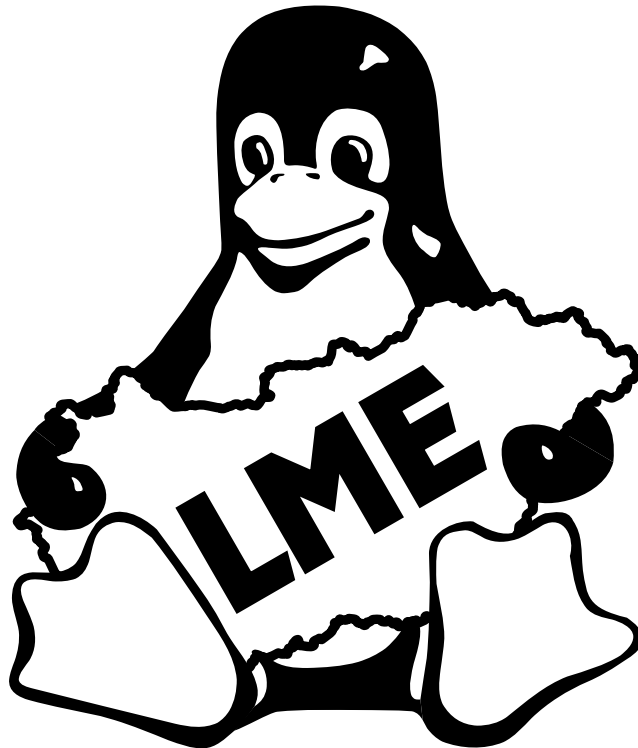


Linux 2000 Konferencia



Budapest, 2000. október 7.

A kiadvány tördelése a \TeX 3.14159 verziójával készült
Linux operációs rendszeren.
A \TeX az American Mathematical Society bejegyzett védjegye.

Szerkesztette: *Zelena Endre* ezelena@lme.linux.hu

Linux-felhasználók Magyarországi Egyesülete
1395 Budapest 62, Pf. 432.
URL: <http://lme.linux.hu/>
E-mail: info@lme.linux.hu

Minden jog fenntartva. Jelen kiadványt, illetve annak részeit reprodukálni, adatrögzítő rendszerben tárolni, bármilyen formában vagy eszközzel -elektronikus úton, vagy más módon- közölni csak a szerző(k) írásos hozzájárulásával szabad.

Tartalomjegyzék

Tartalomjegyzék	3
A konferencia támogatói	4
Előszó	5
Egyesületünkéről...	6
Előadások	7
Czakó Krisztián: Professzionális e-mail szerverek kialakítása (kivonat)	9
Érdi Gergő: Bonobo, A GNOME CORBA alapú komponens-megoldása Unix-okra	11
Györkő Zoltán: Hardvertokenes autentikációs eszközök (kivonat)	21
Kétszeri Csaba: PHP alapú on-line bolt, integráció a vállalat nem Linux alapú ügyviteli rendszerével	23
Kósa Attila: Alternatíva a vállalati informatikában: a <i>Linux</i>	25
Magosányi Árpád: Hozzáférésvezérlési modellek Linuxon	35
Mátó Péter: A tűz, avagy mi ellen véd a tűzfal? (kivonat)	55
Noll János: Template-rendszerek PHP alatt, a Prim template-parser rendszere (kivonat)	57
Scheidler Balázs: Hálózati határvédelem eszközei (kivonat)	59
Szentpétery Ferenc: Könyvtári karton feldolgozó rendszer	61

A konferencia támogatói

Fő támogató: ProWare Kft.

Médiatámogató: Prim Online Kft.

Támogatók/kiállítók:

- BalaBit IT Kft.
- Ericsson Magyarország
- HUMANsoft Elektronikai Kft.
- IDG Magyarországi Lapkiadó Kft.
- Kiskapu Kft.
- MrSoft Oktatási és Kereskedelmi Kft.
- Pilatus-Comp Kft.

Előszó

Üdvözljük abból az alkalomból, hogy kezében tarthatja a Linux-felhasználók Magyarországi Egyesülete második önállóan megrendezett szakmai konferenciájának kiadványát.

Egyesületünk 1998 őszén azzal a céllal jött létre, hogy összefogja a Linux-szal foglalkozó szakembereket és vállalatokat, szakmai fórumokat teremtsen, széles körben terjessze a Linux-szal kapcsolatos ismereteket, jogi személyiséggel képviselje a „pingvinhívők” hazai társadalmát.

Rövid működésünk alatt is sok eredményt könyvelhetünk el, és büszkén mondhatjuk, hogy képesek vagyunk fejlődni. Az egyre növekvő taglétszámmal együtt az egyesület ereje is növekszik.

A tavaly szeptemberi, első általunk szervezett szakmai konferencia sikerén felbuzdulva elhatároztuk, hogy idén is rendezünk egy hasonlót. Akkor úgy gondoltuk, elegendő egy 400 fős előadóterem, ám aztán már látható lett, hogy alábecsültük a szakma figyelmét. Idén az érdeklődők két szekcióban hallgathatják az előadásokat, két 300 fős teremben. Reméljük, hogy az itt található anyagok, az előadások találkoznak az Ön érdeklődésével, és viszontlátjuk az LME jövőbeli rendezvényein. Értékes észrevételeit, javaslatait, kérjük ossza meg velünk személyesen az LME standján, vagy e-mail-ben az: *info@lme.linux.hu* címen.

Tisztelettel,

a szervezők.

Egyesületünkről...

A Linux-felhasználók Magyarországi Egyesülete (LME) a GNU/Linux szabad operációs rendszer magyarországi megismertetését és elterjesztését tűzte ki célul, ezzel segítve a magyar gazdaság és számítástechnikai kultúra fejlődését.

Egyesületünk már eddig is több szakmai rendezvényen jelen volt. Az Info '99 kiállításon bemutattunk több alkalmazást, valamint válaszoltunk az érdeklődők kérdéseire. Szakmai támogatóként, illetve előadókkal is segítettük az ETB szervezésében májusban megtartott Linux konferenciát. 1999. szeptember 18-án megrendeztük az I. Linux Szakmai Konferenciát a MÁV BEIG épületében, nagy sikerrel. Még az év decemberében rendeztük meg a Linux Mikulást, ahol játékos formában vetélkedhettek a résztvevők. Idén standunkkal megjelentünk az Info2000 kiállításon: terjesztettük egyesületünk szórólapját és a '99-es konferenciakiadványunkat, illetve supportot nyújtottunk az érdeklődőknek. Részt vettünk az ITB Konferencián Nyílt forráskód: lehetősége az EU és a Magyar Közigazgatás számára témakörében, és nagy sikert arattunk. 2000. július 9-én az MTV1-es csatornáján a Delta című műsorban volt egy összeállítás a Linuxról, és egyesületünk elnöke beszélt az LME-ről is. Ez év szeptemberében egyesületünk előadást tartott a NetGeneration Konferencián is.

Egyesületünknek már több csoportja is alakult (Szolnoki, Veszprémi, KDE Csoport), ők a saját területükön végzik a kitűzött célok megvalósítását.

