helyezkednek el. A csoport dinamikusan változhat. Bármely állomás bármikor csatlakozhat, vagy elhagyhatja a csoportot. *(Ennél mélyebbre nem megyünk, habár a téma izgalmas.)*
3. Egy állomásnak több multicast címe is lehet egyidejűleg.

Tudni kell még, hogy bizonyos címek foglaltak, vagy speciális tulajdonságokkal bírnak. Néhány példa:

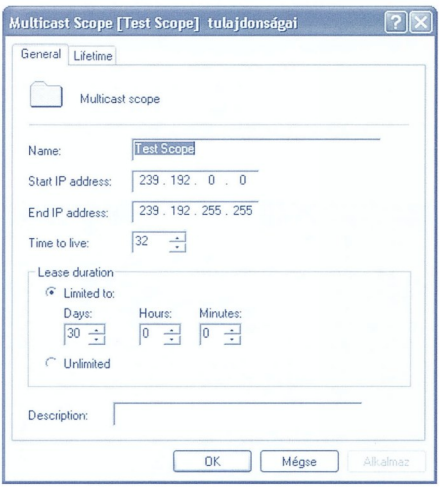
1. A 224.0.0.0-224.0.0.255 címek csak a helyi alhálózaton érvényesek, az útválasztók semmiképp sem továbbítják őket.
2. A 239.0.0.0-239.255.255.255 címtartomány olyan alkalmazások számára van fenntartva, amelyek elérhetőségét *(vagy inkább „sugárzási távolságát")* szabályozni lehet.
3. A fenti címtartományon belül a 239.192.0.0 255.252.0.0 alhálózati maszkkal nagyjából hasonló szerepet játszik a multicast világban, mint a 10.0.0.0/8 az unicast forgalomnál. Belső használatra ajánlott címtartomány, ideális a MADCAP kiszolgálók számára. Ez a címtartomány 262144 címet jelent, ami óriási szám, ha meggondoljuk, hogy egyetlen címet akár több ezer állomás is használhat.
4. 224.0.0.1 – minden állomás a helyi hálózaton.
5. 224.0.0.2 – minden útválasztó a helyi hálózaton
6. 224.0.0.5 – OSFPv2 útválasztók a hálózatban
7. 224.0.0.6 – OSFPv2 útválasztók a hálózatban
8. 224.0.0.9 – RIPv2
9. 224.0.1.1 – Network Time Protocol

A 224.0.1.0 – 238.255.255.255 szabadon felhasználható multicast-ra épülő alkalmazások számára, legyenek akár az Interneten is. Persze a címeket ugyanúgy meg kell igényelni, de talán itt nincs olyan címéhség, mint az egyedi címek esetében.
Fontos tisztázni, hogy a multicast címek függetlenek az egyedi IP azonosítóktól, tehát a MADCAP szerver sem függ a DHCP címtartományok meglététől. Ha egy kiszolgálón csak multicast bérlettartomány található, akkor azt MADCAP szervernek mondjuk. Ám később itt is éppúgy lehet „normális" scope-okat létrehozni, jól megférnek egymás mellett.
Az sem szükséges, hogy az ügyfél unicast címe egy DHCP kiszolgálótól származzon és fordítva: egy DHCP ügyfél nem csak a MADCAP által adott multicast címmel kapcsolódhat olyan csoporthoz, amely többi tagja viszont MADCAP ügyfél.

A MADCAP kezelőfelülete

Egy multicast címtartomány létrehozása épp olyan könnyű, mint egy közönséges bérlettartományé, sőt, ha lehet, még egyszerűbb. A MADCAP kiszolgálók ugyanis szinte kizárólag címtartományokat definiálnak, opciókat azonban nem – ennyivel egyszerűbb az élet.

A varázsló, amely a scope-ot létrehozza, csupán a címtartomány nevét, kezdő- és végcímét, a kizárt címeket és a bérletidőt kérdezi meg. Mindezt azonban később is beállíthatjuk a scope tulajdonságai között, ahogy a következő ábra mutatja.
A multicast címtartományokkal nem bánik olyan finnyásan a kiszolgáló. A tulajdonságok lapon könnyedén módosíthatjuk a kezdő és végső címet is, ha ez szükséges.
Külön említést igényel a TTL érték. Az útválasztók számát jelenti, amelyeken még áthaladhat a multicast forgalom. Ez egyszerűen szabályozza, hogy „milyen messziről" érhető el az adott multicast alkalmazás (pl. egy e-learning tanfolyam).



Egyszerűen konfigurálható multicast címtartomány

A multicast világnak van egy különös sajátossága, az illékonyság. Egy videokonferencia záros időn belül véget ér, akárcsak egy tanfolyam, vagy egy élő közvetítés. A címeket hipp-hopp elhagyják a munkaállomások, sőt akár az egész címtartomány is feleslegessé válhat. A Windows 2000 készítői számoltak ezzel a szituációval. A multicast bérlettartomány tulajdonságai között a második lapon beállíthatjuk, hogy a scope meddig éljen, szakszóval: mikor járjon le. Ha eljött az idő, üt a scope órája. Elillan, mintha sohasem lett volna.
Amilyen összetett és bonyolult tud lenni egy multicast forgalmat lebonyolító hálózat, olyan egyszerű a MADCAP életre keltése – ez most kiderült. Egy valamit azonban érdemes fejben tartani: a MADCAP csupán egy eleme a hálózatnak, amely a fenti alkalmazásokat biztosítja. Szükséges, de nem elégséges feltétele multicast alkalmazások bevezetésének. Nem kerülhető el az útválasztók alapos felülvizsgálata és alkalmassá tétele e speciális forgalom lebonyolítására.

A DHCP és a RIS

Még 2001 végén indult e lap hasábjain egy öt részből álló sorozat, amelynek keretében az Olvasó részletekbe menően megismerkedhetett a RIS, vagyis a Remote Installation Services rejtelmeivel. Most csak röviden szólunk a RIS és a DHCP kapcsolatáról.

A RIS és a DHCP között néha „rejtélyes" kapcsolat van. Először is minden RIS szervert fel kell venni az Active Directory hitelesített DHCP kiszolgálóinak listájára. Vagyis a Windows 2000 bizonyos szempontból a DHCP kiszolgálókhoz hasonlóan bánik az automatikus szoftverdisztribúciós feladatokat végző RIS szerverekkel. Miért is? A RIS kiszolgálók feladata a PXE ügyfelek kiszolgálása, s csodálatos módon a PXE ügyfelek DHCP csomagok segítségével kommunikálnak a RIS szerverekkel. Ezért szükséges a hitelesítés.

Lássuk a tényleges forgalmat. Feltételezzük, hogy mind a DHCP, mind a RIS kiszolgáló az PXE klienssel azonos alhálózatban van.

1. A PXE ügyfél kibocsát egy DHCP-DISCOVER csomagot, amelyben IP címet és PXE boot szerver címet kér.
2. DHCP-OFFER csomag érkezik a DHCP kiszolgálótól egy IP címmel.
3. DHCP-OFFER csomag érkezik a RIS kiszolgáló BINLSVC komponensétől a PXE szerver címmel.
4. A PXE kliens DHCP-REQUEST csomagot küld a DHCP szervernek, IP címet kér.
5. A DHCP kiszolgáló DHCP-ACK csomaggal nyugtázza a kérelmet.
6. A PXE ügyfél egy újabb DHCP-DISCOVER csomagot bocsát ki.
7. DHCP-OFFER csomag érkezik a DHCP kiszolgálótól az előző IP címmel.
8. DHCP-OFFER csomag érkezik a RIS *(BINLSVC)* kiszolgálótól a PXE szerver címmel.
9. A PXE kliens DHCP-REQUEST csomagot küld a RIS szervernek, és a PXE boot szerver címét kéri.
10. A RIS *(BINLSVC)* egy DHCP-ACK csomagot küld, amely a szerver IP címét, nevét, valamint annak az állománynak a nevét tartalmazza, amelyet a TFTP kiszolgálótól kell kérnie az ügyfélnek *(Ez a startrom.com)*.

Látható, hogy a harmadik és hetedik csomag felesleges. A harmadik csomagra sohasem fog válaszolni a PXE ügyfél, mert nem tartalmaz IP címet, míg a hetedik azért felesleges, mert IP címmel már rendelkezik a delikvens. Mindez azonban a szabványból következik. DHCP-OFFER csomagra minden DHCP kiszolgáló válaszol. *(A BINLSVC konfigurálható ennél finnyásabb módon is. A RIS-nél beállítható, hogy csak az ismert ügyfeleknek válaszoljon. Ha a PXE ügyfél nem tud ismert GUID-ot felmutatni, a BINLSVC egyszerűen hallgat, nem küld válaszokat.)*

Ha a RIS és a DHCP egy kiszolgálón dolgoznak, lényegesen egyszerűbb párbeszéd zajlik a szerver és az ügyfél között. Csupán az alábbi:

1. DHCP-DISCOVER csomagot bocsát ki az ügyfél *(IP címet és PXE boot szerver nevet keres)*.
2. DHCP-OFFER csomagot küld a kiszolgáló IP-cím ajánlattal és PXE boot szerver névvel.
3. DHCP-REQUEST kérés indul az ügyféltől a kiválasztott IP címmel.
4. DHCP-ACK nyugtázza a cím kiadását. A csomag tartalmazza az IP címet, a RIS szerver címét, és az első letöltendő állomány nevét.

A trükk annyi, hogy a DHCP megkérdezi a BINLSVC szolgáltatást, hogy kívánja-e hozzáadni a maga információit a DHCP csomagokhoz. A PXE kliens egyébként az utóbbi forgalmat szereti. Ha két DHCP kiszolgálótól is kap csomagot, akkor azt fogja minden esetben preferálni, amelyik egyben RIS szerver is. A RIS kiszolgálók terheléselosztásánál ezt figyelembe kell venni.

A fenti forgalomnál feltételeztük, hogy a három szereplő *(PXE ügyfél, DHCP és RIS kiszolgáló)* egy alhálózaton működik. Az élet azonban lehet ennél bonyolultabb, megeshet, hogy akár az egyik, akár mindkét szerver egy másik alhálózaton dolgozik. Ekkor természetesen DHCP Relay Agentekre van szükség, amelyeknek több helyre is továbbítani kell az elkapott DHCP-DISCOVER csomagot: a címkiszolgálók mellett a RIS szervereknek is értesülnie kell a PXE ügyfél indulásáról.

A Q259670 cikk egy érdekes, de a Microsoft által nem támogatott eljárást ír le. Ennek lényege, hogy három opciót be kell állítani egy adott scope-hoz *(ezek közül egyet netsh felületen keresztül)*, amelynek nyomán a DHCP kiszolgáló rögvest elvégzi a BINLSVC munkáját is, mert megadja a RIS szerver nevét, címét és a boot állomány nevét. A megoldás hátránya, hogy a RIS szervert a scope-hoz köti, ráadásul kicselezi a BINLSVC fent említett biztonsági mechanizmusát is. Viszont a megoldás teljesen hasonlatos egy BOOTP induláshoz. Mi kell egy BOOTP kliensnek? IP cím, TFTP szerver cím, boot fájl név. És mi kell egy PXE kliensnek? Ugyanez. Nohát, nohát. Akkor miért kellett megalkotni egy új szabványt, a PXE-t, ahelyett, hogy az öreg motoros BOOTP-t használták volna?

A válasz nem triviális, de valamennyire érthető. A BOOTP egy kihalóban lévő szabvány, mára már alig használják, legfőképpen pedig nem dinamikus. A DHCP ugyan dinamikus, de alapértelmezetten nem tartalmaz olyan adatokat a kliensekről, amilyenek egy PXE hostnak szüksége van. *(Láttuk, megoldható, de nem túl szerencsés)*. Kell tehát egy olyan szabvány, amely IP címet oszt, TFTP szerver nevet és boot image elérhetőséget és dinamikus. Ilyen nem volt, ezért megalkották a PXE-t.

Nem lehetett volna ezt a funkciót mégis a DHCP-be gyúrni? Talán lehetett volna, ki tudja. Az mindenesetre árulkodó, hogy a PXE nem egy RFC szabványon alapul, vagyis nem IETF alkotás. Ki tudja? Egyszer talán összeolvasztják a két eljárást – olyan sok közös vonásuk van. Egyelőre azonban ez csak találgatás. Volt a BOOTP, van a DHCP és a PXE, nekünk pedig mindegyiket tudni kell alkalmazni a maga helyén.

Lepenye Tamás, MCSE 2000
lepenyet@mal.hu

Felhasznált és ajánlott irodalom

Windows 2000 Server Internetworking Guide – Chapter 3: IP Unicast
Windows 2000 TCP/IP Core Networking Guide
RFC-951 „BOOTP Protocol"
RFC-1930 „Guidelines for creation, selection, and registration of an Autonomous System *(AS)*."
RFC-2365 „Administratively Scoped IP Multicast."
Q244036 Description of PXE Interaction Among PXE Client, DHCP, and RIS Server
Q259670 Using Dynamic Host Configuration Protocol Options 60, 66, 67 to Direct PXE Clients to RIS Servers May Fail

Szolgáltatáskiesés – a százfejű szörny

A Tech.net magazin szeptemberi számában egy nagyon érdekes cikket olvashattunk a Stratus cég ftServer megoldásáról, amely 99,999%-os rendelkezésre állást ígér. Sőt, nem csak ígéri, hanem be is tartja, hiszen a cég honlapján naponta újraszámolják a világon futó ftServerek összesített rendelkezésre állását, az pedig még ennél is jobb. Mindezt fürtszolgáltatás nélkül…

Biztonság kontra rendelkezésre állás

Az első kérdés, amely felötlött bennem, az volt: vajon mennyire biztonságosak a Stratus hardveren futó Windows 2000 vagy Windows Server 2003 operációs rendszerek? Ha ugyanis valóban 99,999%-os a szerverek rendelkezésre állása, akkor abba az évi öt percbe nem fér bele több, mint legfeljebb egyetlen újraindítás. Mi hiszünk abban, hogy egy jól belőtt Windows 2000 akár évekig is futhat, de a gyakorlati tapasztalatunk sajnos az, hogy bármilyen jó is a rendszerünk, néha elkerülhetetlen az újraindítás – külső körülmények miatt. A szoftverfoltokra és a javítócsomagokra ugyanis szükség van. Maga a gyártó ajánlja a biztonsági patchek alkalmazását, a nagyobb javítócsomagok pedig gyakran kompatibilitási feltételt jelentenek egy-egy alkalmazás futtatásához.
Meglátogattam a Stratus honlapját, hogy van-e a speciális hardvermegoldásuk mellett még valami, amivel képesek felszámolni a rendszerükben meglévő egyetlen Achilles-sarkot: a Windows 2000 újraindulási kényszerét. Kellemes meglepetés volt, hogy a fenti cikken kívül további, a rendelkezésre állást növelő szoftveres kiegészítőkről olvashattam. A **[stratus_sw]** lapon a „Hardened drivers", a „Quick dump", az „ActiveService™ architecture" és még megannyi más, a cég által fejlesztett megoldásról esik szó, ám a szoftverfoltok újraindítás nélküli alkalmazásáról nem láttam semmit. Ekkor rákeresem a Search gombbal a „patches" szóra, s lám, a harmadik találat épp az volt, ami engem érdekelt **[startus_patch]**. Ez pedig már világos beszéd: a Stratus a rendelkezésre állás növelése érdekében letesztel minden egyes kiadott Microsoft javítófoltot, és kategorizálja azokat aszerint, hogy az általa elvégezett operációs rendszerbeli módosítások és a szoftverfolt együttes használata milyen kockázatokkal jár. *(További részletek a weblapon.)* Ugyanakkor az is világossá vált, hogy a teszten átment kódot pontosan ugyanúgy kell kezelni, mint egy normális frissítést. Ha a gyártó szerint újra kell indítani a gépet, akkor újra kell indítani, nincs mese. Ettől kezdve világos, hogy vagy nem igaz a sok kilences a weblapon, vagy nem kellően naprakészek a Stratus szervereken futó operációs rendszerek. Mivel a cég szavahihetőségében semmi okunk kételkedni, kizárásos alapon a második lehetőséget kell választanunk… Tényleg?

Csigavér

Akkor most egy álommal kevesebb? Oda a remény, hogy 99,999% rendelkezésre állást lehessen Windows platformon biztosítani? Felesleges volt a Stratus sok-sok innovatív fejlesztése? Vagy, ami még rosszabb: nem az teljesül, amit ígérnek? És egyáltalán, mi a Stratus kapcsolata például a fürtökkel? Ha az egyikbe belefektettem a pénzem, akkor a másikba már nem érdemes? Vagy mind a kettőre szükség van?
Ahhoz, hogy ezekre a kérdésekre elfogulatlan és szakmailag helyes választ adjunk, azt kell tisztáznunk, mit is értünk rendelkezésre állás alatt, miben különbözik ez attól, amit a gyártók értenek a fogalmon, és milyen eszközeink vannak a rendelkezésre állás növelésére, vagy megfordítva: a szolgáltatáskiesés elkerülésére.

A rendelkezésre állás fogalma

Nos, rendelkezésre állásról a vállalatok informatikusai általában egy konkrét szolgáltatás esetén beszélnek, s annyit jelent, hogy az adott szolgáltatás *(például SAP)* a felhasználók számára elérhető, rendeltetésszerűen használható, tehát rendelkezésre áll. Mértéke %, amely a szolgáltatás rendelkezésre állásának aránya a teljes birtoklási időhöz képest.
Nem nehéz belátni, hogy az SAP, amely egy *(legfeljebb néhány)* NT service, csak akkor áll rendelkezésre, ha mindazon egyéb hardver és szoftver komponensek is működnek, amelyek közvetlenül vagy közvetve az alkalmazást szolgálják. Vagyis elérhetőnek kell lennie a felhasználó gépén kívül minden aktív hálózati elemnek, amely az SAP kiszolgálót és az ügyfelet összeköti, az SAP alatt futó hardvernek, operációs rendszernek, adatbáziskezelőnek. Emellett működnie kell egy névfeloldási rendszernek, egy hitelesítési rendszernek *(Active Directory)*, esetleg egy háttértároló rendszernek, és akkor még nem is beszéltünk olyan opcionális rendszerekről, mint a webkiszolgálók vagy a terminál szerverek. Ha pedig a végletekig szeretnénk kihegyezni a dolgot, még annak a nyomtatónak a rendelkezésre állása is számít, amellyel egy szállítólevelet nyomtatnak ki, hiszen ha nincs szállítólevél, állnak a kamionok, egyre hosszabb sorban, és teljesen mindegy, hogy működik-e az az átkozott SAP, ha nincs „output", jelen esetben egy A4-es lap, amelyet a sofőr magával visz.
Szóval ennek a borzalmasan bonyolult valaminek kell rendelkezésre állnia – a vállalati informatikusok és végeredményben a felhasználók szemszögéből. Ha egy vagy több komponens nem működik, akkor egy vagy több folyamat, művelet, „output" kiesik, hiányzik, eltűnik, – nem áll rendelkezésre.