Budapesten rendszeresen ingyenes oktatásokat tartunk a Fővárosi Pedagógiai Intézetben a GNU/Linux alapjairól, használatáról, felépítéséről, valamint hálózatokról, biztonságról. Szolnokon hasonló témákban hallgathatnak előadásokat az érdeklődők.

A GNU/Linux magyarázásának megszervezésében, megvalósításában is tevékeny részt vállalunk, hogy ezáltal könnyítsük a GNU/Linux használatát az angolul nem, vagy csak nehezen értők számára. (Kézikönyvlapok, programüzenetek, dokumentációk fordítása.) Az oktatás és a dokumentáció magyarázása magával hozza egy magyar nyelvű oktatási anyag elkészítését is, amin jelenleg is dolgozunk.

Az egyesület jövőbeni céljai közé tartozik az oktatások rendszeressé tétele, nem csak a jelenlegi két helyszínen. Szeretnénk továbbá megalkotni egy teljesen magyarul beszélő GNU/Linux rendszert. Ezen célok eléréséhez minden segítséget szívesen veszünk.

Amennyiben ön is be szeretne lépni tagjaink sorába, támogatni szeretné az egyesületet, vagy szeretne részt vállalni a munkacsoportokban, kérjük tekintse meg az egyesület honlapját, illetve írjon az info@lme.linux.hu címre.

Előadások



Professzionális e-mail szerverek kialakítása (kivonat)

Czakó Krisztián

2000. szeptember 28.

Napjainkban, amikor egyre több cégnél alkalmazzák a Linuxot különböző feladatokra, fontos, hogy a megoldások hatékony, könnyen kezelhető rendszert adjanak, melyet a minimális szakmai ismeretekkel rendelkező felhasználó is kézben tud tartani, és csak komoly problémák esetén kell szakemberhez fordulni. Az egyik leggyakoribb felhasználási területe a Linuxnak az internetes kapcsolat. Itt most ezen belül az e-mail szerver egy profi, könnyen kezelhető megoldásáról lesz szó.

Az első lépés általában a megfelelő Linux disztribúció kiválasztása. Mi most a Debian GNU/Linux alatti megoldást mutatjuk be, bár a legtöbb lépés más disztribúciók alatt is ugyan az. A második lépés a megfelelő e-mail szoftverek kiválasztása. Itt is széles a választék. Az e-mail rendszer kettő alapvető komponensből áll. Az SMTP szerverből és a klienseket kiszolgáló szerverből, ami lehet POP-3 vagy IMAP. Ez utóbbi kettő között dönteni lehet, de választhatjuk mindkettőt egyszerre. Hasznos kiegészítő még egy egyszerű web-szerver, mely webes adminisztrációt tesz lehetővé, megkönnyítve a későbbi felhasználó hozzáadás, módosítás feladatait és nem utolsósorban lehetővé teszi a Linuxban járatlan felhasználók számára a jelszó változtatását, leveleik átirányítását. Az SMTP szerverre a mi választásunk a QMAIL, mivel tervezésénél és készítésénél a biztonságos működés volt a legfontosabb szempont. A sendmaillel szemben ez egy több különálló komponenset tartalmazó rendszer, ami lehetővé teszi, hogy nem privilegizált felhasználóként fusson a legtöbb funkció. Fontos szempont az is, hogy a leveleket az ún. Maildir rendszerben tárolja (minden levél egy-egy külön fájl), mely a levelek kezelésénél sokkal kisebb erőforrást igényel mint az általánosabban elterjedt mbox formátum (egy mailbox egy fájl). Ez utóbbi főleg az IMAP használatakor jelent nagy előnyt. Találunk a QMAIL-hez egy webes adminisztrációs programot, mely a qmail és kiegészítői teljes adminisztrációját elvégzi, valamint egy egyszerű és gyors webes e-mail kliens is fellelhető.

Biztonsági és kényelmi szempontból sem szerencsés, ha minden e-mail felhasználónknak adunk egy Linux felhasználót annak minden jogával. Itt egy bevett

gyakorlat, hogy olyan shellt állítanak be a felhasználónak, mellyel nem tud belépni, de mi ennél is jobb megoldást mutatunk be. Egy teljesen virtuális e-mail rendszert. Az Interneten ezt vpopmail néven találhatjuk meg és pár alapvető tulajdonsága van, ami megfelelővé teszi számunkra: teljes integráció a qmail, teljesen független virtuális domáinek kezelése, virtuális (csak e-mail) felhasználók és valós Linux felhasználók kezelése egyszerre, egyszerű parancsok domáinek és felhasználók felvételére, jelszavak módosítására, webes adminisztrációs felület, saját POP-3 szerver, önálló autentikációs library, mely lehetővé teszi IMAP szerver illesztését. Az illeszthető, Maildir formátumot használó IMAP szervert Courier-IMAP néven találjuk meg az Interneten. A webes adminisztrációs programot qmailadmin néven, a webes levelezőt sqwebmail néven lelhetjük fel.

A rendszerhez illeszthetünk pár kiegészítőt is. Az autorespond automatikus válaszlevelek küldésére alkalmas (pl info@ címre érkezett levelekhez) valamint találunk egy egyszerű levelezőlista kezelőt is. Természetesen ezek funkciói is elérhetők a webes adminisztrációs felületről.

A programok kiválasztásánál az is szempont volt, hogy azok szabadon és forráskóddal együtt elérhetőek legyenek. A szabad szoftver elveknek nem igazán megfelelő ugyan a qmail, de a licensze csak a módosított forráskód és az az alapján készült binárisok továbbadását tiltja, ami saját felhasználás esetén nem jelent problémát.

Az előadás keretében szó lesz a fenti programok telepítéséről és beállításáról Debian GNU/Linux alatt a csomagok beszerzésétől a működő serverig, valamint bemutatunk pár működő rendszert is, melyet a Pilátus-Comp Kft. készített.

Bonobo: A GNOME CORBA alapú komponens-megoldása Unixokra

Érdi Gergő <cactus@opensource.hu>

2000.09.25.

Kivonat

A Unix rendszerek alapvető segédprogramjait jellemző „tegyél egy dolgot, de azt helyesen”, és „működj együtt más, elemi funkciókat ellátó programokkal” elvek eltűnni látszanak a mai monolitikus, behemót „desktop-alkalmazások” árnyékában.

A GNOME Bonobo rendszere lehetővé teszi a szoftverkomponensek együttműködését a modern alkalmazások igényeit is kielégítve.

Tartalomjegyzék

Bevezető	12
1. IPC megoldások	12
2. Komponensaktiváció: OAF	13
3. A rendszer alapköve: A BONOBO::UNKNOWN	14
4. A rendszer működése egy példán keresztül	15
4.1. A modellalkalmazásunk	15
4.2. Komponensbeillesztés, szerkesztés	15
4.3. Nyomtatás, tárolás	16
4.4. Valódi példák	17
5. A GNOME Bonobo implementációja	17
6. MonkeyBeans	18
Összefoglalás	19

Bevezető

A bonobo (*Pan paniscus*) az óvilági emberszabásúak közé tartozó, veszélyeztetett (jelenleg alig 10,000-re tehető a számuk) majomfaj. A ma élő fajok közül ők hasonlítanak a legjobban a csimpánzok és az emberek közös ősére. Laikusok számára a legfeltűnőbb sajátosságuk az, ami miatt a GNOME komponensrendszerének névadói is lettek: nagyon szoros és gyakori az interakció a fajtársak között. Ez a gyakorlatban kevésbé cizelláltan jelenik meg: fehérmájú, szexőrült nimfománok.¹

A Bonobo rendszer segítségével lehetőséget teremthetünk egymástól hálózatilag és platformilag független programok kommunikációjára. Az előadás során a rendszer általános működése mellett egy-két implementációs részlettel is megismerkedhatsz.

1. IPC megoldások

Mint az a kivonatban is szerepelt, alapvető igény a programok iránt, hogy kommunikálni tudjanak egymással. Ez természetesen különböző szinteken és módokon történhet. Tekintsük át az *inter-process communication* lehetőségeit!

Hagyományos UNIX IPC: pipe, socket

Ezeket az eszközöket minden Unix fejlesztő és felhasználó ismeri. Alacsony szintű, általános adatátviteli mód processzek között, lokálisan vagy hálózaton át. Az alacsony szintűség egyben a fő hátrányuk is: ezek az eszközök „analóg” továbbítják az információt. Képzeld el például, hogy egy olyan alkalmazás készítése a feladatod, amely az *lpq* program kimenete alapján készít valamiféle statisztikát a nyomtatási spool pillanatnyi helyzetéről. Különböző *lpq* implementációk különbözőképpen fogják formázni a kimenetüket, ezáltal lehetetlenné téve a kimenet parse-olását.

CORBA

A CORBA többek között ezt a problémát oldja meg, oly módon, hogy az egyes alkalmazások csak jól definiált interfészeket keresztül kommunikálhatnak egymással. Az egyes implementációk között így „kifelé” elvész a különbség, és (az előbbi példánál maradva) nem kell törődnöd különböző gyártók *lpd* implementációinak

¹Szerintem, tisztán az egyensúly kedvéért, minden előadásnak tartalmaznia kellene egy mondaton belül a „cizellált” és a „szexőrült” szavakat.

különbségével. A *CORBA* további, jól ismert előnyei a kommunikáció teljesen transzparens volta hálózati és/vagy platformi határok között.

COM

A *Microsoft* már egészen régi időktől fogva elkezdte a *Windows* komponensrendszerének kialakítását: bizonyára sokan emlékeznek a *Windows 3.1* platform egyik legprominensebb újdonságát jelentő *OLE (Object Linking and Embedding)* hírértékére. Sok-sok reinkarnációval később, mára a COM rengeteg alkalmazás, vírus és backdoor motorját jelenti.

A COM hátránya a többi IPC megoldáshoz képest, hogy kizárólag *Windows* platformokon érhető el, és (ennek megfelelően) architektúrája is sok mozzanatban lehetetlenné, vagy legalábbis nehézkessé teszi portolását más platformokra.

Bonobo

Az előadás tárgya, a *Bonobo* rendszer a COM alapfelépítésének egyes elemeit veszi át, a tényleges kommunikációt *CORBA* objektumokkal oldva meg – így bárki elkészítheti saját Bonobo implementációját a saját platformjára.²

2. Komponensaktiváció: OAF

Egy komponensalapú alkalmazás nem csak a *code reuse* előnyét nyújtja (ez sok más módon is megoldható), hanem lehetővé teszi, hogy olyan komponenseket is fel tudjon használni a programod, amelyeket tőled független fejlesztők (*ISV*-k) készítettek. Ehhez az szükséges, hogy az alkalmazás válogatni tudjon a különböző telepített, és igényeinek megfelelő komponensek közül. Ez két különböző, de összefüggő lépést jelent: (1) a telepített komponensek lekérdezése és (2) a kívánt komponensek aktivációja.

Elliot Lee *Object Activation Framework* nevű programja ezeket az igényeket hivatott kielégíteni. *CORBA*-n át történik a lekérdezés és az aktiválás, így helyi és távoli objektumok egyaránt elérhetőek. A tényleges aktiváció folyamatakor az OAF gondoskodik a szerverprogram elindításáról (vagy ha az egy *shared object*-ben található, betöltéséről és linkeléséről), és a felhasználó már a „készre szerelt” *CORBA* objektumot kapja kézhez.

A kiválasztható komponensek listáját az OAF egy, saját lekérdezőnyelvén írt kérésre juttatja el a programhoz. Ez a lekérdezés megszorításokat tartalmazhat

²Ez így nem százszázalékosan igaz, a Bonobo célpontját a Unix rendszerek jelentik, így ennek megfelelően egyes pontjainak implementálása nem Unix-szerű rendszereken több-kevesebb problémába ütközhet, lásd még a 6. pontot

például a megvalósított interfészekre („milyen nyomtatható komponensek vannak telepítve”), vagy a fejlesztő által definiált tulajdonságok értékeire (pl. file-megjelenítő komponensek közül azok kiválasztása, amelyek egy adott MIME-típust támogatnak). A lekérdezések végrehajtását az OAF ObjectDirectory CORBA objektumokra bízta, így egy szerver több platform és számítógép komponenslistáit mutathatja egységesen a kliensprogram felé.

A komponensaktivációnak létezik egy magasabb szintű lehetősége is, a `BONOBON::MONIKER`³ interfészen keresztül. A különböző `MONIKER` implementációkkal lehetséges például egy file megnyitása a hozzátartozó komponenssel, vagy egy összetett komponens egy adott darabjának kijelölése (például hivatkozhatunk egy *Gnumeric* táblázatfile egy bizonyos munkalapjának egy cellatartományára).

3. A rendszer alapköve: A `BONOBON::UNKNOWN`

Az előző pontokban dobálóztam már objektumokkal, anélkül, hogy konkrétan megmondtam volna, mire is gondolok. Az `UNKNOWN` interfész képezi a Bonobo rendszer alapját: valamennyi, tényleges feladatot elvégző felület ebből származtatik le. Ez az apró interfész mindössze a következőkből áll:

```
interface Unknown
{
    void ref ();
    void unref ();

    Unknown query_interface (in string repoid);
};
```

Nézzük mit is jelentenek az egyes metódusok:

ref/unref: Az objektumokat biztosító szerverprogramoknak valahogyan lehetőséget kell adnunk arra, hogy gazdálkodni tudjanak az erőforrásaikkal. Erre szolgál a Bonobo referenciaszámlálásos nyilvántartása: amikor az egy objektumra való hivatkozások (tehát azon programok, programrészletek száma, amelyek még fognak kommunikálni az objektummal) nullára esik, a szerverprogram azt felszabadíthatja. Az egyes hívások jól definiáltan változtatnak a referenciaszámlálón, ez alapvető feltétele annak, hogy ne maradjanak „elveszett” hivatkozások.

query_interface: Ennek a metódusnak a segítségével tudhatjuk meg egy komponensről, hogy egy adott interfészt megvalósít-e. Amennyiben igen, az

³A kevésbé ismert angol szó jelentése: becenév.

implementáló objektumot adja vissza (újabb QUERY_INTERFACE hívások az eredeti és az így kapott objektumon definíció szerint ugyanazt az értéket adják vissza). Ez a hívás teszi lehetővé, hogy programunk tetszőleges komponenseket használni tudjon, és futásidőben, dinamikusan „fedezze fel” a képességeiket. A másik, így adódó lehetőség, mint azt a 4. szakaszban látni fogod, az, hogy egy komponens több felületet is megvalósítson (a 4. fejezetben például egy komponenst be lehet illeszteni, és ki is lehet nyomtatni).