A fenti szörnyűség már önmagában sem jó hír, de van ennél még rosszabb is. Könnyen belátható, hogy egy rendszer teljes rendelkezésre állása az alrendszerek rendelkezésre állásának szorzata. Tegyük fel, hogy van egy webkiszolgálónk. Ha egy évben egyetlen alkalommal meghibásodik a kiszolgáló egyik

alkatrésze *(pl. hálózati kártya egy napra)*, akkor a hardver mint alrendszer rendelkezésre állása 364/365, vagyis 99,72%. Ha az operációs rendszer is meghibásodik egy napra, akkor annak a rendelkezésre állása is 99,72%. Csakhogy a teljes rendszer két napot nem üzemelt, így a rendelkezésre állás már csak 99,45%. Axiómaként elfogadható, hogy egyetlen alrendszer rendelkezésre állása sem érheti el az elméleti 100%-ot, vagyis minél több alrendszerből áll a teljes üzemeltetési környezetünk, a sok egynél kisebb szám szorzata egyre kisebb végeredményt jelent. A sok kilences hamar eltűnik, ha a teljes rendszerről beszélünk.

Van tehát egy nagyon bonyolult rendszerünk, amely sok-sok alrendszerből, komponensből áll, ezért a teljes rendelkezésre állási mutatója erősen lefelé tendál, ugyanakkor vannak az üzleti felhasználók, akik viszont egyre türelmetlenebbek bármiféle kieséssel szemben. A jövőben várhatóan még sokkal bonyolultabbak lesznek az eszközeink és még türelmetlenebbek a felhasználók. Van kiút ebből a harapófogóból? Persze, hogy van, csak nem könnyű és nem mindig olcsó.

A gyártók által hirdetett rendelkezésre állás

A fenti rendelkezésre állás fogalom világos, és a gyártók által ajánlott is valami hasonló, de nem pontosan ugyanez. Minden gyártó a maga termékének rendelkezésre állását hirdeti. Ebben igazán semmi csoda nincs. Csakhogy a saját terméküket szeretik „megoldásként" kínálni, mintha az a termék egyedül képes lenne a felhasználó problémáját megoldani. A Microsoft is szívesen beszél a Windows szerverek magas rendelkezésre állásáról, ám az már csak kisebb betűkkel olvasható, hogy a hardver és egyéb komponensek magas rendelkezésre állását feltételezik. Röviden megfogalmazva: bármely gyártó bármely termékét is vásároljuk meg, a mi környezetünkben az csak egy alrendszer, egy komponens lesz, a hirdetett rendelkezésre állást is így kell tehát kezelni. Érvényes mindez a Stratus termékeire is. A felhasználó által tapasztalt stabil szolgáltatásért egyedül mi, üzemeltetők vagyunk felelősek, nem pedig a gyártók.

Még egy apróságot meg kell említeni, – cégektől teljesen elvonatkoztatva. Gyakran előfordul, hogy egy terméket magas rendelkezésre állási mutatóval hirdetnek, csak épp a szolgáltatáskiesésbe nem számítják bele a tervszerű leállásokat. Ha ilyet tapasztalunk, az csúnya csúsztatás. Kicsit olyan, mint amikor a merevlemez-gyártók kiszámítják termékük kapacitását. Valahogy a végére mindig egy kicsit kisebb, és még kisebb értékeket kapunk, ha mi formázzuk meg azokat a lemezeket. Hiába, – mindenki felfelé kerekít.

Harc a hibák ellen

Műszaki szakembereknek gyakran feláll a szőr a hátán, ha a vezetők stratégiáról kezdenek el beszélni, mert az gyakran csak üres frázisokat jelent. Mi most a stratégia szót arra használjuk, hogy „hiba" ellen módszeresen felkészülhessük. Józanul végiggondolva a következő fázisokat különböztethetjük meg:

1. **A hiba még nem létezik.** Ezt az időszakot arra lehet felhasználni, hogy mind műszaki beruházásokkal, mind szervezési lépésekkel felkészüljünk a hiba megelőzésére, vagyis, hogy ne is következzék be.
2. **A hiba bekövetkezik.** Használhatnánk itt a Murphy-féle elcsépelt törvényeket, de a fenti okfejtésből világosan adódik, hogy a hibák bekövetkezése elkerülhetetlen. A hibák egy darabig felderítetlenek. Mi viszont megtehetjük, hogy olyan segédeszközöket működtetünk, amelyekkel a hibák bekövetkezése és észlelése közötti idő lerövidíthető.
3. A hiba észlelése után fel kell állítani a helyes **diagnózist**. Csak miután tudjuk, mi a pontos hiba, akkor kezdhetünk hozzá az elhárításhoz szükséges teendőkhöz.
4. A helyes diagnózis megállapítása után két fontos dolog következik. Az előre elkészített tervek szerint **cselekedni kell, hogy a szolgáltatást** – akár a hiba elhárítása nélkül – **helyre lehessen állítani**. Itt kapnak fontos szerepet a tartalék rendszerek, egy példa ezek közül a fürtszolgáltatás. A másik fontos teendő természetesen a hiba kijavításának megkezdése. A javítás alatt előfordul, hogy a szolgáltatás nem minden paraméterében felel meg a hiba előttinek. A második feladat természetesen a hiba tényleges elhárítása.
5. A hiba kijavítása a megoldással vesz fordulatot, de nem ott ér véget. A megoldás után ugyanis még helyre is kell állítani a szolgáltatást.

A főbb fázisokat és egymáshoz való viszonyukat az alábbi ábra szemlélteti

A hibák életciklusa

A rendszergazdák számára minden olyan eszköz, szoftver, technológia, amely a szolgáltatáskiesés idejét, vagyis a helyreállítási időt – felfedezés, reagálás, javítás – csökkenti, hasznos lehet, mert a teljes rendszer rendelkezésre állását növeli. Mielőtt az ftServer-hez és egyéb újításokhoz visszakanyarodunk, érdemes megvizsgálni, milyen elméleti meggondolások segíthetik a rendelkezésre állás növelését.

Virtualizáció és redundancia

A rendelkezésre állást sok minden segítheti, akár a lelkiismeretesség is, mégis van két olyan elvi-technológiai megoldás, amelyek a leghasznosabbak. Közülük talán a virtualitás a kevésbé ismertebb.

A virtualitás mint megoldás abból a meggondolásból származik, hogy a hiba bizonyosan be fog következni, ám jó esélyünk van rá, hogy ezt a felhasználó elől elfedjük. Tipikus példa a virtualitásra a RAID tömb. Amikor egy merevlemeztömböt kialakítunk, a tényleges lemezek fizikai geometriája helyett logikai *(virtuális!)* köteteket hozhatunk létre. Elfedjük a tényleges valóságot az operációs rendszer elől. Hasonló a helyzet a fürtöknél: az ügyfelek nem az egyik, vagy a másik node-hoz kapcsolódnak, hanem egy virtuális szerverhez. Számukra az a kiszolgáló éppen olyan, mint a többi, s csak a rendszergazdák tudják, hogy milyen architektúra dolgozik a háttérben. Az ftServerek is erősen építenek a virtualitásra, mint elméleti megoldásra. Az operációs rendszer „nem látja" a tényleges valóságot, az igazi hardverelemeket. Sőt, ha belegondolunk, maga az operációs rendszer is használja ezt a

technológiát. Az egyes alkalmazások nem látják, hogy a memória, amelyet használnak, fizikailag létezik, vagy csak a háttértár egy kijelölt része. A DOS alkalmazások szentül meg vannak győződve arról, hogy ők az egyedüli futó alkalmazások a rendszeren, mert a Windows 2000 elrejti a valóságot előlük egy virtuális gép segítségével.

A virtualitás azonban igazán a redundanciával együtt szolgálja hatékonyan a rendelkezésre állást. A RAID5 tömbökben mindig legalább eggyel több lemez van, ezzel biztosítva, hogy a kontroller a hibákat kijavíthassa és elrejthesse az operációs rendszer elől. A virtuális gépeknek szükségük van egy másik node-ra, hogy vész esetén újraindulhassanak stb.

A virtualitásnak és a redundanciának „két gyermeke" van: a hibatűrés és a nagy rendelkezésre állás. A hibatűrés olyan technológiai megoldás, amely során a virtualizáció révén a meghibásodás magasabb rétegek elől teljes mértékben eltüntethető. A RAID5 és az ftServerek hibatűrő megoldások. A magas rendelkezésre állás ezzel szemben nem tünteti el a hibát, az láthatóvá válik, ám lecsökkenti a felfedezési és reagálási időt. Tipikus példa a fürtszolgáltatás. A virtuális szerverek költözése alatt a szolgáltatás nem érhető el, de a kiesés minimális az egyetlen szerver esetéhez képest.

Természetesen a két fogalomnak nem kell feltétlenül együtt járnia. A DNS vagy az AD kiszolgálók megkettőzése nem jár együtt virtualizációval, mégis növelhető a szolgáltatás biztonsága.

ftServerek kontra fürtépítés

A fogalmak megértése után láthatjuk, hogy nincs sok értelme összehasonlítani a fürtöket az ftServerekkel. Mert az igaz, hogy mindkét technológia a rendelkezésre állást növeli, csakhogy – Héraklész egyik próbájára utalva – a hidra más-más fejét vágják le. Az ftServerek hibatűrő technológiájukkal a bekövetkezés valószínűségét csökkentik, a fürtök viszont a maradó kockázat – tehát a mégiscsak bekövetkezett hiba esetén nyújtanak gyors megoldást. Az ftServerek remekül működnek mindaddig, amíg van áram, ha azonban egy egész telephely marad tápellátás nélkül, akkor egy földrajzilag elosztott fürt jelentheti a megoldást.

Mit jelent mindez számunkra? Több mindent is. Először is már tudjuk, hogy a Stratus a saját kiszolgálóinak rendelkezésre állását mutatja a weblapján, nem az ügyfelei teljes rendelkezésre állását. Másrészt a Stratus megoldása nem csodaszer, mert csodaszer nincs. Egyetlen eszköz nem elég a százfejű szörnyeteg, a szolgáltatáskiesés legyőzésére. Minden technológiának megvan a helye a nap alatt. A Stratus ajánlata sem csupán egy speciális hardver, hanem – ahogy azt a szeptemberi cikkben olvashattuk – rendszerfelügyelet is. Nem más a szándékuk ezzel, mint a hibák korai detektálása, vagyis az észlelési- és reakcióidő csökkentése, minimalizálása.

Ha valaki engem kérdezne, a Stratus megoldást stabil, nem változó, nagyon egyszerű *(értsd: kevés szolgáltatást nyújtó)* környezetben használnám. Minimalizálnám a biztonsági kockázatokat, letiltanám a nem használt szolgáltatásokat, függetleníteném a rendszert a hitelesítési és névfeloldási problémáktól *(néha a munkacsoport is elegendő!)*. Akár egy tűzfallal is védeném, persze redundáns tűzfallal. A javítócsomagokat különösen megválogatnám, és kizárólag akkor alkalmaznám őket, ha a konkrét rendszer üzemeltetési tapasztalata alapján indokolt.

A Microsoft cluster megoldása csupán egyike a szoftvercég rendelkezésre állást növelő technológiáinak. A felhasználó szemszögéből nem jelent hibatűrést, de a szolgáltatás kiesését minimalizálja. Nem igényel speciális hardvert, pontosabban a szükséges alkatrészek nyílt szabványok alapján több gyártótól is beszerezhetők. A nem tervezett leállások mellett alkalmas a tervezett karbantartás alatt is a szolgáltatás folyamatos biztosítására. Tervezése és kialakítása körültekintést igényel, alapos kompatibilitási teszteket kell végezni. Az azonos telephelyű node-ok mellett mód van földrajzilag elosztott rendszerek kialakítására is.

Üzleti megfontolások – végül és elsősorban

Mielőtt bárki elszalad magas rendelkezést ígérő megoldásokat vásárolni, ajánlom, hogy végezzen megtérülési számításokat. Azt kell megbecsülni, vajon a szolgáltatás kiesése kerül többe, vagy a technológia, amellyel az üzletmenet folytonossága biztosítható. Nem mindig és nem mindenhol éri meg drága megoldást választani. Lehetséges, hogy üzleti megfontolások után csupán egy jól megtervezett mentési rendszert, esetleg egy katasztrófatervet készítünk, netalán szerződést kötünk egy vagy több hardverszállítóval, hogy tartalékoljanak nekünk alkatrészeket. Miért? Mert ez az olcsóbb.

Nem született volna meg azonban ez a cikk, meg a korábbiak sem, ha a mérleg nyelve nem billenne egyre gyakrabban a szolgáltatások kiesését megakadályozó újítások felé. Ebben az esetben viszont ajánlom, hogy minden rendszergazda kösse fel a nadrágját, és küzdjön meg derekasan a rendszere Achilles-sarkaival és hibáival. Olyan lesz a küzdelem, mint a már említett Héraklész próbája a lernai háromfejű hidrával. Amikor ugyanis a hidra egyik fejét levágta, másik kettő nőtt a helyére, a középső feje pedig halhatatlan volt. A derék Héraklész rendületlenül vágta a hidra fejeit, ám csak nem bírt vele. Végül Lolaosz, az unokaöccse segített neki, együtt győzték le a szörnyeteget.

Ha szeretnénk levonni a tanulságot, annyit mondhatunk, ronda dögökkel egyedül ne kezdjen az ember, még akkor sem, ha félisten. A bonyolult rendszerek pedig ronda dögök.

Lepenye Tamás, MCSE 2000
lepenyet@mal.hu

Felhasznált és ajánlott irodalom

TechNet CD: IT Solutions – Availability Management
Szabó György: Mitológiai kislexikon

Tanúsítványkiadók a Windowsban

Qualified Subordination

A Qualified Subordination *(kb. „minősített altanúsítványkiadók")* fogalom egy új eszköz- és eljáráscsomag fedőneve a Windows Server 2003 Enterprise Edition nyílt kulcsú szolgáltatásai között. Segítségével a tanúsítványkiadók hierarchiájának ágaban szereplő altanúsítványkiadók működését minden eddiginél finomabban testreszabhatjuk.

A CAPolicy.inf fájl

Amikor egy tanúsítványkiadó szolgáltatást telepítünk, a tanúsítványkiadó *(CA)* nevén, valamint a kulcshossz, kulcstárolási mód kiválasztásán kívül túl sok szabadságunk nincs. Az altanúsítványkiadók „automatikusan" generálják a tanúsítványuk elkészítéséhez szükséges kérést, majd azt elmentik, vagy továbbítják egy elérhető, vállalati tanúsítványkiadó felé. A tanúsítványkérés – és így persze a kérés alapján készített tanúsítvány is – tartalmát és jellemzőit azonban „kívülről", mi is befolyásolhatjuk; erre „való" a CAPolicy.inf fájl.
Ez a konfigurációs állomány a Windows rendszerkönyvtárában *(alapértelmezésben C:\Windows)* található – vagy ha még nincs ott, a CA telepítésekor létrejön. Ha a tanúsítványkiadó telepítésekor, vagy a saját tanúsítványának megújításakor ez a fájl létezik, tartalmát a CA feldolgozza, és hozzáigazítja a készítendő tanúsítványkérést is. A fájl – jó .inf fájlokhoz méltóan – kötelezően mindig az alábbi két sorral kezdődik:

```
[Version]
Signature="$Windows NT$"
```

Ezt követik a különféle beállítások. Ezek közül mutatunk be néhányat a következőkben:

Issuer Statement

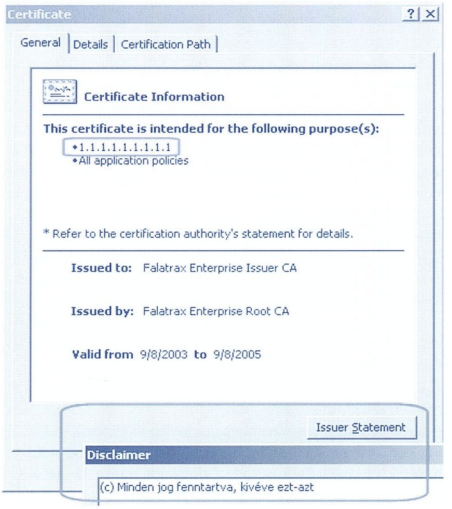
A „Issuer Statement" jogi szempontból fontos szövegrész, amellyel sok tanúsítványban találkozhatunk. Feladata az, hogy a szülő CA által kiadott tanúsítványokban felhívja a figyelmet a tanúsítványkiadóval kapcsolatos jogi feltételekkel, korlátozásokkal kapcsolatban. Az Issuer Statement szöveges és/vagy URL formában kerülhet be a tanúsítványokba, ehhez a tanúsítványkiadó telepítése *(vagy kulcsának megújítása)* előtt a CAPolicy.inf fájlba fel kell vennünk azt:

```
[CAPolicy]
Policies=LegalPolicy

[LegalPolicy]
OID=1.1.1.1.1.1.1.1.1.1
URL="http://ca.falatrax.hu/legalpolicy.htm"
Notice="(c) Minden jog fenntartva, kivéve ezt-azt"
```

Az OID mező a policy-bejegyzésben kötelező, és az itt látott érték természetesen csak példaként szolgál *(lásd a cikksorozat korábbi részében az OID leírását).*

Az így kért tanúsítvány elkészülte után a jogi szöveg elolvasható, ha a tanúsítvány tulajdonságlapjának első oldalán az „Issuer Statement" gombra kattintunk:



Issuer Statement a tanúsítványkiadónk tanúsítványában

Ugyanitt láthatjuk, hogy a CAPolicy.inf fájlban definiált Issuer Statement OID is megjelenik.
CRL információk, a tanúsítványkiadó tanúsítványa, kulcshossz, tanúsítvány érvényessége
A CAPolicy.inf fájlban meghatározhatjuk azt is, hogy a leendő tanúsítványban milyen CRL disztribúciós pontok szerepeljenek, valamint azt, hogy hol érhető el a tanúsítványt készítő CA saját tanúsítványa *(Authority Information Access, AIA mező)*:

```
[CRLDistributionPoint]
URL="http://CompanyWebSite/Public/myCA.crl"

[AuthorityInformationAccess]
URL="http://CompanyWebSite/Public/myCA.crl"
```

Hasonlóképpen, ha a szükség úgy hozná, a **[certsrv_server]** szakaszban meghatározhatjuk a leendő tanúsítvány egyéb adatait, például:

- RenewalKeyLength: megújítás esetén az újonnan generált kulcs hossza
- RenewalValidityPeriod, RenewalValidityPeriodUnits: a leendő tanúsítvány érvényességi ideje
- CRLPeriod, CRLPeriodUnits, CRLDeltaPerion, CRLDeltaPeriodUnits: a CRL frissítési időszak meghatározása

```
[certsrv_server]
RenewalKeyLength=2048
RenewalValidityPeriod=Years
RenewalValidityPeriodUnits=5
CRLPeriod=Days
CRLPeriodUnits=2
CRLDeltaPeriod=Hours
CRLDeltaPeriodUnits=4
```

Basic Constraints

A CAPolicy.inf bemutatása után most lapozzunk vissza gondolatban a tanúsítványok mezőit leíró *(néhány hónappal ezelőtt megjelent)* oldalakra! A mezők között volt egy, a Basic Constraints nevű, amely csak a tanúsítványkiadók saját tanúsítványaiban szerepel, ott viszont kötelező jelleggel – tulajdonképpen azt jelzi, hogy az adott tanúsítvány gazdája egy tanúsítványkiadó szervezet.