4. A rendszer működése egy példán keresztül

4.1. A modellalkalmazásunk

Ebben a fejezetben egy hipotetikus alkalmazás: egy szövegszerkesztő működését mutatom be, az eddigi elméleti információ gyakorlati megjelenését. Szövegszerkesztőnk a Bonobo rendszer segítségével lehetővé teszi felhasználóinak, hogy a dokumentumot alkotó szöveget és a beillesztett objektumokat egységesen kezelje. Lássuk, hogyan is éri ezt el!

4.2. Komponensbeillesztés, szerkesztés

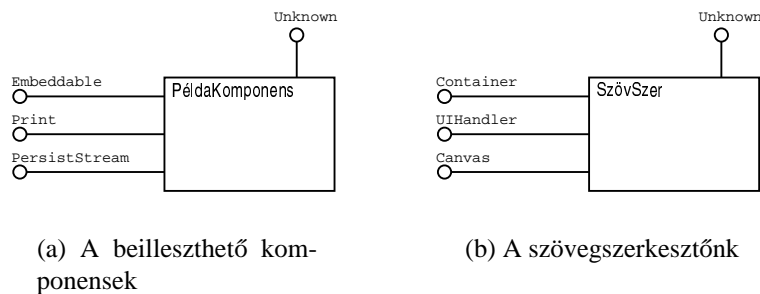
Felhasználónk⁴ szeretne beilleszteni egy képet szövegébe. Ehhez kiválasztja alkalmazásunk megfelelő menüpontját. Ennek tényleges megvalósítása többféle is lehet: például egy file beillesztése a megfelelő MONIKERrel, vagy programunk lekérdezi az OAF szervertől azon komponensek listáját, amelyek megvalósítják az EMBEDDABLE és/vagy CONTROL interfészeket.

Mindkét esetben eredményképpen egy UNKNOWN objektumot kapunk, amelyet vagy rögtön megjeleníthetünk (CONTROL), vagy egy adathoz több nézetet készíthetünk (EMBEDDABLE, VIEW). Ezekután a dokumentumunkban való tényleges megjelenítést az X protokollra bízhatjuk.

Amikor a felhasználó módosítani akar a beillesztett elemen, két megoldás lehetséges: vagy külön ablak nyílik a szerkesztés elvégzésére, vagy pedig (a UI-HANDLER interfész⁵ segítségével) a komponens felhasználói felülete összeolvad a konténerével.

⁴minden baj forrása

⁵a Bonobo 0.19-es verziója óta a UICONTAINER interfész váltja fel



1. ábra. Az eddigi példák során bemutatott komponensek

4.3. Nyomtatás, tárolás

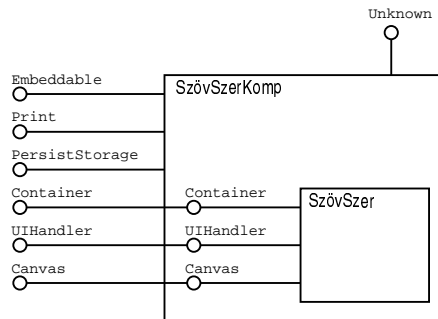
A nyomtatás tulajdonképpen teljesen ugyanúgy történik, mint a beillesztés: amennyiben a komponens megvalósítja a PRINT interfészt, szövegszerkesztőnk kijelöl a papíron egy területet, és megkéri a komponenst, hogy nyomtasson oda.

Példaalkalmazásunk szeretné a saját adatai (a dokumentum) mellett a beillesztett elemek állapotát is rögzíteni. Erre a PERSIST* interfész-család szolgál. A legegyszerűbb esetben a PERSISTFILE felületet használva, egy filenév megadására van csak szükség, és a komponens gondoskodik a file kezeléséről. Ez azonban nem javasolt, mivel így a komponens futási helye szabja meg a file helyét is, később ugyanaz a komponens például egy másik számítógépen futva nem ugyanazokat az adatokat éri el.

Ezen okokból két olyan interfész is része a Bonobo rendszernek, ahol az adatok CORBA-n át jutnak vissza a komponenstől a konténerig. A PERSISTSTREAM interfész segítségével egy STREAM-be, tehát egy egyszerű byte-sorozatba/ból írhatjuk/olvashatjuk adatainkat, a PERSISTSTORAGE segítségével pedig a STORAGE nevű, fastruktúrájú „mini-filerendszerbe”.

Szövegszerkesztő alkalmazásunkkal a fenti lehetőségek megismerése után a teljes dokumentumot egy STORAGE-ban tároljuk. Minden komponensünk kap ezen belül egy STREAM-et vagy egy újabb STORAGE-ot, amibe a megfelelő PERSIST* interfészen át elmentjük az állapotát.

Láttuk tehát, hogyan épül fel modellalkalmazásunk. Az 1. ábra az általunk használt komponensek és a szövegszerkesztőnk durván sematizált szerkezetét mutatja. Természetesen ha lehetővé akarod tenni, hogy más alkalmazások most már a te szövegszerkesztődet is használni tudják, ugyanúgy implementálnod kell a megfelelő interfészeket. A 2. ábra egy olyan megoldást mutat, ahol az elkészített



2. ábra. Beilleszthető szövegszerkesztő komponens

objektumot „becsomagoljuk” (ez a technika akkor hasznos, ha például utólag merül fel a komponensként is viselkedés igénye)

4.4. Valódi példák

Természetesen az ebben a részben bemutatott modellalkalmazás mellett egyre több „valódi” alkalmazás is használja a Bonobo rendszert. Az alábbi lista a teljesség igénye nélkül említi meg néhány fontos, Bonobo-alapú alkalmazást.

Gnumeric táblázatkezelő; a Bonobo hőskorában ez volt az implementáció tesztelési platformja, egyben első felhasználója

AbiWord az *AbiSource* multiplatform szövegszerkesztője; A komponensmodell absztrakciójával elérték, hogy minden platformon a natív rendszert tudják használni (Windows: COM, Unix: Bonobo)

Evolution a *Helix Code* csoportszoftvere (műfaji okok miatt itt érthetően nagyon fontos volt biztosítani az ISV-k lehetőségét a kiegészítésekhez és egyediesítésekhez)

Nautilus az *Eazel* leendő GNOME grafikus shellje

StarOffice a Sun nemrég bejelentette, hogy szándékában áll GPL alatt kiadni a StarOffice új verzióját, amely GTK+-t fog használni a felhasználói felület felépítéséhez, beépül a GNOME rendszerbe, és minden porcikájában felhasználja a Bonobot.

5. A GNOME Bonobo implementációja

A legtöbbször nem a „nyers” Bonobo CORBA hívásokon át kommunikálnak a többi komponenssel, hanem az adott platform és nyelv sajátosságaihoz igazodó,

a felhasználást megkönnyítő API-ken át. A Bonobo rendszer kifejlesztésével párhuzamosan folyt-folyik egy implementáció fejlesztése is.

A GNOME/C Bonobo implementáció fő célja, hogy minél tökéletesebben beépüljön a GTK+ objektumrendszerébe, ezért valamennyi objektuma a *GtkObject* osztály leszármazottja. Ez nem csak a CORBA objektumok *GtkObject*-té történő leképezését jelenti⁶, hanem sok implementációt, amely lehetővé teszi, hogy egy-két függvényhívással működő Bonobo komponenst készítsünk. Ahhoz például, hogy egy elkészített *GTKWIDGET*-ből egy *BONOBO::CONTROL*-t hozz létre, elegendő a *bonobo_control_new (GtkWidget *widget)* hívást használnod. Ezekután a GNOME Bonobo implementáció gondoskodik arról, hogy megjelenjen a túloldalon a widget, lehetőleg optimális méretű legyen, stb. Hasonlóan egyszerűen hozhatsz létre egy *STORAGE* objektumot egy *EFS* fájlban, vagy nyújthatsz a felhasználónak lehetőséget, hogy a valamilyen feltételnek megfelelő komponensek közül válasszon.

A *libbonobo* egyszerű kezelhetőségét talán mi sem mutatja jobban, mint hogy annak idején, amikor először ismerkedtem a Bonobo rendszerrel, a GTK+-os ismereteim alapján fél napba telt egy működő *EMBEDDABLE* létrehozása.

6. MonkeyBeans

2000 júliusában kezdtem el dolgozni a *MonkeyBeans* rendszeren. A project célja egy Java-alapú Bonobo implementáció létrehozása, ezáltal a gyakorlatban is vizsgálva a Bonobo architektúra platformfüggetlenségét.

Sajnos hamar kiderült, hogy száz százalékosan tisztán Javában nem lehetséges megoldani a feladatot: a rendszer Unix-orientáltsága miatt szükség van olyan lehetőségek kihasználására, amik a Java platformfüggetlensége miatt nem érhetők el a virtuális gépen belül (például alacsonyszintű X hívások a beágyazhatósághoz, vagy adott filedescriptorokba írás az OAF-fal történő aktiválhatósághoz).

Mindemellett a ráfordításokhoz képest nagyon gyorsan eljutott az „éppen-hogy használható” szintre: feltöltöttem egy fájl-ból egy komponenst a *PERSIST-STREAM* interfészen át, és beillesztettem egy AWT ablakba egy *CONTROL*-t, a menük összeolvasztásával együtt.

Sajnos a jövőben bizonytalan a *MonkeyBeans* sorsa: magam a sokkal hasznosabbnak (bár kevésbé úttörőnek) ígérkező *Bonobomm C++* felület fejlesztésére tértem át, és egyelőre nem látni, hogy valaki átvinné a fejlesztést. Az érdeklődők bármikor megtalálják a kódot a *cvs.gnome.org* szerver *monkeybeans* moduljában.

⁶Érdemes megemlíteni, hogy több kezdeményezés is indult már egy olyan ORBit frontend létrehozására, amely közvetlenül *GtkObject*-eket használ

Összefoglalás

A GNOME eredeti célkitűzése (mint az a nevében is megjelenik) egy objektumokból felépülő, moduláris környezet létrehozása volt. Ennek elérésében a legfontosabb eszköze a Bonobo, minden határt megszüntetve a különböző szállítók, platformok, számítógépek között.

Remélhetőleg előadásom meggyőzőtt, hogy a Bonobo egy erőteljes, de ugyanakkor egyszerűen használható megoldást nyújt a komponensalapú fejlesztéshez. Amennyiben igen, a *libbonobo* dokumentációjában megtalálhatóak azok az információk, amelyek a mindennapi használathoz szükségesek – előadásom célja nem függvénynevek felsorolása volt, hanem egy átfogó kép átadása a rendszer felépítéséről.

Hardvertokenes autentikációs eszközök (kivonat)

Györkő Zoltán

2000. szeptember 27.

Néhányan közülünk biztosan láttak már monitor szélére ragasztott sárga cetlit, melyre a monitor tulajdonosa vész esetére felfirkantotta jelszavát, nehogy véletlenül elfelejtse.

Az is biztos, hogy van, amikor ez nem a „legjobb” megoldás. Ma már kritikus feladatokat bízunk számítógépre, olyan feladatokat, ahol fontos az, hogy a felhasználó, illetve a felhasználó digitális személyisége egymástól elválaszthatatlan legyen. Nem szabad, hogy a jelszót könnyen ki lehessen találni, nem szabad a jelszót kölcsönadni, nem szabad felírni a monitor szélére, és nem szabad elfelejteni sem.

Ellentmondás? Kizárólag jelszavak alkalmazása esetén biztosan. A biztonságos jelszó minimum 8 karakter hosszú, és egyaránt tartalmaz számokat, betűket és írásjeleket. A legjobb, ha a `/dev/random`-ból olvasott bájthalmaz minden jelszó alapja, viszont ezt megjegyezni egy halandó számára nem egyszer képtelenség.

Ilyenkor megoldás a hitelesítéshez valamilyen hardvertoken alkalmazni, mely helyettesíti a jelszót, és melynek elvesztése kézzelfogható.

Előadásomban bemutatom a legfontosabb rendelkezésre álló kártyákat, kártyaolvasóval illetve anélkül használható típusokat.

PHP alapú on-line bolt, integráció a vállalat nem Linux alapú ügyviteli rendszerével

Kétszeri Csaba <ketszeri_csaba@sominfo.hu>

2000.08.22.

Kivonat

Bár egyre több cég használja a Linuxot, mint szerver operációs rendszert, ügyvitelüket döntően nem Linux alapú alkalmazásokon végzik. Egy hatékony on-line kereskedelmi rendszer működtetéséhez azonban elengedhetetlen a két rendszer bizonyos mértékű integrációja. Az előadásban egy ilyen integrációra mutatok példát.