A tanúsítványkiadók ismert hierarchiája úgy épül fel, hogy a fa alján, a „sűrűben" működő CA-k tanúsítványait egy másik CA, azokét egy újabb, majd a hierarchia mélységétől és bonyolultságától függően újabb és újabb CA adja ki – egészen addig, míg el nem jutunk ahhoz a CA-hoz, akinek saját maga által aláírt tanúsítványa van – ő a gyökértanúsítványkiadó *(Root CA)*. Mivel elvileg bármelyik tanúsítványkiadó osztogathat tanúsítványt újabb és újabb gyermek CA-knak, előfordulhat, hogy a fa kezelhetetlenül bonyolulttá – és nem mellékesen – jogi úton zavarossá válik. Ekkor hasznos, ha a tanúsítványkiadó-tanúsítványt kiadó tanúsítványkiadók *(;-))* kezét egy kicsit meg tudjuk kötni, ebben pedig pontosan a Basic Constraints segít nekünk.

A Basic Constraints egyik almezője a Path Length Constraint, ami – ha van értéke, és nem „None" – azt határozza meg, hogy a kérdéses tanúsítvánnyal rendelkező CA adhat-e ki, és ha igen, milyen „mélységig" CA-tanúsítványokat. Ha a CA tanúsítványában a Path Length Constraint értéke 0, az a CA gyermek-CA tanúsítványt már nem állíthat ki. Ha mondjuk 2, akkor a CA és még az ő gyermeke is adhat ki CA-tanúsítványt *(mivel a gyermektanúsítványok Path Length Constraint értéke mindig eggyel kisebb mint a szülőé)*, de a 3. generáció CA-i már nem. Ha a Path Length Constraint üres *(„None")*, nincs korlátozás.

Próbaképpen – egyelőre vállalaton belül – hozzunk létre egy pici tanúsítványkiadó-fát *(egy root és egy kiadó CA-val)*, ahol a gyermek-CA-nak már nem engedjük meg további CA-k tanúsítványainak kiadását. *(Vállalaton belül a végtelen és a 0 a két „beállítható" érték, mert azt – látni fogjuk – a tanúsítvány Active Directory-beli sablonja definiálja. Vállalatok között a Path Length egyéb értékre is beállítható).*

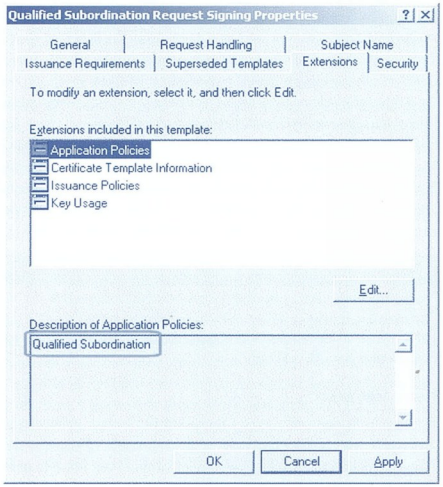
Mivel azonban ez már a Qualified Subordination, egy kis előkészületre is szükség lesz. Mindenekelőtt, szükség lesz egy vagy több Windows Server 2003, Enterprise Edition kiszolgálóra, a Qualified Subordination – a szerkeszthető tanúsítványsablonok miatt – ugyanis csak ott működik. Ezután, vállalatszerte letiltjuk majd a régi jó, megszokott altanúsítványkiadó-tanúsítvány sablont, és egy új, v2.0 tanúsítvány lép majd a helyébe. Emellett létrehozunk egy új tanúsítványsablont, amelyet a Qualified Subordination-kérések digitális aláírásához használunk – e nélkül az aláírás nélkül az új, jó kis CA-tanúsítványunkból senki sem kaphat!

A meglévő SubCA tanúsítványsablon letiltása

A Root CA kiszolgálóján nyissuk meg a Certification Authority MMC konzolt, majd a tanúsítványkiadó konzolfájában kattintsunk jobb gombbal a Certificate Templates sorra, és válasszuk a Manage... parancsot. Erre egy új MMC ablak nyílik meg, benne a címtárban található tanúsítványsablonokkal. Kattintsunk a Subordinate Certification Authority sablonra, majd a tulajdonságlapjának Security oldalán kattintsuk be az Authenticated Users – Enroll jog tiltását *(Deny)*! Ezzel a „régi" sablont kivontuk a forgalomból.

Qualified Subordination aláíró tanúsítványsablon létrehozása

Hozzuk most létre a kérés aláírásához szükséges tanúsítvány sablonját! Az Administrator sablonon jobb gombbal kattintva válasszuk a Duplicate Template parancsot, a létrejövő új tanúsítványsablonnak pedig adjuk a hangzatos „Qualified Subordination Request Signing" nevet.



A Qualified Subordination Request Signing sablonban egyetlen Application Policy szerepel: a Qualified Subordination

Miután létrehoztuk és elneveztük az új tanúsítványsablont, lépjünk az Extensions oldalra, ott pedig az Application Policies tartalmából töröljünk mindent, és vegyük fel a Qualified Subordination policy-t. Ezután a Security oldalon ellenőrizzük, hogy csak megfelelő felhasználók *(mondjuk az Administrators csoport)* rendelkezzen a sablonra Enroll joggal.

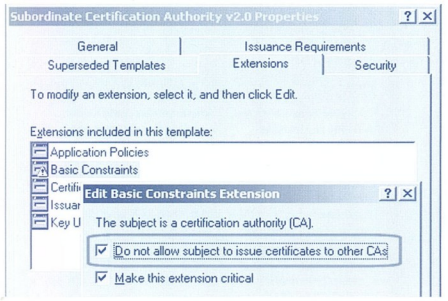
Új gyermektanúsítványkiadó tanúsítványsablon létrehozása

A következő lépés a letiltott SubCA sablon utódjának elkészítése. Ehhez a fentihez hasonló módon duplikáljuk a régi SubCA sablont, az újszülöttet pedig nevezzük el Subordinate

Certification Authority v2.0-nak. Ezután itt két dolgot kell módosítanunk:

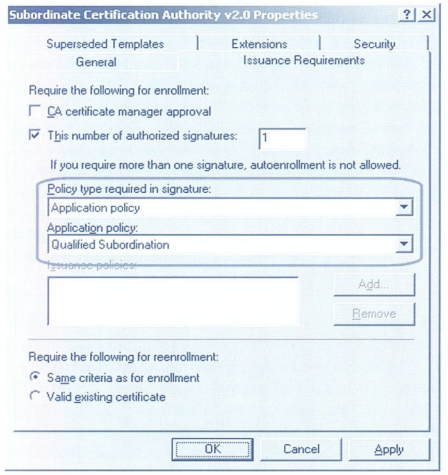
- A Basic Constraints bővítményben a Path Length Constraint értékét 0-ra állítani
- A tanúsítvány kéréséhez előírni a digitális aláírás szükségességét

A Basic Constraints beállításait a tanúsítványsablon tulajdonságlapjának Extensions oldalán találjuk:



A Path Length Constraints 0-ra állítása

A Do not allow subject to issue certificates to other CAs pipa elhelyezése a háttérben 0-ra állítja a sablon alapján kiadott tanúsítványok Path Length Constraints értékét.
Az aláírásigényt pedig – már ismert módon – az Issuance Requirements oldalon nyújthatjuk be:



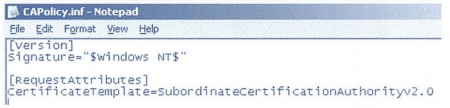
A tanúsítvány kéréséhez Qualified Subordination Application Policy-t tartalmazó tanúsítvánnyal történő digitális aláírás szükséges

Egy nagyon fontos momentumról ne feledkezzünk meg: ez a tanúsítványsablon a letiltott SubCA sablonból készült, ezért „örökölte" annak biztonsági beállításait is! Úgyhogy mielőtt továbblépnénk, a mellékhatások elkerülése érdekében töröljük az új tanúsítványunk Security oldalán az Enroll tiltó jogát!

Ezzel a gyökértanúsítványkiadót előkészítettük. Most „üljünk át" gondolatban a gyermek CA-hoz, és kezdjünk munkához!

Gyermek-CA tanúsítványának telepítése / frissítése

Most két lehetőségünk van: vagy még a CA telepítése előtt létrehozzuk a CAPolicy.inf fájlt, vagy pedig a CA-t telepítjük *(tanúsítványt a gyökér-CA-tól úgysem kaphat, hiszen a SubCA sablon le van tiltva, az új sablonról pedig egyelőre rajtunk kívül senki sem tud)*, és a CA tanúsítványának „megújítását" választjuk. Mindegy, hogy melyiket választjuk, először is a CA-val közölnünk kell, hogy nem a SubCA-ból, hanem egy attól különböző tanúsítványsablonból kell tanúsítványt kérnie. Ehhez *(és ebben az esetben csak ehhez)* a CAPolicy.inf fájlt kell szerkesztenünk:

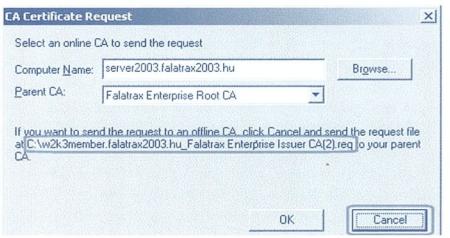


A sablon nevét a CAPolicy.inf fájl [RequestAttributes] szakaszában állíthatjuk be

Ha most megpróbálnánk megújítani a tanúsítványkiadónk tanúsítványát, az a sablont már megtalálná, ennek egyik bizonyítéka az is, hogy a kérés befejeztével csúnya hibaüzenet várna:

A kérés digitális aláírása nélkül márpedig tanúsítvány nem jár! – üzeni a Root CA

Úgyhogy nem tehetünk mást, mint a tanúsítvány-megújító folyamat vége előtt egy pillanattal *(a tanúsítványkiadó kiválasztásának ablakában)* megállunk, a kérést fájlba mentjük, és átvesszük a parancsnokságot.

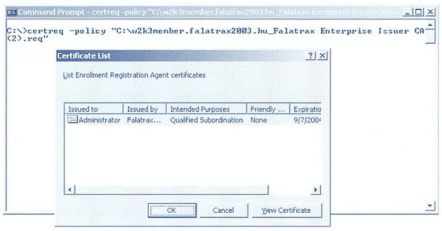


A kérés mentéséhez kattintsunk a Cancel gombra. A kérés fájlnevét az ablak közepén megtaláljuk

Jegyezzük meg ezt a fájlnevet, mert a továbbiakban ezzel fogunk dolgozni.

A tanúsítványkérés digitális aláírása

A digitális aláíráshoz mindenekelőtt szükség lesz egy aláíró tanúsítványra. Ez a tanúsítvány – és a vele készült aláírás – bizonyítja majd, hogy a gyermek CA tulajdonosa jogosult a tanúsítványláncba léptetni kiszolgálóját. Valós esetben az aláíró tanúsítvány valószínűleg valakinek a személyes kulcstároló eszközén, smart cardján foglal helyet, a példa kedvéért most kérjünk egyet a gyermek CA Administrator felhasználójának *(jelentkezzünk be a Root CA webes felületére, és válasszuk ki a Qualified Subordination Request Signing tanúsítványsablont).*

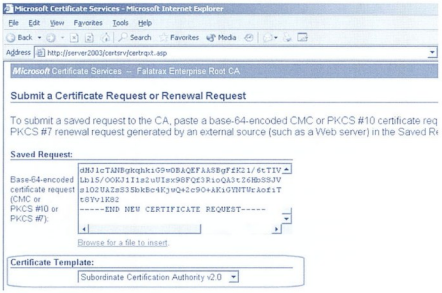


A kérés aláírásához ki kell választanunk az aláíró-tanúsítványt

A fájlba mentett kérést a

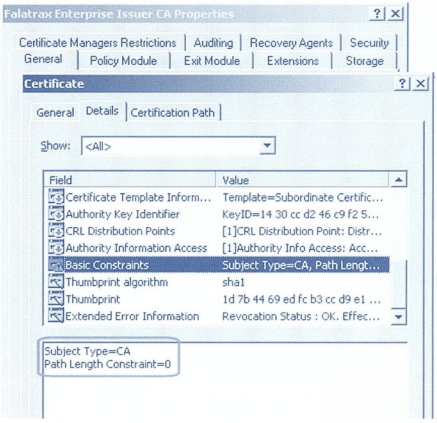
```
certreq -policy <kérésfájl.req>
```

paranccsal írhatjuk alá digitálisan. A parancs kérni fog egy .inf fájl nevet, itt adjuk meg a CAPolicy.inf-et, azután a megjelenő ablakban válasszuk ki az aláíró tanúsítványt. Az aláírt kérést mentsük ismét fájlba – e fájl tartalmát kell majd a gyökér CA webes felületén elküldenünk.



A webes kérésbe másoljuk be az elmentett, aláírt kérésfájl tartalmát és ne felejtsük el kiválasztani a megfelelő tanúsítványsablont!

A gyökér CA ezt a kérést már szívesen kiszolgálja, az eredményként keletkező .cer fájlt mentsük el, majd a gyermek-tanúsítványkiadó MMC konzoljában válasszuk az Install CA certificate... parancsot. Itt adjuk meg az újonnan készült .cer fájl nevét és készen is vagyunk.



A gyermek-tanúsítványkiadó tanúsítványa: látható a Path Length Constraint 0 értéke

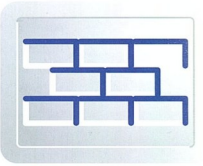
A tanúsítványkiadónk ezennel elkezdheti a munkát, belépett a tanúsítványkiadók hierarchiájába, de a tanúsítványában szereplő korlátozásoknak „köszönhetően" csak felhasználói tanúsítványokat adhat ki.

A következő részben folytatjuk a Qualified Subordination bemutatását, a „maradék" három korlátozó policy-vel. Most jön még csak a java!

Folytatjuk...

Fülöp Miklós
mick@inetcom.hu

Találd el a felhasználó igényeit!



Igényfelmérés, követelményanalízis

Hiába vagyunk a piac legnagyszerűbb fejlesztői, hiába találjuk ki a leghatékonyabb algoritmusokat, hiába dolgozunk a legmodernebb technológiákkal, ha egy dolgot nem tartunk szem előtt: a fő cél az ügyfelek elégedettsége.

Bizonyára mindenki ismeri azt a szituációt, amikor sok-sok átvirrasztott éjszaka, a gép mellett töltött hétvégék után büszkén rohanunk ügyfelünkhöz, hogy időben odaérjünk a 10 perce még warningokkal teli, de az utolsó pillanatban valami csoda folytán mégis forduló és futó, kész alkalmazással. Büszkén lépünk be az ajtón, beesett szemünk álmosan pislog, de csillog a lelkesedéstől, hogy kijelenthetjük: készen vagyunk!
Az ügyfél visszafogottan örül. Elővesszük táskánkból a notebook-ot, hogy megmutathassuk, milyen is lett a legújabb szülemény. Ám ahogy a program elindul, az ügyfél egyre sápad, majd hirtelen vörösbe csap át arcszíne, és bármilyen lelkesen mutogatjuk is a funkciókat, egyszer kitör belőle: „De hiszen ez így nem jó, mi nem ezt kértük!"

Ki a „hibás"?

Ilyenkor persze elkezdődnek a csatározások, mindenki a saját álláspontját védi: a megrendelő váltig állítja, hogy ő elmondta, mit szeretne, és nem érti, hogyhogy nem voltunk képesek megérteni?! Mi viszont abban vagyunk egészen biztosak, hogy egészen pontosan azt valósítottuk meg, amit kértek tőlünk, és nem értjük, miért változik folyamatosan a vevő igénye az időjárásnak és a csillagok állásának megfelelően.
Ha ilyenkor külső szemlélőként tudnánk vizsgálni a dolgokat, érdekes tanulságokat szűrhetnénk le *(azon kívül persze, hogy pszichológiai szempontból magunkat is jó lenne időnként kívülállóként látni)*. Az esetek többségében azt tapasztalnánk ugyanis, hogy tulajdonképpen mindenkinek igaza van…

Kommunikációs hiányosságok

A probléma gyökere tulajdonképpen abból ered, hogy nem vagyunk egyformák. A megrendelő többnyire nem informatikus szakember, mi pedig nem értünk az ő szakmájához. Így aztán nem is csoda, ha az egyeztetések során elbeszélünk egymás mellett, és nem értjük, ami a másik számára triviális. Úgy tűnhet tehát, hogy egy tolmácsra lenne szükségünk, aki ért „informatikusul" is, és az adott megrendelő szaknyelvén is: gyógyszerészül, közgazdászul, politikusul, távközlésül stb.
Ilyen emberek azonban vagy nem léteznek, vagy számunkra elérhetetlenek, így csak magunkra hivatkozhatunk. A megrendelő általában nem ismeri fel a közös nyelv megtalálásának szükségét, mert úgy érzi, teljesen triviális, amit mond, és nem is érti, hogy lehet nem érteni őt. Néha megpróbál ugyan informatikus kifejezéseket használni, de sok esetben ebből is csak félreértések vagy mosolyogtató élmények születnek *(pl.: „a rendszer lefagyás-szerű dolgot szimulál…" ☺)*. Ezért nekünk kell felkészülni az ilyen típusú problémák kezelésére még akkor is, ha ez nem egészen mérnöki típusú feladat.
Így a felhasználói igények felmérésénél nagyon oda kell figyelnünk, mit és hogyan rögzítünk. Ez azonban még mindig nem elég: a fejlesztés során folyamatosan egyeztetni kell a felhasználóval, hogy az adott modult, funkciót valóban így képzelte-e el, hiszen menet közben sokkal egyszerűbb *(és olcsóbb)* módosítani a rendszert, mint az átadás után – vagy helyett.
Többször tanúja voltam már annak, hogy a követelmények nem megfelelő rögzítése esetén maradtak „lyukak", amelyeket a projekt végén a felhasználó természetesen megpróbált kihasználni *(nem feltétlenül rosszindulatból, csupán saját akarata érvényesítése céljából)*. Ilyenkor szokott az történni, hogy a zárás előtt feláll a vevő, és közli, hogy ő tulajdonképpen még ezt és ezt a dolgot beleértette a specifikációba, amelyben ez nem szerepel ugyan, de ki sem zárja azt. És jönnek az éjszakába nyúló alkudozások: hatásköre-e még ennek a projektnek *(és a hozzá tartozó költségvetésnek)* ez a módosítás/kiegészítés *(a megrendelő szerint: hibajavítás)*, vagy már csak egy következő fázisba fér bele?

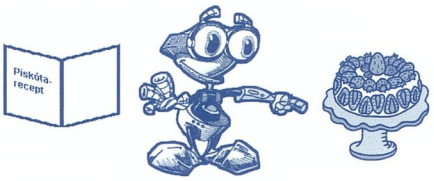
Nehéz megfogalmazni, hogy mi is kell ahhoz, hogy megértsük az ügyfelet, és megértessük magunkat vele. Egyrészt kell egyéni adottságok egész sora, amelyek fejleszthetők ugyan, de a szó hagyományos értelmében nem tanulhatók; másrészt elsajátítható ismeretek, tapasztalatok is szükségesek.
Természetesen ezen ismeretek megszerzéséhez idő kell. Egyrészt áldozni kell az elméleti tudás megszerzésére, másrészt a gyakorlatban is el kell sajátítani, ami elméletileg már a fejünkben van, hiszen az kb. annyit ér gyakorlati tapasztalat nélkül, mint ha valaki oda-vissza bemagol egy szakácskönyvet, amely alapján még soha nem főzött – így azt sem tudja, melyik recept rejteget finom süteményeket, és melyiket kell módosítani valamelyest – esetleg melyik az, amelyik teljesen használhatatlan.