Tartalomjegyzék

1. Elvárások az integrációval kapcsolatban	24
2. Védelmi kérdések	24
3. A meglévő ügyviteli rendszer	24
4. On-line bolt PHP-ban	24
5. Az adatcsere megvalósítása a két rendszer között	24
6. Összefoglalás	24

1. Elvárások az integrációval kapcsolatban

Az integráció során szem előtt kellett tartani néhány, az üzletmenet szempontjából fontos kritériumot. Ezek a teljesség igénye nélkül:

- A terméktörzsben alkalmazott változtatások automatikusan és idővesztés nélkül kerüljenek be az on-line bolt terméktörzsébe,
- Az on-line bolton keresztül érkezett megrendelések automatikusan kerüljenek át a rendelések közé, hogy...
- ... a készletgazdálkodási rendszer képes legyen a szállítói rendelések generálásakor az on-line érkezett megrendeléseket is figyelembe venni.

2. Védelmi kérdések

Az on-line bolt, mivel ki van téve az Interneten át rá leselkedő veszélyeknek, és itt a kritikus üzleti szerep miatt hatványozottabb a káresemény bekövetkeztékori veszteség, megfelelő védelmet kíván.

3. A meglévő ügyviteli rendszer

A cég ügyviteli rendszerét saját fejlesztőink készítették, így az integráció során magukkal a fejlesztőkkel lehetett egyeztetni a rendszerrel kapcsolatos kérdésekben.

4. On-line bolt PHP-ban

Az általam php-ben írt on-line bolt bemutatása, a fejlesztés buktatói és tanulságai.

5. Az adatcsere megvalósítása a két rendszer között

6. Összefoglalás

A bemutatott eljárások és folyamat-modellek egy a cégünk programozói által készített ügyviteli rendszer, és az általam írt on-line bolt közti, http kommunikációra alapozott adatcserét mutatják be, kiemelve azokat a buktatókat, amiken egy hasonló rendszer fejlesztése közben túl kell jutni.

Alternatíva a vállalati informatikában: a *Linux*

Kósa Attila

2000. szeptember 28.

Kivonat

Előadásom elején a PC fejlődésével párhuzamosan szeretném bemutatni a rajtuk futtatott operációs rendszerek fejlődését, hogy lássuk, milyen környezetben alakult a *Linux* olyanná, amilyen napjainkban.

A második rész a rendszer jelenlegi helyzetéről szól.

A harmadik részben pedig példákat fogok bemutatni, hogy milyen területeken használják ma Magyarországon a vállalati informatikában.

Tartalomjegyzék

1. A PC-s korszak kezdete	26
2. A közelmúlt és a jelen	27
3. Cégek bemutatása	29
3.1. Bábolna Rt.	30
3.2. Budapesti Műszaki Főiskola	30
3.3. Dunaferri Távközlési Intézet	31
3.4. Fornax Rt.	31
3.5. Medicontur Kft.	31
3.6. Philos Laboratories Kft.	32
3.7. SZTAKI	32
3.8. TVNET Kft.	33
3.9. Westel Rádiótelefon Kft.	33
4. Összefoglalás	34

1. A PC-s korszak kezdete

A mai hatalmas teljesítményű PC-k korában érdekes visszatekinteni a kezdeti időszakra. A PC fejlődésével párhuzamosan szeretném bemutatni a rajtuk futtatott operációs rendszerek fejlődését, hogy lássuk, milyen környezetben alakult a *Linux* olyanná, amilyen napjainkban.

Még 20 év sem telt el azóta, hogy megszületett az első PC, 1981. augusztusában. Ekkor jelent meg az *MS-DOS 1.0* is. Az év szeptemberében a *Microsoft* megkezdte a később *Microsoft Windows* néven megjelenő program fejlesztését. Megjelenéséhez több, mint 4 évre volt szükség, az *Intel* 80386-os processzora után egy hónappal, '85. novemberében adták ki. A feljegyzések szerint meglehetősen használhatatlan darab volt, holott a 386-os processzor már lehetővé tette volna a valódi többfeladatos működést, a multitaskot.

1987. januárjában a *Commodore* cég bemutatta az *Amiga 500*-at. Áprilisban az *IBM* megjelent egy nem PC-kompatibilis 386-alapú géppel, és a *Microsoft*-tal közösen bejelentették az *OS/2*-t, amely csak decemberben került a piacra. Még áprilisban a *Microsoft* is bejelentette az *MS-DOS 3.3*-at és a *Windows 2.0*-t. Ez utóbbi már júniusban piacra is került.

1988. októberében megjelent az *OS/2 1.1*-es verziója. Ugyanekkor a *VMS* egyik volt fejlesztője – David Cutler – a *Microsoft*-nál belefogott az *NT* tervezésébe. Egy hónappal később – novemberben – kiadták az *MS-DOS 4.01*-et, amely már a 32 MB-nál nagyobb merevlemezekkel is elboldogult.

A hardverek fejlődése is haladt közben, '89. tavaszán az *Intel* bemutatta a 80486-os processzort, amely 1 mikronos technológiával készült.

1989. májusában megjelent a *Windows 3.0*, amely már tudta használni a 640 Kb feletti memóriaterületet is.

Ez a helyzet akkor, amikor 1991. nyarán egy *Linus Torvalds* nevű finn egyetemista elkezd ismerkedni a 386-os processzor védett módú és feladatváltó lehetőségeivel. Nincs megelégedve a használható programokkal, operációs rendszerekkel – ő egyébként Minixet használ, amely egy oktatási célokra írt *UNIX*-klón –, és elhatározza, hogy ezeknél ő jobbat készít. '91. júliusában már a *POSIX* szabvány iránt érdeklődik e-mailben, állítása szerint ekkor már futott az alaprendszer. Augusztusban kiadja a 0.01-es változatot, és áttér a C nyelvű fejlesztésre assembly-ről. Októberben az első „hivatalos” verzió megjelenik az Interneten. *Linus* ekkor határozza el, hogy bevonja a fejlesztésbe a szabad kapacitással rendelkező programozókat. Decemberre jelentősen megnő a futtatható alkalmazások száma, a képességei közé tartozik ekkor például a kód- és adatmegosztás nem kapcsolódó processzek között. '92. januárjára már képes a virtuális memória kezelésére és virtuális konzolok használatára is lehetőséget nyújt.

Márciusban az *IBM* megjelenik az *OS/2 2.0*-val. Áprilisban a *Microsoft* kiadja a *Windows 3.1*-et és a *DOS 5.0*-t.

Júniusban megjelenik a *PCI* sín, majd '93. márciusában a *Pentium* processzor. Az év májusában megjelenik az *OS/2 2.1*. Augusztusban piacra kerül a *Windows NT* első verziója, 3.1-es verziószámmal, valamint a *DOS 6.0*, később pedig a *DOS 6.22*.

1994. elején elérkezik a *Linux* az 1.0-s verzióhoz, amelyet *Linus* csak akkor volt hajlandó kiadni, amikor a *POSIX* szabvánnyal való kompatibilitás kielégítővé vált. Októberben megjelenik az *OS/2 Warp 3*, '95. augusztusában pedig kiadják a *Windows 95*-öt, amely még mindig nem tudott megszabadulni a *DOS* korlátaitól.

1995. novemberében az *Intel* bemutatja a *Pentium Pro*-t, majd '96. márciusában bejelenti az *MMX* technológiát. A ráépülő processzorok azonban csak '97. januárjában jelennek meg. Abban az évben tűnnek fel a piacon a *Pentium II*-es processzorok is.