Az ügyfél-kommunikáció

Az ügyféllel folytatott kommunikáció egyik alapfeltétele, hogy valamelyest értsünk a szakterülethez, amelyet ő képvisel. Természetesen nem kell profinak, szakértőnek lennünk, hiszen akkor mi ülnénk az ő pozíciójában, viszont elengedhetetlen, hogy a projekt során egy nyelvet beszéljünk. Ebben elsősorban maga a megrendelő nyújthat nagy segítséget számunkra, viszont az együttműködéshez a mi kezdeményezésünkre van

szükség. A projekt előkészítése során kérdezzünk rá minden kifejezésre, amit használ, esetleg kérjük meg, állítson össze egy szótárat számunkra. Egy másik lehetőség, hogy amennyiben írásos anyag áll rendelkezésünkre a témában, annak elolvasása során mi magunk jegyzeteljük ki a kérdéses szavakat, kifejezéseket. Ez a szótár természetesen folyamatosan bővül az idő múlásával. Egyetlen dolgot tartsunk szem előtt: kérdezni nem szégyen, sőt! Egykori filozófiatanárom szerint aki nem tud kérdezni, az nem is érti, miről van szó. Mennyire igaza volt!
Persze nem elég a megrendelő szavainak megértése, azzal is tisztában kell lenni, egy-egy mondatot hogyan ért, mit gondol. Ehhez fontos igazán a kérdezés! Mindig tartsuk szem előtt, hogy a felhasználó csak körvonalakban tudja, mit szeretne, és még azt is csak korlátozott módon tudja megfogalmazni!
Tegyük fel például, hogy a megrendelő egy olyan robotot szeretne felprogramoztatni velünk, amely sütemények elkészítésére képes. Hogy egyszerűbb legyen a dolog, a robot „gyermekkorában" még csupán piskótatésztát képes sütni, tudását egy későbbi fázisban szándékozunk továbbfejleszteni. Lássuk először azt a megoldást, amikor nem kommunikálunk eleget az ügyféllel.

Két mondatban elmeséli, hogy ilyen robotot szeretne, és a fantáziánk azonnal beindul: van némi fogalmunk arról, hogyan készül a piskótatészta, ezért nem is firtatjuk tovább, azonnal nekiállunk fejleszteni a rendszert. A robot egyre ügyesebb, csodálatos tésztát kever, szép sárgára kisüti – mehetünk a megrendelőhöz, készen vagyunk.



A piskótakészítés folyamata

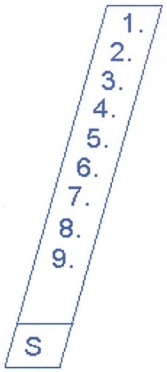
Most jön az, amit nem várunk: a cukrász ügyfél halálsápadtan közli, hogy a piskótatészta nem így készül, mert ő még citrom héját is szokott reszelni bele, és különben is, ezen a hőfokon nem jó sütni, hiszen ha 5 fokkal melegebb a sütő, sokkal szebben feljön a tészta, és a színe is szebb lesz.
Rendben, hazamegyünk, leprogramozzuk az újabb igényeket. Működik, mehetünk a megrendelőhöz, készen vagyunk... Ezt a lépést háromszor-négyszer megismételve eljutunk odáig, hogy a vevő hajlandó lesz átvenni a robotprogramot. Elégedetten dőlünk hátra, elkezdjük tervezgetni, mit is kezdünk azokkal a milliókkal, ami most a számlánkra érkezik majd. Már-már elnyom bennünket az álom, azonban ekkor megszólal a telefon. Cukrász barátunk sikítva közli, hogy a robot elejtett egy tojást, és ettől teljesen összezavarodott, most éppen a fokhagymasót önti kilószámra a tésztába.
Ja tényleg, a hibakezelés valamiért kimaradt, hiszen olyan triviális egy piskótatésztát megsütni, eszünkbe sem jutott... Újra nekiállunk hát programozni, és próbáljuk összeszedni, milyen hibákat követhet el a robot, illetve milyen külső tényezők befolyásolhatják a piskóta sikeres elkészülését *(például ha egy tojás már záp, nem túl szerencsés beleütni a tésztába)*.
Mindezek az újabb és újabb iterációk sokkal több időt, energiát vesznek igénybe, mintha már a kezdetektől figyelembe vettük volna ezeket a tényezőket. Ezt valahogy érezzük, viszont azt nem tudjuk megmondani, mi az, amit másképp kellett volna csinálnunk, hiszen az ügyfél tehet mindenről, ő az, aki nem mondott el mindent részletesen. Honnan kellene ilyen pontosan tudnunk egy piskótasütés minden csínjátbínját...?
Leülünk tehát gondolkodni, hogy lehet az, hogy mindig van valami félreértés a megrendelő és közöttünk, nem volt még egyetlen olyan projektünk sem, ahol ne történt volna hasonló eset... Kétségbeesésünkben rábukkanunk a **[tervbaki]** URL-en található képsorozatra, és hangos üdvrivalgásban törünk ki, látva, hogy ez nem csupán számunkra okoz problémát...

A megoldás

A megoldást az jelentheti, ha a megrendelő igényeit pontosan rögzítjük, mégpedig úgy, hogy kellően formális legyen ahhoz, hogy később ez alapján meg tudjuk tervezni a rendszert, viszont a megrendelő számára is érthető, hétköznapi nyelven legyen megfogalmazva.
Mindezen feltételeknek tesznek eleget a Use Case-ek, *(magyarra ferdítve: használati esetek)*, melyek célja a funkcionális követelmények szisztematikus leírása. Hogyan is néz ki mindez?
A megrendelő igényeinek felmérése közben nem elég, ha ő megmondja nekünk, milyen folyamatot kell megvalósítanunk, azt is tisztázni kell, hogy milyen lépésekből áll ez a folyamat. Felépítünk tehát egy forgatókönyvet, amely a folyamat sikeres lefutásának lépéseit tartalmazza. A fenti példánál maradva, a sikeres piskótasütés lépései az alábbiak lesznek *(cukrász barátunkkal történt egyeztetésünk után)*:

1. Hozzávalókat összekészít.
2. Tepsit kivajaz, sütőt begyújt.
3. 12 tojást felüt, sárgáját és fehérjét szétválaszt.
4. A sárgájába belekever 12 evőkanál cukrot, majd habosra ver.
5. A fehérjébe tesz 1 csipet sót és kevéske porcukrot, majd habosra ver.
6. A habos fehérjét óvatosan a sárgájába kever.
7. Hozzákever lassan, egyenletesen 12 evőkanál lisztet.
8. A tepsibe önt, a tepsit a sütőbe tolja.
9. Aranysárgára süt.

Ezeket a lépéseket kell tehát megvalósítania a rendszernek, ez az a forgatókönyv, amelynek minden lépése sikeres, így természetesen a végső kimenetel is. Ezt szemlélteti az alábbi ábra, amelyet a későbbiekben tovább építgetünk *(a számok a fenti, sorszámozott lépéseket jelölik)*:



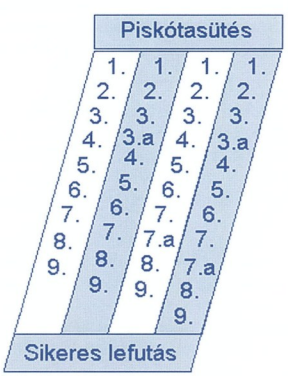
A sikeres lefutás menete

A sikeres lefutást tehát a megrendelővel karöltve leírtuk, így ezt már tudjuk, hogyan kell megvalósítani. Ekkor viszont egészen biztosan előjönnek azok a lehetőségek, amikor egy-egy lépés sikertelen ugyan, valamilyen hiba lép fel, de ez a probléma még orvosolható, így végül a folyamat egészét tekintve sikeres lefutást kapunk.
Ilyen eset lehet például, ha a robot elejt egy tojást, amely összetörik, de megoldást jelenthet egy újabb tojás elővétele. Ezeket a lépésenkénti alesetet úgy jelöljük, hogy az eredeti lépés sorszáma mögé elkezdjük felsorakoztatni az ABC betűit. Például:

3. 12 tojást felül, sárgáját és fehérjét szétválaszt.
3a. Tojás eltörik, újat vesz elő.
3b. Tojás megromlott, újat vesz elő.

7. Hozzákever lassan, egyenletesen 12 evőkanál lisztet.
7a. Liszt kiborul, új csomagot vesz elő.

Ennek megfelelően a fenti ábra kiegészíthető a hibás eseteket lekezelő, úgynevezett kiegészítő forgatókönyvekkel:



A piskótasütés folyamata a kiegészítő forgatókönyvekkel

Jól látható, hogy az első oszlop továbbra is a főforgatókönyvet tartalmazza, míg a többi a tojástörést illetve lisztkiömlést, a legutolsó forgatókönyv szerint dolgozó robotot pedig valószínűleg rövid idő alatt leselejtezi a tulajdonosa, hiszen minden lehetséges hibát elkövet.

A fenti Use Case-ek tehát a sikeres lefutású forgatókönyveket írják le, lépésről lépésre, pontosan definiálva minden egyes műveletet. S mindezt úgy, hogy a cukrász megrendelő is könnyedén megértse. Természetesen ezek a Use Case-ek folytonos egyeztetések során jönnek létre, az ügyféllel közösen alakítjuk ki őket. Annak köszönhetően, hogy lépésről lépésre mindent végiggondolunk, benne is tisztázódik, hogy pontosan mit és hogyan szeretne, a ködös elképzelésekből kirajzolódik a majdani rendszer váza.
(Egy ismerősöm szerint olyan ez, mint egy ember csontváza. Ha viszont már az elején rosszul rögzítjük a követelményeket, soha nem lesz a vázból énekes halott...)

A valóságban azonban nemcsak sikeres lefutás létezik, hanem esetenként meghiúsul az egész műveletsor. Ezt is tudni kell modellezni.
Természetesen a Use Case-ek erre is használhatók. A sikeres lefutáshoz hasonlóan leírhatjuk azokat az eseteket is, amikor egy-egy lépés meghiúsulása a teljes folyamat meghiúsulásához vezet.

Lássunk erre is néhány példát:

1. Hozzávalókat összekészít.
1x. Nincs meg minden ➔ Meghiúsulás.

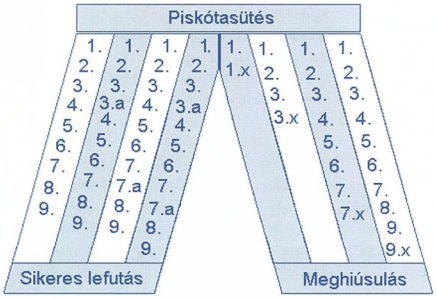
3. 12 tojást felül, sárgáját és fehérjét szétválaszt.
3x. Tojás eltörik, nincs otthon több ➔ Meghiúsulás.
3y. Tojás megromlott, nincs otthon több ➔ Meghiúsulás.

7. Hozzákever lassan, egyenletesen 12 evőkanál lisztet.
7x. Liszt kiborul, nincs otthon több ➔ Meghiúsulás.

9. Aranysárgára süt.
9x. Piskóta megég ➔ Meghiúsul.

Napestig sorolhatnánk a hibalehetőségeket, amelyek meghiúsítják a folyamatot. Ezek mélysége természetesen a felhasználó igényeitől függ elsősorban, azonban azt mindenképp figyelembe kell venni a fejlesztés során, hogy a konkrét hibák lekezelésén túl az előre nem látott eshetőségekre is fel kell készíteni a rendszert. Például ha csak a fenti hibaeseteket vesszük figyelembe, a robot akkor se omoljon össze, ha esetleg a 8. lépésben kiborítja a tepsi egész tartalmát.

Ennek megfelelően az eddig építgetett ábránk végleges, a hibalehetőségeket is tartalmazó formája:



A piskótasütés lehetséges forgatókönyvei

Ha Lagzi Lajcsi tudta volna, hogy közismert nótájában a Use Case-ek rejtelmeibe vezet be bennünket!... ☺ *(Megkérdeztem a szerzőt, mégis mire gondol: „Van nekem egy csíkos gatyám...", lásd a fenti ábrát. – a szerk.)*

További lépések

A Use Case-ek segítségével leírhatjuk tehát, hogy mit szeretne a megrendelő, de természetesen ez nem elég ahhoz, hogy az elképzelésből működő rendszer legyen. A formailag ugyan kötött, de köznyelven megfogalmazott Use Case-ek kiindulási pontot adnak a további tervezéshez.

Igen ám, merül fel ilyenkor a jogos kérdés, de hogyan menjünk tovább, mi legyen a következő lépés, ha már tudjuk, mit kell megvalósítanunk? Honnan fogjuk tudni, hogyan valósítsuk azt meg?

A rendszer megtervezéséhez először is nagyfokú rálátásra van szükségünk: ha már ismerjük az ügyfél igényeit, el kell tudnunk dönteni, melyik technológia a legalkalmasabb annak megvalósítására. Természetesen ennek eldöntéséhez nem csupán szakmai szempontok figyelembevételére van szükség, hanem egyéb dolgokat is mérlegelni kell.

Először is a vevő anyagi és egyéb erőforrásaira kell tekintettel lennünk: ha nem engedheti meg magának, hogy milliós nagyságrendben ruházzon be szerverekre, szoftverekre és egyéb komponensekre, valószínűleg nem célszerű ebbe az irányba indulni. Figyelembe kell venni, milyen hardverek és szoftverek állnak rendelkezésére: ha a vállalati struktúra alapvetően Windows szerverekre és kliensgépekre épít, nem célszerű Linuxos rendszer kifejlesztésén gondolkozni, legalábbis erősen mérlegelni kell, és erős érveket kell felsorakoztatni ahhoz, hogy mégis így döntsünk.

Tekintettel kell lennünk a rendelkezésre álló időre is: munkám során többször találkoztam már azzal a hozzáállással, hogy nem a feladathoz kellett igazítani a határidőt, hanem a határidőhöz a feladatot. Ez persze nem egészséges megoldás, de a gyakorlatban mégis létezik: a megrendelő azt tudja, hogy például 6 hónapon belül szeretne egy ilyen meg ilyen rendszert. A pontos funkciók még nem tisztázódtak benne, de abban egészen biztos, hogy 6 hónap múlva már használni szeretné. Ilyenkor jön az a fajta egyeztetés, amikor a követelmények rögzítése, a Use Case-ek felállítása közben már arra is figyelnünk kell, hogy vajon beleférünk-e a megadott időkorlátba.

Lássunk egy példát: A felhasználó egy piskótasütő robotot szeretne. A megbeszélések során felmerül, hogy jó lenne, ha nemcsak ki tudná sütni a piskótát, hanem különféle krémeket is keverne rá. Ez azonban újabb problémákat vet fel: egyrészt újabb funkciókat jelent, másrészt újabb erőforrásokat is, hiszen pl. tej nem kell a tésztához, de a krémhez egészen biztosan. Ha azt látjuk, hogy ennek megvalósítása legalább 3 hónapot venne igénybe, és ezzel már 8 hónapra nőne a szükséges idő, akkor nyilván nem szabad vállalni ennek a funkciónak a megvalósítását. *(Vagy későbbi fázisra halasztani.)*

Néhány szó az időről

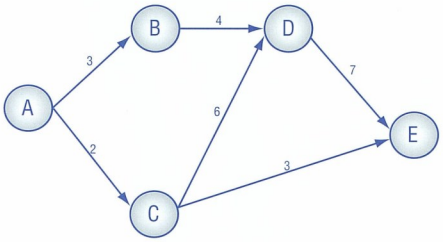
Az idő mindig kritikus kérdés, ennek megfelelően nagyon körültekintően kell bánnunk vele. Ahhoz, hogy a teljes folyamat időigényét becsülni tudjuk, először is részfolyamatokra kell bontanunk azt. Egy szoftverprojekt alapvetően négy fázisra bontható:

- Előkészítés
- Tervezés
- Megvalósítás *(fejlesztés, tesztelés)*
- Lezárás *(átadás, működtetés)*

Az előkészítés során a legintenzívebb a kommunikáció a megrendelővel, hiszen ekkor rögzítjük a követelményeket, megszületik a funkcionális specifikáció. A tervezés már elsősorban mérnöki, informatikusi feladat, a megrendelővel lazábbá válik a kapcsolat, ritkulnak a találkozások. Nagyon fontos azonban, hogy a kapcsolatot lehetőség szerint folyamatosan tartani kell, hiszen a vevő akkor tud konstruktívan részt venni a folyamatban, akkor tud segíteni, ha engedjük neki. A megvalósítás során „kódoljuk" le a megtervezett funkciókat, ez az a fázis, ahol a klasszikus értelemben fejlesztünk. Az utolsó szakaszban történik a rendszer átadása, beüzemelése, a felhasználók betanítása, stb.

Jól látható módon ezzel a felbontással máris elindultunk a részfolyamatra bontás útján. Az egyes fázisokat tovább már csak a szakértők bevonásával oszthatjuk, és a szükséges idő megbecsüléséhez is fontos az ő bevonásuk.

A folyamatokat eleinek tekinthető teendőkre bontjuk tehát, majd ezekhez a teendőkhöz időtartamokat rendelünk. Ha feltételezzük, hogy ezek a teendők nem párhuzamosíthatók, a teljes időigény természetesen ezek összege. A valóság azonban általában az, hogy bizonyos teendők párhuzamosíthatók, míg mások egymásra épülnek. Ezeket az összefüggéseket egy gráfon ábrázolhatjuk. Egy példát mutat az alábbi ábra:

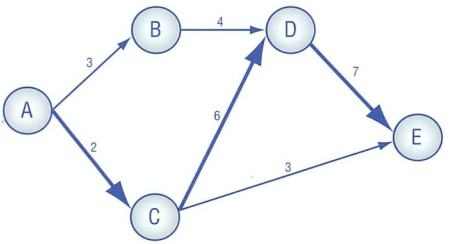


Tevékenységek grafikus ábrázolása

A fenti ábrán a nagybetűvel jelölt körök egy-egy mérföldkövet jelentenek, vagyis elérendő állapotot, megvalósítandó célt, a közöttük futó irányított nyilak pedig a tevékenységeket, rajtuk az elvégzésükhöz szükséges időtartam, például hetekben kifejezve *(a mértékegység tetszőleges, azonban használatának*

következetesnek kell lennie). A folyamat tehát az A állapotban kezdődik. Ahhoz, hogy innen eljussunk a B-be, 3, a C-be 2 hétre van szükség. A nyilak párhuzamossága a tevékenységek párhuzamosítására utal, ez a két teendő tehát végezhető egy időben. A B állapot elérése előfeltétele annak, hogy továbblépjünk D-be, a C pedig mind a D-nek, mind az E-nek. A befejező E állapotnak azonban a D is feltétele.

Lássuk, mennyi idő szükséges a folyamat teljes befejezéséhez: B-be 3 hét, C-be 2 hét alatt jutunk el. D-hez egyrészt be kell fejezni a B-t követő 4 hetes munkát, másrészt a C-t követő 6 hetest, így a két kritérium: a B-n áthaladó 3+4=7 hetes, a C-n áthaladó 2+6=8 hetes feladatsor befejezése. Mivel ez a két feltétel ÉS kapcsolatban áll egymással, a D állapot leghamarabb max*(7, 8)* = 8 hét alatt érhető el. Hasonlóan kapjuk, hogy az E-be leghamarabb max*(8+1, 2+3)* = 9 hét munka árán juthatunk el. Lássuk, melyik az a tevékenységsorozat, amely igényli ezt a 9 hetet:



A kritikus út

A fenti ábrán vastag vonallal megjelöltem azon tevékenységeket, amelyek kitöltik a 9 hetet. Ezek közül bármelyik csúszik is, az az egész folyamat késését jelenti. Például ha a C→D tevékenység 6 helyett 7 hét alatt készül el, az azt jelenti, hogy a teljes folyamat minimális időigénye 10 hétre nő. Az ilyen tulajdonságú tevékenységek sorozatát kritikus útnak nevezzük *(a fenti ábrán ezt húztam ki vastag vonallal).*

Azon tevékenységek késése, amelyek nincsenek a kritikus úton, nem feltétlenül okozza az egész folyamat késését. A C→E tevékenység például mindaddig nem okoz fennakadásokat, amíg az A→C→E út hossza a kritikus út hosszánál kisebb marad.

Ez a tervezési módszer többféleképp használható. Egyik megközelítés az, amikor az elvégzendő tevékenységek ismertek, és azt szeretnénk megtudni, mennyi idő alatt végezhetünk velük. Ebben az esetben felrajzoljuk a folyamat fenti diagramját, és a kritikus út megkeresésével becsülhetjük a szükséges időtartamot.

A másik megközelítés, amelyet már említettem, szintén sokszor előfordul a gyakorlatban: az időkorlát ismert *(vagyis a kritikus út hossza),* és az ebbe beleférő tevékenységek lehető legbővebb halmazát keressük. Ennek módja a következő: első megközelítésben felveszünk minden tevékenységet, amelyet szeretnénk megvalósítani, az időkorlát figyelembe vétele nélkül. A diagramon megkeressük a kritikus utat, ez alapján megkapjuk, mennyi lenne a szükséges idő ebben az esetben. Ha ez az idő nem nagyobb a rendelkezésre álló hetek számánál, készen is vagyunk, hiszen beleférünk a keretbe.

A probléma akkor kezdődik, ha le kell faragni az időből, hiszen túllépnénk a korlátokat. Ekkor egyrészt elkezdhetünk alkudozni a megrendelővel, viszont ő többnyire hajthatatlan. Nem marad más, mint a tevékenységekből lefaragni. Mivel a kritikus út az, ami alsó határt szab a szükséges időtartamnak, e mentén kell „visszafogni" magunkat, azaz valamely tevékenységet vagy teljes egészében csökkenteni, vagy másik, kevesebb funkcionalitást megvalósító, de kevesebb időt is igénylő feladattal helyettesíteni. Ekkor a módosított gráfon újra meg kell keresni a kritikus utat, majd ezt vizsgálva *(belefér-e az időkorlátba)* a fenti műveleteket mindaddig folytatni, amíg megfelelő eredményt nem kapunk. A tevékenységek által ekkor megvalósított funkcióhalmaz lesz az, amely belefér a meghatározott időkeretbe.

Természetesen vannak még más módjai is annak, hogy időt takarítsunk meg. Egyrészt elképzelhető, hogy újabb emberek bevonásával gyorsabban halad a fejlesztés. Ezzel azonban óvatosan kell bánni, hiszen ha az új emberek képzettsége nem a megfelelő, többet veszítünk a képzésükkel és felzárkóztatásukkal, mint amennyit nyerünk az általuk elvégzett munkával. A másik szempont, amelyet újabb erőforrások esetén figyelembe kell venni, hogy több erőforrás több pénzbe is kerül. Meg kell tehát találni az idő, a pénz és az elvégzett munka optimumát, azonban arra is figyelni kell, hogy a feladatok nem párhuzamosíthatók a végtelenségig. *(Ha egy nő kilenc hónap „munka" után szül meg egy kisbabát, kilenc nő nem képes ugyanerre egyetlen hónap alatt...)*

Az is lehetséges, hogy az emberek átcsoportosításával, a feladatok más leosztásával optimálisabb időbeosztáshoz jutunk. Nagyon triviálisnak tűnő dolgok ezek, de nagyon sokszor látni, hogy vezető beosztású emberek nem veszik figyelembe dolgozóik személyes adottságait: a jó adatbázisfejlesztőt beosztják kliensoldali alkalmazás fejlesztésére, a webdesignert tárolt eljárások írására stb.

Egyik nagy tiszteletnek örvendő tanárom, Prónay Gábor hasonlata rendkívül találó: tegyük fel, hogy van lehetőségünk összeszedni a világ legjobb focistáit, és egy csapatba összeállítani őket: a legjobb csatárt, a legjobb kapust, a legjobb hátvédet stb. Azt feltételezzük, hogy ekkor a világ legjobb focicsapatát kapjuk. Ha viszont a játékosokat mind egy kicsit más helyre állítjuk a pályán, mint amiben a legjobbak *(a csatárt kapusnak, a kapust hátvédnek stb.),* az eredmény siralmassá válik.

Az időgazdálkodáshoz fontos szempont még a megfelelő technológia kiválasztása is. Ezt nem csupán a megrendelő pénztárcájához, hanem saját erőforrásainkhoz is igazítani kell: ha például cégünk mindeddig Java és PHP programok fejlesztésével foglalkozott, és most egy .NET alkalmazást várnak tőlünk, kellő időt kell hagyni a képzésre, tanulásra is, hiszen az „in medias res" beleugrás a fejlesztésbe nem szokott sikerre vezetni.

A fejlesztési folyamat tervezésekor figyelembe kell vennünk a rendszer más rendszerekkel való kapcsolatát is, hiszen egyre több az integrált szoftvert, új programunk nem önállóan fog működni, hanem a vállalat egyéb folyamatival összhangban. Nem mindegy tehát, mi van körülöttünk. Ehhez ismét a vevő nagyfokú közreműködése szükséges.

Lássunk itt is egy példát. Tegyük fel, hogy piskótasütő robo-

tunkat olyan környezetben kívánjuk elhelyezni, ahol más robotok is dolgoznak. Az egyszerűség kedvéért legyen ez egyetlen szakácsrobot. Ha a két robotot két különböző cég fejleszti, és nem tudnak egymásról, nyilvánvalóan nem is merül fel annak gondolata, hogy munkájukat például össze is kell szinkronizálni. Érdekes eredményeket produkálhatunk például, ha a piskótatésztába beletesszük a sütnivaló csirkét, vagy egyik robot 180, a másik pedig 300 fokra állítja a sütő hőmérsékletét. A hétköznapi életre átfogalmazva tehát ügyelni kell a közösen használt erőforrások *(adatbázisok, nyomtatók, stb.)* megfelelő szinkronizációjára.

Másik probléma lehet az integrációnál, ha a két rendszernek együtt kell működnie, például egymástól kapnak adatokat, esetleg utasításokat is. Ilyenkor a rendszerek közötti kommunikáció biztosítására megfelelő interfészt kell biztosítanunk egymás irányába. Ha például a piskóta tetejére csokimázt szeretnénk önteni, és ezt egy másik robot készíti, annak mindenképp tudnia kell az aktuális piskótalap méretét, vastagságát stb., hogy ahhoz tudja igazítani a keverendő máz mennyiségét.

A való életben itt is előjöhetnek egyéb, nem szakmai jellegű problémák is. Előfordulhat, hogy egy üzleti döntés értelmében két vagy több rendszert egyidőben szeretnének bevezetni. Ilyenkor egy-egy határidő módosítása több másik határidő módosítását is maga után vonhatja, hiszen ha a rendszerek közül akár csak egy is késik, az felborítja az egész átadási folyamatot.

Ezért rendkívül fontos, hogy amennyiben más rendszerektől is függünk, pontosan tisztában legyünk minden olyan tényezővel, amely akár a legkisebb mértékben is befolyásolja projektünk sikerességét.

Ha mégis hiba csúszik be...

Mi történik akkor, ha minden elővigyázatosságunk ellenére csúsznak az egyes fázisok, és késik a végtermék?

Talán ezt a legnehezebb megmondani. Először is amint prognosztizáljuk a késés lehetőségét, el kell ismernünk azt, és nem a csodára várnunk, bízva abban, hogy valami történik, aminek következtében mégis készen leszünk határidőre.

Késés esetén alapvetően két lehetőség közül választhatunk. Egyrészt igyekezhetünk megvalósítani mégis minden funkciót, kevésbé kitesztelve, kevésbé odafigyelve a minőségre, hiszen még így is van esélye annak, hogy helyesen fog működni.

A másik taktika, hogy tovább szűrjük a funkciókat *(a megrendelővel közösen)*, és csak annyit valósítunk meg, amennyit biztonsággal be tudunk vállalni, azaz amelyek esetében biztosan nem következik be minőségromlás. Gondolom, nem kell ecsetelnem, miért célszerű ez utóbbi lehetőséget választani...

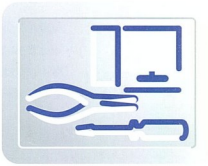
A harmadik lehetőség a határidő közös, a vevővel egyeztetett módosítása. Előfordulhat, hogy bár a projekt elején ragaszkodott a 6 hónaphoz, azóta mégis történtek olyan változások, amelyek alapján a 7 hónap is elfogadható számára, vagy esetleg a rendszer fejlesztése közben rálátott arra, hogy milyen alaposan, színvonalasan dolgozunk, és látja, nem jár rosszul, ha rááldozza még azt a plusz időtartamot.

Ezek azonban már nem szorosan informatikai, sokkal inkább menedzsmentkérdések. Ahhoz azonban, hogy egy fejlesztési folyamatot sikeresen le tudjunk vezényelni, nem elég a fejlesztéshez érteni, a feladat sokkal összetettebb. Egy kicsit fejlesztőnek, kicsit menedzsernek, kicsit pszichológusnak és szociológusnak is lennünk kell egyszerre, ez azonban nem könnyű feladat.

Az élet viszont azt igazolja, hogy nem is lehetetlen... ☺

Molnár Ágnes
Molnar.Agnes@cleware.hu

Kistestvére születik az ADO.NET-nek



XMLA és ADO MD .NET előzetes

Az MS SQL Server 2000 egyik újdonsága volt, hogy támogatta a relációs adatbázisaiban tárolt adatok elérését XML segítségével. Az ADO.NET-tel a relációs adatok és az XML fogalma még jobban összefonódott. De mi van az Analysis Serverben tárolt régi jó adatkockáinkkal és adatbányász modelljeinkkel? El tudjuk-e érni őket a .NET, vagy XML segítségével? Ezt fogjuk most megvizsgálni.

Mire nem jó az ADO.NET?

Számtalan hasznos és érdekes cikk jelent már meg e magazin hasábjain az SQL Server 2000-rel kapcsolatban. Azonban ezek a cikkek szinte kivétel nélkül a relációs adatbázismotorhoz kapcsolódó témákkal foglalkoztak. Ahogy a világban sem minden szögletes *(hajrá kerek formák)*, az SQL Server 2000 adatbázisai közül sem minden relációs. Ez a másik világ a többdimenziós adatbázisok csodálatos világa.

Hol is rejtőznek ezek az adatbázisok? A választ az SQL Server konferenciákra járó kedves olvasó is bizonyára jól tudja, hogy a többdimenziós adatbázisainkat az Analysis Server rejti. Elevenítsük csak fel, hogy mi is van ezekben az adatbázisokban? A konferencialátogatók már tudják, hogy csupa fontos információt tartalmazó válaszok:

- Mely termékek hozzák a legtöbb profitot az egyes értékesítési régiókban?
- Milyen szezonalitás figyelhető meg az egyes termékek esetében?
- Ha továbbra is ilyen ütemben bővül a piac, akkor várhatóan jövőre kinek, miből mennyit fogunk értékesíteni?
- Mi jellemzi a tipikus fogyasztónkat, milyenek a vásárlási szokásai?
- A régi ügyfeleink közül a piaci verseny élesedésével kiket tud elcsábítani a konkurencia?

Ugye fontosak ezek az információk? De hogyan érhetjük el őket? Simán, vághatnánk rá kapásból. Van nekünk egy gyönyörűséges ADO.NET-ünk, amivel olyan egyszerűen érjük el az SQL Serverben levő adatbázisainkat, hogy öröm nézni, és tuti, hogy vannak benne olyan osztályok is, amelyekkel gyerekjáték a többdimenziós adatok elérése. Ki tudja megmutatni, hogy melyek is ezek az osztályok? *(Segítségként elárulom, hogy az ADO.NET osztályokat az XML-es osztályokkal együtt a legkisebb poszteren találhatjuk.)*

A kérdés bizony beugratós volt. A többdimenziós adatbázisszerver már a 7-es verziójú SQL Serverben 1998-ban megjelent *(akkor még OLAP Servernek hívták, a 2000-es továbbfejlesztett verzióban Analysis Serverre keresztelték át a fejlesztői, köszönhetően az adatbányász kiegészítéseknek)*, azonban ADO.NET-tel csak a relációs adatbázisainkat tudjuk elérni. *(Ha valaki mégis rálelne ezekre az osztályokra, kérem szóljon, mert akkor én egy nyomdahibás posztert kaptam. ☺)*

Akkor most mit tegyünk? Töröljük le az Analysis Servert és mindent, ami rá emlékeztet? Eszünkbe ne jusson!! Már látszik a fény az alagút végén, jön az ADO MD .NET.

ADO MD .NET, hát az meg mi a szösz?

Akik figyelmesen átolvasták az ADO *(ActiveX Data Objects)* dokumentációt, azoknak észre kellett venniük a dokumentációban szereplő ADO MD-t is. De mit is jelent a végén az MD és mire jó? Az MD a Multi Dimensional, azaz többdimenziós rövidítése és az Analysis Serverben levő adatbázisaink sémáját és a sémában tárolt adatainkat tudjuk vele lekérdezni. Csak lekérdezni, ugyanis a séma felépítésére, illetve módosítására a DSO *(Decision Support Objects)* használható. Az ADO MD dokumentációja az **[1]** címen, a DSO használatát bemutató cikkek pedig a **[2]** és **[3]** címen érhetőek el.

Ahogy a világban sem minden szögletes (hajrá kerek formák), az SQL Server 2000 adatbázisai közül sem minden relációs. Ez a másik világ a többdimenziós adatbázisok csodálatos világa.

Mind az ADO MD, mind a DSO a .NET előtti COM komponensek világából származik, menedzselt kódból eddig csak COM interop segítségével használhattuk őket.

Néhány hete, 2003 augusztus elején jelent meg az Interneten az ADO MD .NET bétaváltozata, jelezve, hogy az ADO MD sem marad ki a .NET-esítésből. A **[4]** címről a szoftverfejlesztői készletet *(SDK)*, az **[5]** címről pedig a dokumentációt lehet letölteni. A béta változatot jelzi, hogy sajnos még önállóan nem tud megállni a lábán, előtte fel kell telepíteni az XML for Analysis *(XMLA)* 1.1-es *(szintén béta)* verzióját, ami a **[6]** címen érhető el.

XMLA röviden

Az XML for Analysis szabványtervezet a Microsoft és az Essbase, illetve az időközben hozzájuk csatlakozott cégek elképzelése arról, hogy hogyan lehetne egységes módon elérni a többdimenziós adatbázisokat. A kezdeményezésről és a résztvevőkről további információk a **[7]** címen olvashatók.

Az első verzió 2001 áprilisában jelent meg, segítségével SOAP borítékokon *(Simple Object Access Protocoll, XML for-*

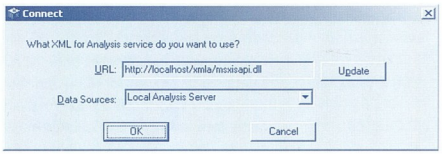
mátumú kommunikáció szabványos HTTP protokollon) keresztül tudtuk elérni a kiszolgálóra telepített XMLA szolgáltató *(provider)* metódusait, ha ismerjük a szolgáltató URL-jét.

XMLA bővebben

Nézzük is meg kicsit részletesebben, hogy mi is került az 1.1-es telepítőkészletbe. Alapértelmezés szerint csak a biztonságos HTTPS protokollon keresztül érjük el az adatainkat. A HTTP protokollt a telepítés során külön engedélyeznünk kell! A telepítés során, ha nem változtatunk rajta, a következő helyre települnek a komponensek:

```
C:\Program Files\Microsoft XML For Analysis SDK
```

A működéshez azonban még néhány lépést el kell végezni. Először is olvassuk el a readme.htm-et, ahonnan megtudhatjuk, hogyan állíthatjuk be az adatforrásunkat és az msxisapi.dll-t tartalmazó virtuális könyvtárunkat, aminek a nevére nem adnak ajánlást, de a példaprogramok xmla néven hivatkoznak rá. Dönthetünk persze más név mellett is, ebben az esetben viszont át kell írni a hivatkozásokat.



Az msxisapi.dll-t az xmla virtuális könyvtárba tegyük, vagy írjuk át az elérési utat

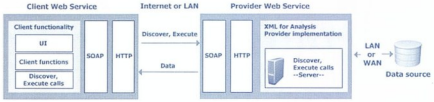
A Help könyvtárban találhatunk egy elég jó súgót, amiből az is megtudhatjuk, hogy a Samples könyvtárban három egész példaprogram azt várja, hogy kipróbáljuk:

- Sample: A sokat próbált MDX Sample nevű ADO MD-s példaalkalmazás átírása XMLA-ra.
- Sample.NET: Ez is az MDX Sample alkalmazás VB .NET-es átírása XMLA-val.
- Simple: Egyszerű, de nagyszerű példaprogram az adatbázis sémájának és a benne tárolt adatok lekérdezésére. A megjelenítés lehet XML és HTML alapú is.

A .NET-es példa elindításához szükségünk van VB 6-os komponensek regisztrálásához is. Ebben az esetben a 318579-es számú Tudásbázis cikk írja le a megoldást. Már csak egy apró problémánk lehet, megtalálni a hiányzó Mdxquery.xml nevű fájlt a mintalekérdezésekkel. Ezt a fájlt a Sample nevű példaprogram könyvtárában találjuk.

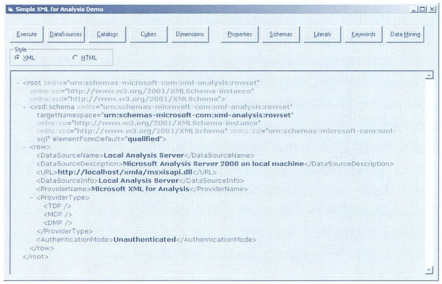
De mitől mozog?