Közben – '96. elején – a *Linux* rendszermagjának számozása elérkezik a 2.0.0-hoz. Ez már lehetővé teszi a modulok használatát. A kernel bizonyos részei ettől kezdve modulként is elkészíthetők, és akár automatikusan, akár kézzel betölthetők a memóriába, ahonnan a rendszer eltávolítja őket, ha egy bizonyos ideig nem használjuk. Ezáltal a rendszermag memóriaigénye kisebb lett, hatékonysága és megbízhatósága megnőtt. Ősszel a *Novell* kiadja a *NetWare 4.11*-et.

2. A közelmúlt és a jelen

'98-ban a *Microsoft* kiadja a *Windows 98*-at, majd 2000. februárjában a *Windows 2000*-t. A két rendszer megjelenése közötti évben – 1999-ben –, sok nagy cégnek akadt meg a szeme a *Linux*-on. Ekkorra már körülbelül 7,5 millióan használnak *Linux*-ot a világon, de a cégek csak most látnak benne fantáziát. Az *Intel*, a *Netscape*, az *Oracle*, a *Compaq*, a *Novell*, az *IBM* és a *Hewlett-Packard* – más kisebb nevű cégek mellett – pénzt fektet (többek között) a *Red Hat Linux*-ot árusító cégbe. Egyre több cég szervereit, munkaállomásait és noteszgépeit lehet előre telepített *Linux*-szal is megvásárolni. Ezek közé tartozik például a *Dell*, a *Compaq*, a *Hewlett-Packard*, a *Siemens*, a *Silicon Graphics*, az *IBM* és a *Toshiba*.

Az *IBM* az 1999. márciusában megrendezett *LinuxWorld Expo*-n demonstrálta, hogy egy 17 *Netfinity* szerverből álló *Beowulf*-fürt 36 P-II Xeon processzorral ugyanarra a teljesítményre képes és ugyanúgy skálázható, mint egy *Cray* szuper-számítógép. Az árkülönbség pedig nem elhanyagolható: a *Netfinity/Linux* fürt mintegy 150.000 dollárba került, míg egy ugyanekkora teljesítményű *Cray* 5,5 millió dollár.

Az *IBM* az első – de már nem az egyetlen – cég, amely hardverrel, szoftverrel és támogatással teljes megoldásokat szállít *Linux*-ra. Átültették a *Linux*-ot majdnem minden géptípusukra az *RS-6000*-től, az *AS-400*-on át az *S/390*-es mainframe-ükig.

Nem csak a hardvergyártók ismerték fel a *Linux* előnyeit, hanem a szoftvergyártók is. A nagy – és kis – cégek sorra jelentik be, hogy portolják különböző szoftvereiket *Linux* alá. Ezek között vannak:

fejlesztői rendszerek *Cygnus, IBM, Borland, Inprise, Informix, Magic Software Enterprises, Oracle, Compaq, PlugSys International, SGI;*

adatbáziskezelők *IBM, Sybase, Oracle, Informix, Progress, Pervasive Software Inc.;*

üzleti szoftverek *SAP, IBM, Pervasive Software Inc.; McAfee, Gentia Software; Computer Associates International, Inc.; Lotus; Check Point Software Technologies Inc.; Progressive Systems Inc. Data Fellows Corporation; Macro-media Inc.;*

Végigkövethettük, ahogy egy hobbiprogramból 8-9 év alatt világméretű mozgalommá vált a *Linux*, és ez talán egyedülálló a számítástechnika történetében. Mindez úgy történt, hogy lényegében ingyen juthat mindenki hozzá, így még lehetetlenebb a siker.

Megvizsgálva a szabad szoftverek és gyártóik történetét, ez az egész mégis érthetővé válik. Több évtizede csiszolódtak a megszállott programozók szoftverei, ötletei, amire a *Linux* elkezdődött. Ezekbe a programokba mindenki azt adta bele, amihez a legjobban értett, nem pedig valami kívülről ráerőltetett feladaton dolgozott. A programok tesztelése is igen széleskörű volt. Ezenkívül, a freeware-programok nem üzleti célból készültek, így alkotóik nem a maximális anyagi hasznot keresték, hanem maguk számára akartak használható rendszert összehozni, és villogni akartak a többiek előtt tudásukkal. Ez a légkör sokkal jobban kedvezett a hatékony, stabil programok kialakításának, mint a szoftverbirodalmak pénzorientált rendszere.

A *Linux* történetébe való bepillantás tehát egy olyan világba vezet minket, ahol a programozók – úgymond – „dicsőségért” programoznak, mindenki szabadon átadja ismereteit a többieknek, és mégis a szoftvernagyhatalmakkal összevethető eredményességgel dolgoznak.

Ez a mentalitás nemcsak a *Linux*-ot jellemzi, hisz már a '70-es években jelentkeztek a *UNIX*-os világban az első szabadterjesztésű programok (maga a *UNIX* is az volt eredetileg). Ezeket a szabad szoftvereket a *GNU project*, valamint a *Free Software Foundation* fogja össze, melyeknek tevékenysége szélesebb körű, mint a *Linux* rendszer.

Ma a *Linux* egy 32 bites, *POSIX* szabványt követő *UNIX* változat, amely eredetileg csak *IBM PC* gépeken futott (80386 vagy jobb processzor esetén), de mára nagyon sok hardverre adaptálták. Így létezik *Linux DEC AXP, PowerPC, M680x0, Sun Sparc* alapú gépekre is. A 32 bites változat mellett létezik 64 bites és 8 bites

Linux is. A 64 bites *Sun* gépekre készült, a 8 bites pedig az 8086, 8088, 80186 és 80286-os processzorra. 1999. augusztusában Craig Barrett – az *Intel* elnöke-vezérigazgatója – bemutatta egy Merced-alapú rendszeren az *IA-64 Linux* prototípusát, demonstrálva ezzel, hogy lesz *Linux* a 64 bites PC-kre is. A *Dell* a 2000. augusztusában megrendezésre kerülő *LinuxWorld* konferencián és kiállításon mutatja be az *Intel* 64 bites *Itanium* processzorára épülő *PowerEdge* kiszolgálójának prototípusát. *Red Hat Linux* fut rajta, és el lehet érni róla a mySAP.com elektronikus üzleti alkalmazásait és az *IBM DB2* adatbáziskezelőjét.

A rendszer kidolgozottsága olyan fokú, hogy egyre több helyen alkalmazzák *UNIX*-os munkaállomásként, vagy hálózati szerverként. Mindkét esetben hatalmas előny a szokásos *IBM PC*-s programokkal szemben a nagyfokú megbízhatóság és az alacsony ár, valamint az sem elhanyagolható, hogy nagyon nagy a hasonlóság a *Linux* és a „nagygépek” operációs rendszerei közt, azaz pl. egy linuxos program könnyen átvihető mondjuk egy *Sun SPARC* gépre, de gondos programozás esetén akár egy *CRAY* szupergépre is.