A példák kipróbálása után nézzük is meg, mi van a mélyben, mitől is működnek. A fő mozgatórugó az xmla virtuális könyvtárban lévő XMLA szolgáltató. A szolgáltatót HTTP protokollon keresztül elküldött SOAP borítékba csomagolt XML alapú kérésekkel szólíthatjuk meg. A megszólítás vonatkozhat az adatbázis sémájára, ekkor a Discover metóduson keresztül, illetve többdimenziós lekérdezések végrehajtására, ekkor pedig az Execute metóduson keresztül kaphatjuk meg az igényelt információt, természetesen SOAP borítékba csomagolva.



Adatbázisok elérése XMLA segítségével

Miután a Discover metódussal felderítettük az elérhető adatbázisokat, adatkockákat, dimenziókat, adatbányász modelleket, az Execute metódussal le tudjuk futtatni az MDX-ben *(MultiDimensional eXpression)* megfogalmazott lekérdezéseket. Az MDX a többdimenziós adatbázisok lekérdező nyelve. Szintaktikáját tekintve annyiban hasonlít a relációs adatbázisok lekérdezésére használt SELECT utasításra, hogy itt is van FROM és WHERE záradék, azonban el is tér tőle, mert többdimenziós és adatbányászati adatok lekérdezésére alkalmas. Az MDX-ről rövid ismertető a **[8]** címen, az SQL és az MDX rövid összehasonlítása a **[9]** címen érhető el.



Egyszerű példák a sémák és az adatok lekérdezésére

Az XMLA egy remek megoldás de azért csak kényelmesebb lenne egy objektummodellen keresztül elérni az adatokat.

ADO MD .NET

A hiányzó objektummodellt az ADO MD .NET fogja szolgáltatni, egyelőre úgy, hogy a háttérben XMLA-n keresztül kommunikál a kiszolgálóval. Az objektummodellt a **Microsoft.AnalysisServices.AdomdClient** névtérben találhatjuk meg. A névtér legfontosabb objektumai:

- ADOMDConnection: Segítségével a kiszolgálón található adatbázisokhoz, vagy a helyi .cub kiterjesztésű kockafájlokhoz kapcsolódhatunk. A kockafájlokba az adatbázisunk egy részét tudjuk exportálni.
- ADOMDCommand: MDX lekérdezések lefuttatására alkalmas, ADOMDDataReader vagy Cellset objektummal tér vissza.
- ADOMDDataReader: MDX lekérdezés eredményét adja vissza táblázatos formában; csak előrefele olvashatjuk.
- CellSet: Szintén MDX lekérdezés eredményét tartalmazza, de lassabb, mint az ADOMDDataReader, viszont előre és hátra is navigálhatunk az adatokon és a metaadatokon.
- CubeDef: Az elérhető adatkockák felépítéséről kaphatunk információkat

Sajnos a fejlesztők elfelejtettek példaprogramot mellékelni a telepítőcsomaghoz, meghagyva a felfedezés örömét az elszántságot érző vállalkozóknak. Remélhetőleg a végleges változat kibocsátásakor nem lesznek ilyen feledékenyek.

Homokszemcse a gépezetben

Az XMLA SDK-hoz példaként kapott programok nem tökéletesek. Mindkét Sample nevű példaprogram ugyanazt az apró hibát tartalmazza. Ha a Foodmart 2000 adatbázisban a következő lekérdezést lefuttatjuk, eredményül nullával való osztási hibát kapunk.

```
SELECT FROM SALES
```

A hiba a megjelenítési részben van: az frmXmlAnalysis 569-es sorából hiányzik egy 0-val való ellenőrzés. A hibát okozó sor és az utána következő négy sor az Else ágba került.

```
If cTuples0 = 0 Then
   ctlFlexGrid.TextMatrix(ctlFlexGrid.FixedRows,
↳ctlFlexGrid.FixedCols) =
↳xmlElement.selectSingleNode("FmtValue").Text
Else
   iRow = iCellOrdinal \ cTuples0
   iCol = iCellOrdinal Mod cTuples0
   If Not xmlElement.selectSingleNode("FmtValue")
↳Is Nothing Then
ctlFlexGrid.TextMatrix(ctlFlexGrid.FixedRows +
↳iRow, ctlFlexGrid.FixedCols + iCol) =
↳xmlElement.selectSingleNode("FmtValue").Text
   End If
End If
```

A Sample.NET példa esetében az frmXmlAnalysis.vb 1114-es sorához szintén kívánkozik az előbbi ellenőrzés, de itt az If ágban a következő kód szerepeljen:

```
ctlFlexGrid.set_TextMatrix(ctlFlexGrid.FixedRows,
↳ctlFlexGrid.FixedCols,
↳xmlElement.SelectSingleNode("xamd:FmtValue",
↳g_nsMgr).InnerText)
```

Szép új világ

Nem kell már objektummodell nélkül álomra hajtani a fejüket azoknak, akik az Analysis Serverben tárolt adataikból szeretnének jelentéseket készíteni a fontos üzleti döntéseik meghozatalához. A.NET-hez felnőtt ADO MD-vel most már nemcsak a relációs adatokat, hanem a relációstól eltérő módon tárolt többdimenziós és adatbányász adatainkat is könnyen és egyszerűen nyerhetjük ki az SQL Server adatbázisaiból.
Remélhetőleg az ADO MD .NET a .NET keretrendszer 2.0-ás – az SQL Serverrel szorosan integrált *(bővebben a [12] címen)* – következő verziójában már mindenki régi ismerősként fog visszaköszönni.

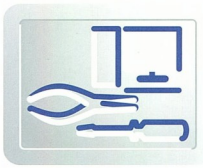
Hasznos holmik

Aki nem ismerné a cikkben szereplő .NET-es technológiákat, azoknak ajánlanám a Technet Magazin korábbi számait és a **[10]** címen található oldalt, ahol regisztráció után ingyenes, magyar nyelvű oktatóanyagokat igényelhet.
Az Analysis Serverrel most ismerkedőknek ajánlom az Analysis Manager elindítása után a jobb oldali ablakban a kis zöld könyvecske mellett megjelenő Analysis Manager Concepts & Tutorial-ban levő gyakorlatokat, a Books Online-t és az SQL Server 2000 Resource Kit-et **[11]**.

Hangyál Zoltán
hangyal@novosys.hu

A cikkben szereplő URL-ek:

[1] http://msdn.microsoft.com/library/en-us/ado270/htm/
↳adomdo01.asp
[2] http://msdn.microsoft.com/library/en-us/olapdmpr/
↳prabout_84a4.asp
[3] http://msdn.microsoft.com/library/en-us/dnsql2k/html/
↳dsosql.asp
[4] http://www.microsoft.com/downloads/details.aspx?
↳familyid=49737737-681e-465e-83c5-51b7223b1585
[5] http://www.microsoft.com/downloads/details.aspx?
↳familyid=D4DD8A54-F43A-4D93-A099-f3FBF117BF1DB
[6] http://www.microsoft.com/downloads/details.aspx?
↳familyid=02DDCC80-604B-4343-9D42-4298E10A1A56
[7] http://www.xmla.org
[8] http://msdn.microsoft.com/library/en-us/dnolap/html/
↳intromdx.asp
[9] http://msdn.microsoft.com/library/en-us/olapdmad/
↳agmdxbasics_90qg.asp
[10] http://www.developer.hu
[11] http://www.microsoft.com/technet/prodtechnol/sql/
↳reskit/sql2000/sql2krk.asp
[12] http://www.microsoft.com/hun/net/Roadmap.asp



Reguláris kifejezések

Szövegfeldolgozás elmélet és gyakorlat regexekkel

A szövegekben keresés, egyes részek cseréje, kiemelése már az informatika kezdete óta foglalkoztatja a szakembereket. A regexek segítségével rendkívül kifejező módon tudunk részleteket megfogni egy hosszabb és nem szigorúan szabályos szövegből is. Ennek elméleti és gyakorlati kérdéseit boncolgatjuk a következő kilenc oldalban.

Regex történelem

Reguláris kifejezés, angolul Regular Expression. Eredetileg a neuronfiziológiában vezették be ezt a fogalmat az 1940-es években. Érdekes nem, hisz akkor hol voltak még a maihoz hasonló számítógépek? Aztán jópár évvel később Stephen Kleene kitalálta a Reguláris Halmazokat mint matematikai elméletet, és ehhez vezetett be egy jelölésmódot, amit Regular Expression-nek hívott. Akit érdekelnek a matematikai részletek *(engem nem)*, annak itt a referencia: Robert L. Constable, "The Role of Finite Automata in the Development of Modern Computing Theory," in The Kleene Symposium, eds. Barwise, Keisler, and Kunen *(North-Holland Publishing Company, 1980)*.

A regexek első gyakorlati felhasználása Ken Thompson nevéhez fűződik *(remélem nem kell bemutatnom, ha igen, akkor sürgősen do google!)*, aki „Regular Expression Search Algorithm" című cikkében vezeti be a reguláris kifejezések használatát szövegfeldolgozásra.

Ő írta meg a qed nevű szövegszerkesztőt, ami a Unixon jól ismert ed kifejlődéséhez vezetett. Ezekben a szövegszerkesztőkben már volt regex kiértékelő, így lehetővé vált bonyolultabb szövegrészek megtalálása és cseréje is.

Az ednek volt egy parancssori eszköze, ami szövegfájlok soraira egyező reguláris kifejezéseket nyomtatott ki. Ez volt a „Global Regular Expression Print", rövidítve grep. Bízom benne, hogy nem Unixon felnőtt programozók is hallottak erről a programról.

Ezek a kezdeti programok persze még sokkal egyszerűbb reguláris kifejezéseket ismertek, mint a mai értelmezők, de *(mint minden fejlődés ebben a szakmában)* a regexek is iteratív módon fejlődtek.

Ezután jött a jóval több metakaraktert használó egrep, ami már teljesen más elven működött, mint elődje. A grep Deterministic Finite Automat-ot *(DFA)* használt, míg az egrep Nondeterministic Finite Automat-ot *(NFA)*. Egyszerűen megfogalmazva a DFA gyors, de butább, az NFA egyes esetekben rettentő lassú, de sokkalta okosabb, és szabadságot ad a regexeken keresztül a motor közvetlenebb vezérlésére. A mai regex motorok zöme NFA.

Majd jött a sed, az awk, a lex, amelyek mind valamilyen szempontból továbbfejlesztették a regexek nyelvtanát. A sokféle nyelvjárás között a POSIX szabvány próbált rendet rakni. Legfőbb előnye, hogy bevezette a locale fogalmát, így a betű végre nem csak az angol ABC karaktereit jelentette. Mi magyarok ismerjük a szakmában az „angol diszkriminációt", így örülünk a Posix törekvéseinek.

Napjainkban a Perl az a környezet, amely a reguláris kifejezések használatában és újításokban a fő húzóerő. A példákban a .NET Framework regex osztályait használjuk, amelyeket a Perl 5 regexei alapján modelleztek, így nagyon magasszintű regex támogatást kapunk.

Bevezetés

A regex a reguláris kifejezés rövidítése. A cikkünkben tárgyalt szövegeket bogarászó regexek már szokszor nem regulárisak matematikai értelemben, ezért általában egyszerűen regexként hivatkozunk rájuk, megkülönböztetve őket a matematikai reguláris kifejezésektől.

Nos, mi is az a regex? Egy olyan leíró nyelv, amely segítségével szövegek különböző részeit ragadhatjuk meg, írhatjuk le. Gondoljunk a fájlrendszerre:

```
dir a.txt
```

Ez kilistázza az a.txt fájlt. Mi van, ha az összes szövegfájl kell?

```
dir *.txt
```

Bevezettünk egy metakaraktert, a *-ot *(csillagot)*, amit úgy definiáltunk, hogy egyezik bármilyen fájlnévre. Nos, a regexek hasonlóak, csak sokkal több metakarakter található bennük, így sokkal gazdagabban fogalmazhatjuk meg az illesztendő szöveget.

Kiinduló példánk a következő lesz. Szeretnénk egy szövegben megkeresni a dadogásokat dadogásokat. Gyakori szövegszerkesztési hiba az ismétlés, ezt kellene megkeresni egy tetszőleges szövegben. Azt gondolnánk, minek ide regex, sima sztringkezelő eszközökkel is megoldható a probléma.

Például feldarabolhatnánk a szöveget szavakra *(whitespaceek mentén)*, majd végigmenve a listán összehasonlítjuk az egymás után következő szavakat. Ha egyeznek, ismétlést találtunk.

Igen ám de lehet, hogy a szavakat markup tagok határolják, mint pl. egy html szövegben:

```
Ez is <i>dadogásnak</i> <u>Dadogásnak</u> számít.
```

Ebben az esetben is meg kell találni az ismétlődő szavakat. Természetesen ez és minden más probléma is megoldható alapvető sztringműveletekkel *(Find, Split, Replace)*, de sokszor egy regexes megoldás sokkal egyszerűbb lesz.
A problémát megoldó regex így néz ki:

```
→\b(\w+)(\s|<[^<>]+>)+(\1)\b←
```

A regex csak a nyilak közötti rész, de a nyilakat mindig kiírom mind a szövegekben mind a kódblokkban, hogy jelezzem ha regexről beszélünk. A cikk célja, hogy a végére mindenki számára magától értetődő legyen ez a regex.

Karakteregyezések

Egy karakter saját magával mutat egyezést, ha nem vezérlőkarakter. A példákban az egyezéseket mindig aláhúzással jelölöm. Ahol fontos, ott kiírom, hány egyezést találna a regex motor.

```
→e←
Mesterkurzus - 2 egyezés
```

A legtöbb regex motor átkapcsolható kis-negybetűre nem érzékeny módra, ilyenkor értelemszerűen alakulnak az egyezések. .NET-ben ez a RegexOption.IgnoreCase opcióval érhető el.

```
→e←
Embergyerek - 4 egyezés
```

Karakterhalmaz egyezések

Egy karakterhalmaz egyezést mutat ugyanazzal a karakterhalmazzal, szóhatártól függetlenül.

```
→ek←
Mekk Elek legyek - 3 egyezés
```

A regexekben minden karakter számít, még a whitespacek is.

```
→1492. ←
Született: 1492.  08. 12.  - 1 egyezés
```

de

```
→1492. ←
Született: 1492.08.12.  - 0 egyezés
```

Ez fontos, mert sokszor hajlamosak vagyunk egy bonyolultabb regexet picit szellősebbé tenni szóközökkel, ám ettől megváltozik a regex viselkedése.
Egyes regex implementációkban, mint a Perl vagy a .NET, lehetőség van whitespace-ek és kommentek használatára a regexekben. .NET-ben a RegexOptions.IgnorePatternWhitespace opció után a whitespace-ek nem számítanak a patternben, és # után még megjegyzéseket is lehet fűzni a sorokhoz. De akkor ebben az esetben hogyan írjuk le a whitespace-eket? Hasonlóan, mint a legtöbb stringet feldolgozó programnyelven: escape szekvenciákkal. A következő táblázatban megtekinthetjük a legfontosabb *(de nem az összes)* helyettesítő karaktert.

Karakter	Leírás
Közönséges karakterek	Mind, kivéve . $ ^ { [(|) * + ? \
\b	Visszatörlés *(backspace)* \u0008 ha [] karakterosztályban van, egyébként szóhatár *(ezekről bővebben kicsit később)*
\t	Tab \u0009.
\r	Kocsivissza \u000D.
\n	Újsor karakter \u000A.
\x20	Egy ASCII karakter hexa kóddal, pontosan két digiten leírva. Ez pl. egy szóköz.
\cC	ASCII vezérlőkarakter *(32-nél kisebb kódú karakter)*, ez pl. a CTRL-C
\u0170	Egy Unicode karakter, pontosan négy hexa számjeggyel leírva, ez egy nagy Ő betű
\	Ha nem escape-elt karakter előtt van, akkor egyszerűen elhagyásra kerül, így marad a mögötte levő karakter. Pl. \g egyszerűen egy \g betű.

A \uxxxx hexa szám a karakter ún. code pointja az unicode táblázatban, nevezzük egyszerűen karakterkódnak. Látható, hogy ezt a jelölést nyugodtan lehet használni regexekben, így biztos nem fürdünk be a sunyiban o betűvé átkonvertálódott ő betűkkel *(a szövegszerkesztők miatt)*. A kódokat legegyszerűbben a Windows Character Map-ban találhatjuk meg.
A visszafele perjel *(\)* beírásához duplázni kell *(\\)*.

Karakterosztályok

Eddig csak konkrét karakteregyezéseket vizsgáltunk meg. Az f betű az f-fel egyezik, kész. Természetesen lehet használni olyan konstrukciókat, amelyek több mint egyféle karakterrel egyeznek, ezek a karakterosztályok.
Karakterosztályokat →[...]← *(szögletes zárójelek)* között lehet definiálni. Az osztályban felsorolt bármelyik karakter szerepelhet az egyezésben, de nagyon fontos, hogy pontosan egyetlen karaktert helyettesít egy karakterosztály kifejezés, nem többet.

```
→[rsk]←
Mesterkurzus - 5 egyezés
```

Látható, hogy a →[rsk]← jelentése: egy karakter, ami r vagy s vagy k lehet.

```
→[0123456789]←
Született: 1492. 08. 12. - 8 egyezés
```

A →[0123456789]← bármilyen decimális számjegyre illeszkedik, ezért 8 ponton egyezik a második sor tesztszövegével. Karaktertartományokat is megadhatunk karakterosztályokban – *(mínusz)*-szal elválasztva, így nem kell felsorolni minden egyes karaktert.
Ez a példa egzaktul azonos az előzővel, csak rövidebb:

```
→[0-9]←
Született: 1492. 08. 12. - 8 egyezés
```

A következő regexet így kell értelmezni: bárhol a szövegben egy decimális számjegy, amit egy decimális számjegy követ. Lehet, hogy így túl analitikusan hangzik, de bonyolultabb regexeknél a túl intuitív, „belelátom én mit csinál egyszuszra" gondolkodásmód gyakran tévútra csal.

```
→[0-9][0-9]←
Született: 1492. 08. 12. - 4 egyezés!
```

Karakterosztályon kívül a - jel közönséges karakter:

```
→[0-9][0-9]-←
Született: 1492-08-12. - 2 egyezés!
```

Érdemes odafigyelni, hogy sok karakter másképp viselkedik karakterosztályon kívül és belül. De ez még nem minden! A - jel karakterosztály elején és végén közönséges karakter. Nézzük kontrasztba állítva. Ebben a példában a regex a a-tól z-ig terjedő karaktert tartomány jelöli ki:

```
→[a-z]←
c - d - 2 egyezés
```

Itt viszont az első mínusz közönséges karakter:

```
→[-a-z]←
c - d - 3 egyezés
```

Karakterosztályon belül az utolsó pozíción is közönséges karakter lenne, így az →[a-z-]← azonos a fenti regexel.
Egy karakterosztályban több tartomány és karakter is felsorolható vegyesen is. Például:

```
→[eza0-2]←
Született: 1975-01-10 - 8 egyezés
→0x[0-9abcdefABCDEF][0-9abcdefABCDEF]←
0x15, 0xaF, 0x5H, 0x1B, hexa számok - 3 egyezés
```

Gyakran egyszerűbb megfogalmazni úgy egy karakterhalmazt, hogy bármilyen karakter, kivéve ezt és ezt. Erre való a negált karakterosztály. A jelölésmódja egy ^ *(kalap)* a karakterosztályt jelölő [*(nyitó szögletes zárójel)* után közvetlenül. A következő példa jelentése: bármilyen karakter, kivéve a 0-9-ig terjedő tartományt, magyarul bármi, ami nem szám:

```
→[^0-9]←
Született: 1492. 08. 12. - 16 egyezés
```

Nem első pozíción már közönséges karakter a ^:

```
→[0-9^e]←
Szü^letett: 1492. 08.^ 12. - 12 egyezés
```

Előre definiált karakterosztályok

Vannak gyakori esetek, amelyeket nem fontos hosszan karakterosztályként definiálni, hanem vannak előre elkészített rövidítések rájuk.

Gyakran kell a →[0-9]← karakteroszályt használni, amit a →\d← helyettesíthet. A következő példa négy egymás utáni számjegyre ad egyezést. Figyelem! Nem négydigites számokat keres! Mint már tudjuk, az egyezések függetlenek a hétköznapi értelemben vett szóhatártól, így az első hosszú számban három egyezés is lesz, a három egymást követő négy számjegyből álló blokk:

```
→\d\d\d\d←
1258632455458: 1492. 08. 12.  - 4 egyezés
```

A \D a →[^0-9]← karakterosztállyal egyezik meg, azaz nem számjegy.

```
→\D\D\D\D←
Született volna: 1492. 08. 12.  - 4 egyezés
```

Az egyezések kifejtve:

```
0 => Szül
1 => etet
2 => t vo
3 => lna:
```

Sajnos az aláhúzásos jelölésmód időnként nem egyértelmű, ezért néha kifejtem a kimenetet. Amikor progamozzuk a regexeket, ebből nincs gond, mert a találatokat egy kollekcióban kapjuk vissza.
A →\w← bármilyen betűt, számot vagy aláhúzásjelet (_) jelent. Nem bármilyen karaktert, hanem bármilyen betűt, beleértve a gonosz ő-ű betűket is. Pontosabban: ez attól függ. A .NET-es implementáció alapban minden betűt beleért, de átkapcsolható ECMA módba is *(RegexOptions.ECMAScript)*, ahol a →\w← == →[a-zA-Z0-9_]←. Azaz ettől kezdve csak az angol ABC betűivel foglalkozik, így az ékezetes betűinket mind kihagyná.

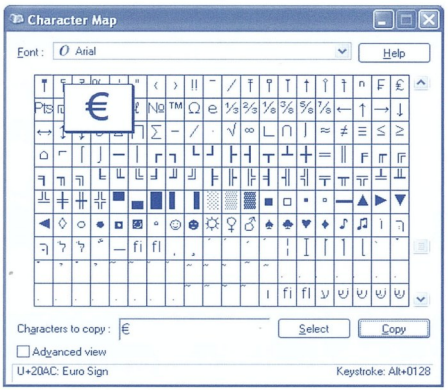
Figyelem! Más implementációban lehet, hogy eleve így működik a regex motor, azért éles bevetés előtt mindeképpen tesztelni kell ékezetes betűkkel is az egyezéseket!
Normál módban:

```
→\w\w\w←
Született: 1492. 08. 12.
0 => Szü
1 => let
2 => ett
3 => 149
```

RegexOptions.ECMAScript esetén látható, hogy az ü betű nem tartozik bele a →\w←-be, így az első hármas betűcsoportot csak a „let"-nél találja meg a motor:

```
→\w\w\w←
Született: 1492. 08. 12.
0 => let
1 => ett
2 => 149
```

Mi a helyzet az euro (€) karakterrel? Az betű? Egyáltalán, hol található ez a billentyűzeten? Nos, első körben én sem találtam. Aztán rákerestem az euro-ra az unicode.org-on **[1]**, ahol találtam egy riportot **[2]**, mely szerint az euro karakter a 2.1-es unicode szabványban jelent meg. Most egyébként *(2003. október)* a 4.0 unicode szabvány az aktuális.
Nos, az euro a 20AC hexa kódot kapta. Szép, mi? Hol vannak már az ASCII 7 és 8 bites számokkal ábrázolt betűk!
Kitartóbbak a Character Map-ban is megtalálhatják, legalábbis XP+SP1-en biztos:

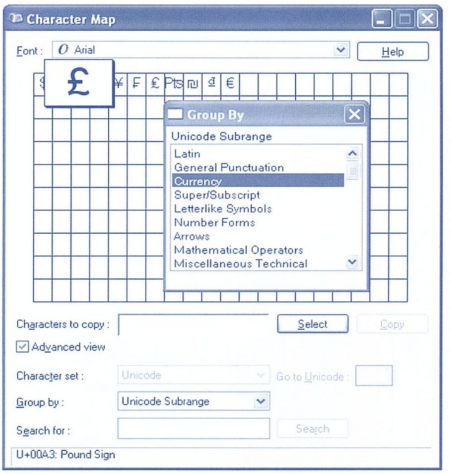


Euro szimbólum a Windows XP Character Map-ben

Nos, a kérdés ugye az, hogy a →\w←-be beletartozik-e az euro? Rövid teszttel kiderül, hogy nem. Az euro szimbólum kategóriájú karakter, nem betű. A .NET regex motor nagyfokú unicode támogatást ad, így a karakterek osztályozásánál is az unicode szabvány által meghatározott kategóriákat használja [3]. Olyannyira, hogy ezt még ki is vezették nekünk. A →\p{Kategórianév}← kifejezéssel hivatkozhatunk a megfelelő kategóriába tartozó karakterekre. A kategóriák nevét a **[3]** táblázat tartalmazza. Nézzünk néhány érdekes példát!

```
→\p{Ll}← #Letter, lowercase, azaz kisbetű
Született A Nagybetű € $ % ^ 1/2 1/4 .+
→\p{Lu}← #Letter, uppercase, azaz nagybetű
Született A Nagybetű € $ % ^ 1/2 1/4 .+
→\p{No}← #Number, other, azaz egyéb szám
Született A Nagybetű 45 € $ % ^ 1/2 1/4 .+
→\p{Ps}← #Punctuation, open, azaz nyitó írásjel
(alma) [körte] {szilva}
→\p{Po}← #Punctuation, other, egyéb írásjel
€ $ % ^ 1/2 1/4 .+ - _ "Idézet.", 'ez is!'
→\p{Sc}← #Symbol, currency, pénz szimbólum
€ $ % ^ 1/2 1/4 . + - _ / * ^ ~ Ft £ ¥
```

Csak meglett az euró, a pénz szimbólumok között találjuk!
A második alkategória karaktert el lehet hagyni, így a →\p{L}← az összes betűt jelenti, amiben nincsenek benne a számok és az aláhúzásjel, azaz nem ugyanaz, mint a →\w←.
Egyébként a furcsa karakterek megkereséséhez hasznos lehet a Character Map Advanced nézete, amikor Unicode kategóriák szerint szűrve láthatjuk a karaktereket.



Csak a pénzeket reprezentáló karakterek

A →\W← *(nagy W)* minden nem betűt jelöl, azaz →^\w←. A →\s← bármilyen whitespace karaktert jelent. A whitespace-eket is az Unicode szabvány rögzíti, de leegyszerűsítve a szóköz, tabulátor, kocsivissza és a soremelés tartozik bele. Vannak még extra karakterek is *(pl. függőleges tabulátor)*, de ezek általában nem érdekesek számunkra. A pontos lista így néz ki:

```
→[\f\n\r\t\v\x85\p{Z}]←
```

A Z unicode kategória a szeparátor karaktereket jelöli, a 85 hexa kódú karakter pedig a szóköz nem PC-s rendszerekben *(bizarr)*.

```
→\s←
alma,_majd_egy_tab:____és_más
```

A →\S← *(nagy S)* minden nem whitespace-t jelöl, azaz →^\s←.

Az univerzális dzsóker: a pont

A →.← *(pont)* bármilyen karakterrel egyezik kivéve az újsor *(\n)* karaktert. Ha RegexOptions.Singleline módban vagyunk, akkor az újsorral is.

```
→.t.←
Született_ma - 2 egyezés
0 => ete
1 => tt(space)
```

```
→.....←
Született<i>ma</i> - 3 egyezés
0 => Szüle
1 => tett<
2 => i>ma<
```

Normál módú viselkedés *(a \r\n a kocsivissza-soremelés páros, amik nem látszanának)*:

```
→.......← #7 db karakter
Első sor(\r\n)
Második sor(\r\n)
```

Látható, hogy az első sorban már nem volt másik 7 egybefüggő karakterhalmaz, így csak egy egyezést tapasztalunk. A második sorban úgyszintén.
Ezzel szemben RegexOptions.Singleline módban a sorvégi újsor *(\n)* karakter nem állítja meg a motort, így a sorokon átívelve is egyezést talál a →.....←:

```
Első sor(\r\n)
Második sor(\r\n)
0 => Első so
1 => r(\r\n)Máso
2 => dik sor
```

A második egyezésben benne van az első sor „r" betűje, a „kocsivissza-soremelés" páros és a „Máso" karakterek.
A pontot nagyon gyakran fogjuk arra használni, hogy ismeretlen szövegre állítsunk fel egyezéseket. A következőkben sok példát fogunk látni a használatára.

Pozícionális karkterek *(Anchors vagy Atomic Zero-Width Assertions)*

Az eddig látott karakterosztályok, jokerek mindig helyet foglaltak el, azaz miután a regex motor egyezést talált, továbblép egy karakterpozícióval mind a forrásszövegben mind a regexben, és onnan keres a regex maradékára egyezést. Például a →.t.← esetén *(tesztsztring: Született ma)* az első →.← talál egy karaktert, az „S"-t, majd a motor továbblép a regexben a →t←-re, és megnézi, hogy az illeszkedik-e a sorok következő karakterre, a „z"-re. Mivel nem, eldobja ezt a próbálkozást, és továbblép a forrásszövegben a „z"-re, visszatekeri a regexet az elejére, és nekiáll a →.←-ot újra ráilleszteni az „ü" karakterre, stb.
Azaz a lényeg, hogy a normál karakter vagy karakterosztály egyezések helyet foglalnak el, továbbléptetik a forrásszöveget. Ezzel szemben a pozícionális karakterek nem foglalnak el helyet, csak kijelölnek egy pozíciót a forrásszövegben, aminek teljesülni kell ahhoz, hogy egyezést kapjunk.
A →^← *(kalap)* a szöveg vagy sor elejét jelenti. Alapban szöveg elejét, RegexOptions.Multiline esetén a sor elejét. Azaz a multiline üzemmódban a regex motor soronként dolgozza fel a szöveget, hasonlóan a grep-hez. *(A normál eset a sed működéséhez hasonló.)*
Normál mód:

```
→^..←
Első sor(\r\n)
Második sor(\r\n)
```

Multiline mód:

```
→^..←
Első sor(\r\n)
Második sor(\r\n)
```

A →$← *(dollár)* a szöveg vagy sor végét jelenti. A multiline ugyanúgy hat rá, mint a →^←-ra.

```
→ma$←
alma alma
```

A második példa csak azokra a sorokra mutat egyezést, amelyekben pontosan és csakis az „alma" szó szerepel:

```
→^alma$←
alma
almama
```

A

```
→^←
```

minden sorra egyezést mutat *(multiline módban)*, azaz nem túl praktikus regex. Habár, ha ezzel a kifejezéssel szétszedünk darabjaira egy szöveget, sorokra bontva kapjuk vissza. Mondjuk ennél egy String.Split egy cseppet gyorsabb lenne, de ezt azért még lehet tovább alakítani, például szűrni a sorokat.
A

```
→^$←
```

az üres sorokat válogatja ki, amikben még whitespace sincs *(természetesen ez is multiline módban működik jól)*.
A →\b← szóhatáron egyezésre való. Szóhatár a →\w← és →\W← átmenet, tetszőleges irányban, így a →\b← szó elejének és végének keresésére is jó.
Baloldalt behatárolt „al" sztring:

```
→\bal←
szilva alma hatalmas almamáter
```

Az „alma" szó keresése kétoldali határral:

```
→\balma\b←
szilva alma hatalmas almamáter
```

Egybetűs, kéttagú urn-ek keresése:

```
→\burn:\w:\w\b←
Például: urn:a:b (vagy urn:x:y)
```

A →\B← *(nagy B)* nem szóhatáron egyezést jelöl ki.

```
→\Balma←
A hatalmas alma hatalma.
```

A →\A← olyan mint a →^←, csak mindig a szöveg és nem a sor elejét jelenti függetlenül a multiline opciótól.
A →\z← *(figyelem, kis z)* pedig olyan mint a →$←, csak mindig a szöveg és nem a sor végére mutat egyezést.
A →\Z← *(nagy Z)* annyival engedékenyebb, mint a kisbetűs párja, hogy a szöveg végén még lehet egy plusz soremelés is. Ez amúgy igaz a →$←-ra is.

Vannak még további pozícionális kifejezések is *(pl. (?!...))*, amelyekre most terjedelmi okokból nem térek ki. **[6]** mindegyiket tárgyalja.
Pozícionális karktereknél nagyon oda kell figyelni, hogy Windowsban a sorok vége nem \n, hanem \r\n, ami miatt sokszor nem jól működnek a Unixon helyesen zenélő regexek. Agyrém, de erre fel kell készülni.

Számosság *(Quantifiers vagy Modifiers)*

Amit eddig láttunk, az csak a jéghegy csúcsa. A regexek első igazi erőssége a számosság adta flexibilitás.
Miről is van szó? Az →a← egy darab a betűt jelöl, fogyaszt el. Az →a?← *(„a" betű, utána egy kérdőjel)* viszont azt jelenti, hogy az „a" karakter 0 vagy egyszeri előfordulása. Ez azt jelenti, hogy akkor is egyezést mutat, ha az adott pozíción van „a" betű, de akkor is, ha nincs. Azaz a kérdőjel jelentése: az előtte levő karakter vagy karakterosztály opcionális.
Az alábbi példa törtszámokat próbál meg elcsípni, a tört rész opcionális:

```
→\d\d\.?\d?\d?←
13.85, 12.5, 15., 45
```

A →\.← a pont karaktert jelöli, csak mivel az metakarakter, meg kellett védeni egy visszafele perjellel. Látható, hogy csak az első két számjegy kötelező, az utána következő karakterek nem. Sajnos ez a regex megengedi a „15."-t is, ami nem szabályos. Amíg nem ismerjük a csoportosítást, addig ezen nem tudunk segíteni.
A →+← 1 vagy több *(legalább 1)* előfordulást jelöl. A példa az összefüggő számsorokat keresi meg:

```
→\d+←
12, 5445, 12.345, 0.33
```

A →*← 0 vagy több *(bármennyi)* számosságot definiál. A ponttal együtt használva könnyedén leírható a bármiből bármennyit minta:

```
→.*←
alma  - 1 találat
```

Ha belegondolunk, ez a minta mindenre egyezik még az üres sorra is, és a bármilyen karaktereket tartalmazó sorokra is.
További számossági jelzők is léteznek. A →{n}← jelentése: pontosan n előfordulás. A példa három összefüggő betűt keres:

```
→\w{3}←
Cica, kutya, sas, őz, ló, kecske
```

Azaz nem hárombetűs szavakat keresünk, ahhoz be kell vetnünk a szóhatárt is, mindkét oldalról:

```
\b\w{3}\b
Cica, kutya, sas, őz, ló, kecske
```

→{n,}←: legalább n találat. Minimum 3 digites egész számok keresése:

```
→\d{3,}←
123, 5445, 12.345, 0.33, alma58942-szilva
```

Láthatóan a tizedes tört utáni részt is megtalálja. Hogy azt kiszűrjük még okosítani kell a regexünket *(→(?<!\.)\d{3,}←, az érdeklődők kedvéért :)* .

És végül a →{n,m}← legalább n, de legfeljebb m darabszámot ír elő.

```
→g{1,2}y←
gyurgyalag, aggyad má
```

Az összes eddig látott számosságjelző mohó, azaz megpróbál olyan hosszú egyezést összehozni amennyit csak lehetséges. Például a →\w{3,}← megpróbálja a lehető leghosszabb, de minimum három karakter hosszú betűcsoportokat megtalálni:

```
→\w{3,}←
Cica, kutya, sas, őz, ló, kecske - 4 találat
```

Azért mohó, mert nem elégszik meg a minimálisan megkövetelt darabszámmal.
Bármelyik számosságjelző mohóságát elvehetjük, ha mögé rakunk egy kérdőjelet:

```
→\w{3,}?←
Cica, kutya, sas, őz, ló, kecske - 5 találat
0 => Cic
1 => kut
2 => sas
3 => kec
4 => ske
```

Látható, hogy most megáll a feltételt minimálisan kielégítő számú karakternél, aztán folytatja a keresét. Ezért vágta ketté a kecskét.
Gyakoroljuk kicsit az eddigieket! Hogyan keresnénk meg a kikommentezett sorokat *(//)* C# kódban?

```
→???←
int i;
//long g;
  // dim i as Integer
```

Hogyan gondolkodunk? Soreleje, majd jön utána akárhány darab whitespace, majd két egymás utáni perjel, aztán a sorvégéig bármi. Elég könnyű lefordítani regexre:

```
→^\s*//.*$←
```

Csoportosítások *(grouping)*

Következő hatalmas fegyverünk a csoportosítás, melyet zárójelezéssel érünk el. Többféle okból csoportosítunk:

- a csoportokra használhatunk számossági jelzőket
- a csoportok által megfogott tartalomra hivatkozhatunk a regex többi részében *(backreferences)*
- a csoportok által elkapott tartalmat kinyerhetjük programozott eszközökkel

Mivel a számosság használható csoportokra, a korábbi törtszámokat kereső regexünket mostmár tökéletesre írhatjuk:

```
→\d+(\.\d+)?←
13.85, 12.5, 15., 45
```

Magyarra fordítva: minimum egy decimális számjegy, aztán egy opcionális csoport, ami belül úgy néz ki, hogy egy pont, aztán minimum egy számjegy. Azaz csak akkor fogjuk meg az egész rész után álló pontot, ha utána van számjegy, egyébként nem.
Egyszerű, de nem teljes email ellenőrző:

```
→\w+@\w+(\.\w+)+←
soci@netacademia.net, alma%@sexybabes.com
soci12@alma.korte.neta.com
```

Visszahivatkoz41 *eferences)*

Tegyük fel, hogy html tagok közötti kifejezéseket akarunk leírni regexszel. Első nekibuzdulásunkban megszüljük ezt:

```
→<\w+>[\w\s]*</\w+>←
<h1>alma</h1> és <h2> körte </h2> <p></p>
```

Az eddigiek alapján ennek teljesen érhetőnek kell lennie. Igen ám, de ez könnyen átverhető:

```
<h1>alma</xxx> - hibás!
```

Megeszi ezt is. Valahogyan meg kellene mondani, hogy a második kacsacsőrös részben azt akarjuk látni, amit az elsőben elkapott a regex motor.
Ehhez először be kell zárójeleznünk az elkapandó kifejezést:

```
→<(\w+)>[\w\s]*</\w+>←
```

Ez nem változtat semmit a kifejezés működésén, de a regex motor már tudja, hogy valami célunk van a zárójeles kifejezéssel, ezért megjegyzi azt.
Már csak az a dolgunk, hogy a zárótagnál hivatkozzunk a zárójeles tartalomra. Erre való a visszahivatkozó kifejezés:

```
<(\w+)>[\w\s]*</\1>
<h1>alma</xxx> és <h2> körte</h2>  <p></p>
```

A →\1← azt jelzi, hogy itt olyan tartalmat várunk el, amit a balról legelső zárójeles kifejezés fogott meg. Fontos megjegyezni a szabályt, balról az n., mert egymásba ágyazott zárójelek esetén így könnyű megtalálni mire akarunk hivatkozni. Az →\2← a második, ... kifejezésre hivatkozik. A visszafelé hivatkozás hatalmas lehetőség a regexekben, és ilyet csak az NFA motorok tudnak, ezért aztán a legtöbb engine NFA.

Elágazások *(Alteration)*

Ha a karakterosztályoknál megadhattunk választást egy karakterpozíción, akkor ezt miért ne tehetnénk meg nagyobb regex kifejezésekre is? Erre való az elágazás, melyet a →|← *(pipe, cső, függőleges vonal)* szimbólum reprezentál. A következő regex jelentése: „alma" vagy „körte" karakterek egymásutánisága:

```
→alma|körte←
alma, körte, körtealma, cser, hatalmas - 5 egyezés
```

Sokféle kommentet kereső kifejezés:

```
→^\s*(//|#|rem|').*$←
int i;
//long g;
        ' dim i as Integer
  rem dos komment
# unix comment
```

Az elágazások bármelyike lehet összetett regex is. A következő példa által ellenőrzött értékek behatárolását a kedves olvasóra bízom.

```
→\b0\d\b|\b1\d\b|\b2[0-3]\b←
15, 28, 21, 14, 5, 142
```

Gondolkodtató példák

Exponenciális számokat felfedező regex:

```
→((\d+)?\.)?\d+e[+-]?\d+←
3e8, 4e+4, 5e-8, 45.6e-5, .34e-6
```

Fájl elérési útból a fájlnevet kiszedő kifejezés:

```
→[^/]*$←
/winnt/system32/drivers/etc/lmhosts.sam
```

Mohó számosság esetén az első .* felemészt mindent, a második kifejezésnek csak a legutolsó szakaszt hagyja meg:

```
→^(.*)/(.*)$←
winnt/system32/drivers/etc/hosts.txt
0 => winnt/system32/drivers/etc
1 => hosts.txt
```

A mohóságát csillapítva megelégszik az első /-ig tartó legrövidebb kifejezéssel:

```
→^(.*?)/(.*)$←
winnt/system32/drivers/etc/hosts.txt
0 => winnt
1 => system32/drivers/etc/hosts.txt
```

Mi történik, ha a második csillag mohóságát is elvesszük? Semmi változás nem történik, mert miután az első kifejezés önmegtartóztató módon csak a „winnt" karaktersorozattal egyezik, a második minden visszafogottsága ellenére kénytelen elvinni a többit.
És a teljesség kedvéért: ha az első mohó a második nem, az első felszed mindent az utolsó perjelig, így a második kapja a maradék részt, ha mohó, ha nem.
Html tagek kitakarítása, például fórum szoftverekhez:

```
→</?\w+[^>]*>←
<html><head>
```

```
<link rel="stylesheet" type="text/css"
href="/css/main.css">
<title>NetAcademia - A legjobbakat
tanítjuk</title>
</head>
<b>Tanfolyami térképek:</b><br>
</html>
```

Az eddigiek után a kiinduló példánkban szereplő regex már gyerekjáték kell legyen:

```
→\b(\w+)(\s|<[^>]+>)+(\1)\b←
```

Ha nem, akkor érdemes újra elolvasni a cikket, és tesztprogramokkal *([4] és [5])* próbálgatni a kódokat *(leglább kiderül, mennyi bug maradt benne :)*. Ez a leggyorsabb módja a regex tanulásának.
Utolsó példaként tetszőleges szepatárorokkal elválaszott, de év-hó-nap formátumú dátumok megtalálását nézzük meg:

```
→\D*(\d\d\d\d)\D(\d\d)\D(\d\d)\D*←
```

Hogy ezen példát hasznunkra tudjuk fordítani itt az ideje, hogy megnézzük programozott módon hogyan lehet elérni a regex szolgáltatásokat.

Regex programozás a .NET Frameworkben

Kiinduló oszályunk a System.Text.RegularExpressions.Regex lesz. Ez képes eltárolni egy regexet, amit aztán rászabadíthatunk egy sztringre.
Létrehozásakor megadhatjuk a kívánt regexet stringként, illetve a működési opciókat a RegexOptions enumerációs típus segítségével:

```
RegexOptions options = RegexOptions.IgnoreCase;
Regex regex =
  new Regex(@"\b(\w+)(\s|<[^>]+>)+(\1)\b",
options);
```

Esetünkben lényeges, hogy a kis-nagybetű különbség ellenére két szót azonosnak tekintsünk, ezért a RegexOptions.IgnoreCase opció.
A regex által elkapott darabokat a következőképpen kaphatjuk meg:

```
MatchCollection matches = regex.Matches(input);
```

Az input a bemeneti stringünk. Az eredményeken könnyű végigiterálni:

```
foreach(Match match in matches)
{
  Console.WriteLine("Pos: {0} ", match.Index);
  Console.WriteLine("1. szó  : {0} ",
                    match.Groups[1]);
  Console.WriteLine("Ismétlés: {0} ",
                    match.Groups[3]);
}
```

A MatchCollection az összes megtalált kifejezést tartalmazza.

Ezeket egyedi Match objektumokként érhetjük el a ciklusban. Minden egyes Match tartalmazza azt a szöveget, amit a regex elkapott. A Match.Index az egyezés pozícióját adja vissza a bemeneti szövegben.
Nekünk csak az első és a harmadik zárójeles kifejezés az érdekes, a középső nem, annak csak az volt a dolga, hogy lenyelje a két szó közötti html tagokat és whitespace-eket.
Szerencsére a zárójelezett tartalmakat közvetlenül elérhetjük a Match objektum Groups kollekcióján keresztül.
A 0. csoport mindig a teljes Match-et tartalmazza, ezért az első zárójeles regex *(→(\w+)←)* által elkapott tartalmat az első Group elemben érhetjük el:

```
Console.WriteLine("1. szó  : {0} ",
match.Groups[1]);
```

Értelemszerűen a 3. csoport a match.Groups**[3]** mögött lesz. Mit találunk a második csoportban? Nos, ez csak az igazán érdekes.

```
→(\s|<[^>]+>)+←
```

Ez a csoport akár többször is szerepelhet az egyezésben, ezért ezt nem lehet egyszerűen match.Groups**[2]**-ként elérni.
Ha a bementi sztring

```
Ez is <i>dadogásnak</i>  <u>Dadogásnak</u> számít.
```

akkor a match.Groups**[2]**.Value-ban „<u>"-t találunk. Pedig ha belegondolunk, a második csoport 4-szer is működött: egyszer elkapta a "</i>"-t *(→</|^>]+>←)*, aztán egy szóközt *(→\s←)*, aztán mégegyet, majd a „<u>"-t. A Value jellemző csak az utoljára elkapott darabkát adja vissza!
Az összeset a csoport Captures jellemzőjén kereszül szedhetjük elő:

```
int i = 0;
foreach(Capture c in match.Groups[2].Captures)
{
  Console.WriteLine("{0}.: {1}", i++, c.Value);
}

0.: </i>
1.:
2.:
3.: <u>
```

A dátumnormalizálós példánkhoz az elkapott csoportok tartalmából kell összeállítani egy formázott dátumot, és azzal kell kicserélni a talált, rosszul formázott dátumnak látszó sztringet:

```
private void Run()
{
  string[] dates =
  {
    "2003/08/12",
    "2003.08.12",
    "2003.08.12....",
    "aa2003/08.12z"
  };
  foreach(string d in dates)
```

```
  {
    Console.WriteLine("{0} -> {1}",
    d, Normalize(d));
  }
}

public string Normalize(string date)
{
  Regex r =
  new Regex(@"\D*(\d\d\d\d)\D(\d\d)\D(\d\d)\D*");
  return r.Replace(date, "$1-$2-$3");
}

2003/08/12 -> 2003-08-12
2003.08.12 -> 2003-08-12
2003.08.12.... -> 2003-08-12
aa2003/08.12z -> 2003-08-12
```

A cserét a Regex.Replace hajtja végre. A $n kifejezésekkel az n. elkapott csoportra hivatkozhatunk, hasonlóan a visszahivatkozások →\n←-jéhez.
A következő példa hyperlinkeket gyűjt ki egy html lapból. A találatokat most alternatív módon érjük el:

```
Regex r;  Match m;

r = new Regex(@"href\s*=\s*""([^""]*)""",
  RegexOptions.IgnoreCase);
for (m = r.Match(ReadTestFile());
  m.Success;
  m = m.NextMatch())
{
  Console.WriteLine("href: {0}, pozíció: {1} ",
    m.Groups[1], m.Groups[1].Index);
}

href: /training/course.aspx?id=2124, pozíció:
2843
href: /training/course.aspx?id=2273, pozíció:
3985
```

Ez utóbbi módszer előnye, hogy a regex egyeztetés lépésenként megy végbe, így a ciklusból idő előtt kilépve a maradék részen nem kell dolgozni a motornak.

Zárszó

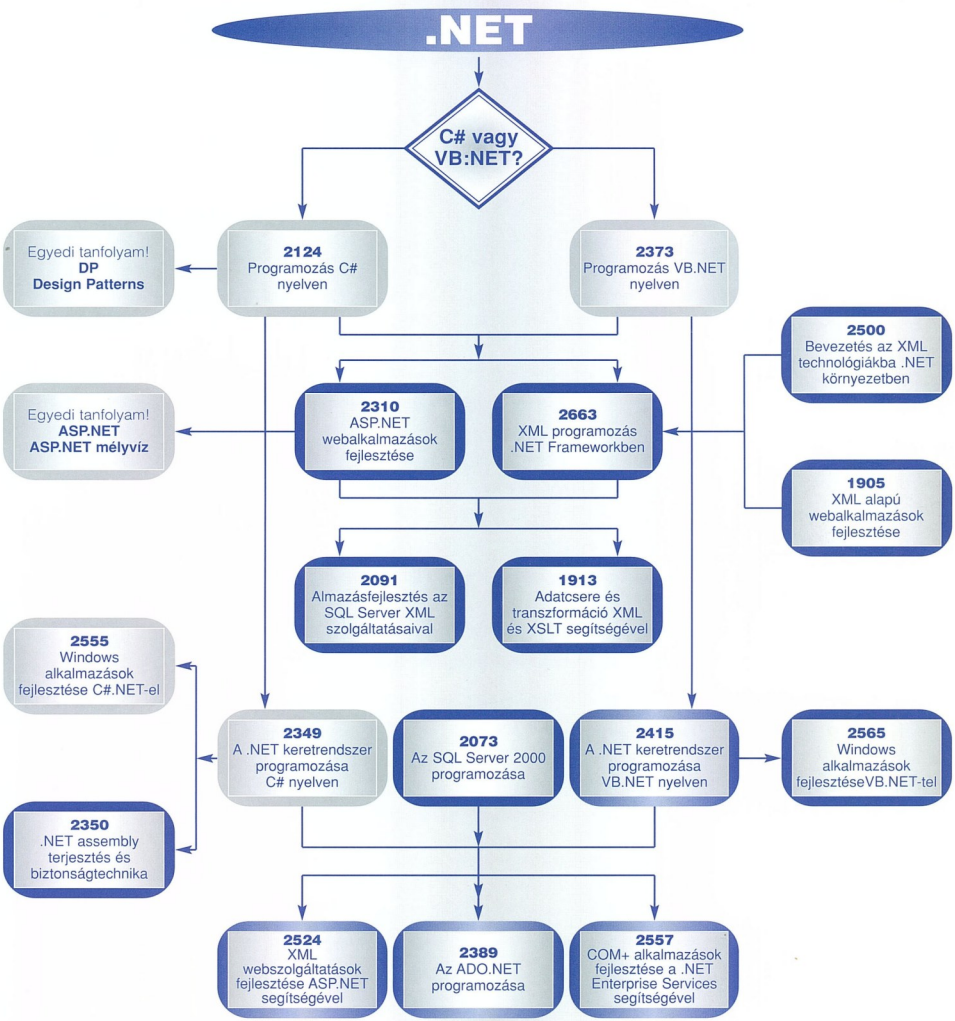
Az eddigiek megértése után gyarkorlati munkák előtt még érdemes áttekinteni a .NET Framework regexekkel foglalkozó fejezetét **[6]**, ugyanis jóval gazdagabb csoportosítási lehetőségek is vannak még, mint amelyekről a cikkben olvashattak. Jó regexelést!

Soczó Zsolt
zsolt.soczo@netacademia.net
A szerző a NetAcademia vezető fejlesztési oktatója,
MCSE, MCDBA, MCSD.NET, MCT

A cikkben szereplő URL-ek:

[1] http://www.unicode.org/
[2] http://www.unicode.org/reports/tr8/index.html
[3] http://tinyurl.com/qw7c
[4] http://www.sellsbrothers.com/tools/#regexd
[5] http://tinyurl.com/ny0f
[6] http://tinyurl.com/atlv

Fejlesztői tanfolyamok térképe



www.netacademia.net • 1062 Budapest, Andrássy út 62. • Telefon: (1) 472-1214 • Fax: (1) 472-1215

A NetAcademia Kft. évvégi tanfolyami kínálata rendszergazdáknak és fejlesztőknek

Ha maradt még felhasználatlan **oktatási kerete** 2003-ra, itt az utolsó lehetőség, hogy az Ön igényeinek megfelelően használja fel! Látogasson el évvégi tanfolyamainkra!
Oktatóink **magas szakértelme** garanciát jelent arra, hogy Ön a legtöbbet profitálja tanfolyamainkból.

Kód	Rendszergazdai tanfolyamok	Hossz	Időpont	Ár*
2433	Visual Basic Scripting Edition and Windows Script Host Essentials	3 nap	2003. november 24 -26.	110.000 Ft/fő
PKI	PKI-workshop - nyílt kulcsú titkosítás a gyakorlatban	2 nap	2003. november 27-28.	140.000 Ft/fő
2159	Deploying and Managing Microsoft ISA Server 2000	3 nap	2003. december 1-3.	110.000 Ft/fő
2072	2072 - Administering a Microsoft SQL Server 2000 Database	5 nap	2003. december 8-12.	150.000 Ft/fő

Oktató: Fóti Marcell

Kód	Fejlesztői tanfolyamok	Hossz	Időpont	Ár*
2555	Windows alkalmazások fejlesztése C#.NET/VB.NET-tel	5 nap	2003.november 24 -28.	160.000 Ft/fő
2524	XML Webszolgáltatások fejlesztése ASP.NET segítségével	3 nap	2003.december 1-3.	135.000 Ft/fő
2124	Programozás C# nyelven	5 nap	2003.december 8-12.	160.000 Ft/fő
2310	ASP.NET webalkalmazások fejlesztése VS .NET segítségével	5 nap	2003.december 15-19.	160.000 Ft/fő

Oktató: Soczó Zsolt

További tanfolyamok között honlapunkon válogathat: www.netacademia.net
Kérje 2003/2004-re vonatkozó ingyenes katalógusunkat! info@netacademia.net

Tegye színesebbé a szürke téli hétköznapokat NetAcademia tanfolyammal!

Jelentkezzen most!
A túloldalon található jelentkezési lapot kitöltve kérjük faxolja el a **472-1215**-ös faxszámra. Csoportos jelentkezés esetén kérjen árajánlatot (472-1214).

* Áraink az ÁFA-t nem tartalmazzák.

NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

Microsoft CERTIFIED Partner

Jelentkezési lap

Kérjük az alábbi megrendelőt kitöltve küldje vissza a fent megadott faxszámra vagy e-mail címre. Felhívjuk szíves figyelmét, hogy a jelentkezési lap leadása megrendelésnek számít.

A tanfolyami jelentkezéseket a tanfolyam kezdete előtt 1 héttel lezárjuk.

Megrendelő cég neve:	
Résztvevő(k) neve:	
Számlázási cím:	
Telefon / Fax szám:	
E-mail cím	
Tanfolyam neve és kódszáma:	
Tanfolyam időpontja:	
Tanfolyam díja: (tartalmazza a tananyag és az ebéd költségét)	

A tanfolyammal kapcsolatos változtatások jogát fenntartjuk.

Lemondás: A tanfolyam a kihirdetett kezdési időpontot megelőző 14 naptári nappal mondható le díjmentesen. A tanfolyam kezdetét megelőző 14-3 naptári napig lemondás esetén a tanfolyami díj 80%-át térítjük vissza. A kezdetet megelőző 3 naptári napon belül lemondást nem áll módunkban elfogadni.

Honnan értesült a tanfolyamról?

☐ NetAcademia hírlevélből ☐ Katalógusból ☐ Kollégától ☐ tech.net magazinból ☐ Egyéb

Vegetáriánus ebédet kér?

☐ Igen ☐ Nem

Dátum:

Pecsét

Cégszerű aláírás

H-1062 Budapest, Andrássy út 62. ● Tel.: + 36 1 472 1214 ● Fax: +36 1 472 1215 ● E-mail: info@netacademia.net
http://www.netacademia.net

„Sajnálattal értesítjük, hogy a tanfolyam a jelentkezők kevés száma miatt elmarad...”

Ugye Ön se szereti az ilyen üzeneteket?

Menjen biztosra!

Az alábbi tanfolyamaink biztosan indulnak. Jelentkezzen, amíg vannak szabad helyek!

A SZÁMALK Továbbképzésnél az őszi/téli időszakra még több mint 10, különböző témakörű képzésből válogathat, a Windows 2000 témáktól kezdve a szerver termékeken át a fejlesztőeszközökig.

November 10–14.:	SQL Server 2000 adatbázis implementáció és programozás (2073)
November 17–21.:	Windows 2000 rendszeradminisztráció (2126)
November 24–25.:	SQL Server 2000 alapok, lekérdezések, Transact SQL (2071)
November 24–28.:	Windows 2000 hálózatinfrastruktúra üzemeltetés (2153)
November 26–28.:	Adatkezelés Visual Studio .NET-tel (2389)
December 1–5.:	Exchange Server 2000 üzemeltetési ismeretek (1572)
December 1–5.:	Windows 2000 üzemeltetési ismeretek (2152)
December 1–5.:	Web alkalmazások fejlesztése Visual Studio .NET-tel (2310)
December 8–12.:	SQL Server 2000 üzemeltetés, adatbázis adminisztráció (2072)
December 15–19.:	Windows 2000 Active Directory adminisztráció és tervezés (2154+1561)
2004. január 26–30.:	Windows alkalmazásfejlesztés Visual Studio .NET-tel (2555/2565)
2004. február 23–27.:	Web alkalmazások fejlesztése Visual Studio .NET-tel (2310)
2004. március 22–24.:	XML webszolgáltatások fejlesztése Visual Studio .NET-tel (2524)

A kínálat frissülhet és változhat, kövesse nyomon internetes tanfolyamlesünket!

Microsoft CERTIFIED Professional

Microsoft GOLD CERTIFIED Partner

Microsoft CERTIFIED Technical Education Center

SZÁMALK

www.szamalk.hu/tisza

Tel.: 203-0304/3050 (Simon Feren
1115 Budapest, Etele út 68.