A médiában is egyre gyakrabban merül fel a *Linux* név. Szinte minden nap jelennek meg hírek, amelyek dicsérik, és arról tájékoztatnak, hogy milyen nagy cégek csatlakoztak a „*Linux* mozgalomhoz”. De arról nem olvashatunk (hallhatunk), hogy tulajdonképpen mire is használják ezt az operációs rendszert. Sokáig tartotta (talán még most is tartja) magát a nézet, hogy otthoni játékszernek kitűnő, de ipari környezetben, igazi munkára nem alkalmas. Ennek az is lehet az oka, hogy a cégek büszkén hirdetik, ha valamilyen „pénzes” operációs rendszert használnak, de nem dicsekszenek azzal, ha valamilyen nyílt forráskódú (ingyenes) rendszert használnak. Reméljük, hogy az előadásban szereplő cégek „merészsége” segít feloldani a többi cég féltékenységét ezen a téren.

Ezt a tartózkodást külföldön már leküzdötték, nem tartják „szégyennek”, ha *Linuxot* használnak. Egészen nagy cégek is önként – büszkén – bevallják, hogy ahhoz az egyre népesebb táborhoz tartoznak, amely *Linuxot* használ. Például: NASA, Boeing Company, Cisco Systems Inc., Corel Computer Corp., Mercedes-Benz AG, Sony Electronics Inc., Editions O'Reilly, United States Postal Service, Netscape Communication Corp., United States Army Publishing Agency.

3. Cégek bemutatása

Most nézzük meg, hogy Magyarországon mennyire elterjedt a *Linux* használata, az egyes cégek milyen feladatokat bízhatnak rá.

A felsorolásban szerepelnek kisebb-nagyobb cégek, sőt még a felsőoktatásból is válogattam. A döntésben, miszerint az említett cégek *Linuxot* használnak más rendszer helyett, elsősorban a rendszer stabilitása, megbízhatósága játszott főszerepet, az ingyenessége csak „hab volt a tortán”. A közzétett adatokat minden cég

önként adta meg, semmilyen ellenszolgáltatást nem kértek érte.

3.1. Bábolna Rt.

A *Linux* futtatására használt gépek skálája meglehetősen széles: 486DX2-66 MHz-től Pentium III 550 MHz-ig terjed. Az Intranet-szerver egy Dell PowerEdge 4200, két Pentium II 266 MHz-es processzorral, 128 MB RAM-mal, és 6*4 GB merevlemez RAID5-be kötve (16 GB háttértár). A feladatok is nagyon sokrétűek: proxy-szerver – squid; belső DNS szolgáltatás – bind; web-szerver (Apache) és adatbáziskezelés (PostgreSQL) – erre alapozva már aktív html eszközökkel oldották meg a nemzetközi gazdanapok informatikai rendszerét (ez még mind a 486DX2-66 MHz-es gépen!); Novell emulátor – Mars_NWE; fájlserver – SaMBA; levelező-szerver – qmail; hálózاتفelügyelet – Tklnd, mrtg, NetSaint (kiterjedt hálózat Békéscsabától Szentgotthárdig); betárcsázó-szerver – a telephelyek és a központ közötti adatszolgáltatást bonyolítja; ftp-szerver; korábban ISDN kapcsolat – a bérelt vonal kiépítésének idejére; tűzfal; x-terminál UNIX rendszerekhez; irc – problémák jelzésére a leghatékonyabb megoldás; webfórum; DHCP szolgáltatás. AIX rendszerek egyes könyvtárainak elérése NFS-en keresztül, és továbbajánlása SaMBA segítségével a Windowsos klienseknek. Az AIX-en futó Oracle alkalmazás indítása egy intranetes aktív weboldalon keresztül történik, a szükséges Java appletek a nagyobb távoli telephelyek esetében nem a központban található gépekről, hanem a helyi webszerverekről töltődnek le. Az AIX-en futó Oracle adatbázis menedzselése ObjectManager-rel. Webes felületen keresztüli levelezés a felhasználóknak – sqwebmail-lel megvalósítva. IBM mainframe elérése x3270-es terminálemulációval.

A felhasználók száma: ~500 fő számára web-elérés, ugyanannyi postafiók biztosítása, ~300 fő számára Oracle beléptetés.

A Debian disztribúciót preferálják.

3.2. Budapesti Műszaki Főiskola

2 darab 366 MHz-es Celeron processzor, 512 MB RAM és kb. 100 GB SCSI merevlemez található abban a gépben, amely az ftp://ftp.fsn.hu/ címen lévő ftp-szervert futtatja. A megvalósításhoz a proftpd nevű szoftvert használják. A gépet rsync szolgáltatással is el lehet érni, valamint fut rajta egy webszerver is – webfsd. Nagy adatforgalmat bonyolít le a gép, napi 150-230 GB, ez havonta kb. 5 TB! Régebben, amikor csak 160 MB RAM volt a gépben akkor is előfordult, hogy 350 felhasználó egyszerre használta a gépet. Ilyenkor kb. 120 MB swap-pet használt aktívan, és a 100-150-es load-ot is elérte a terhelés. A memóriabővítés óta a load-ot sikerült 2 környékén tartani.

A Debian disztribúciót preferálják.

3.3. Dunaferri Távközlési Intézet

Egy 200 MHz-es Pentium MMX, 32 MB RAM, 1 darab 1,2 GB-os és 1 darab 2,1 GB-os IDE merevlemez biztosítja a helyet az operációs rendszernek és az alkalmazásoknak. A hálózati kapcsolatot egy 3c509-es hálózati kártya teremti meg. Erre a gépre a következő feladatokat bízta rá: intranet telefonkönyv, egy UNIX-alapú telefonközpont számlaarchiválása (ftp-vel mirrorozás) és ftp-csere. A telefonkönyv Apache és PHP3 segítségével lett megoldva, az ftp-szerveri feladatot a proftpd csomag látja el.

Egy *Suse Linux* kezeli az ALCATEL telefonközpont hangpostáját.

A Debian disztribúciót preferálják.

3.4. Fornax Rt.

A <http://www.fornax-monitor.hu/> címen lévő szervert 2 darab 550 MHz-es Pentium III-as processzor hajtja, a rendszernek és az adatoknak 2 darab 8 GB-os Ultra66-os merevlemez ad helyet. A másik webszerver egy Sun4U, u1 147 MHz-es processzorral, 128 MB RAM-mal és 2 darab, 4 GB-os SCSI2-es merevlemezrel felszerelve. A szerverek forgalma: napi átlagban 7.500 látogató, ez havonta kb. 200.000 érdeklődőt jelent. Ez adatforgalomban havi 60 GB-ot, találati számban (hit) kb. 220.000-t jelent. A gépek nagyon jól skálázhatók, terhelésük csúcsidőben 70-75%. A hardvermeghibásodás miatti kiesés elhanyagolható, évi 1-2 esetben fordul elő.

Oracle adatbáziszervert is használnak a cégnél. Régebben az iBCS2 emuláció segítségével futtatták a 7.1.3-as verziót, ma viszont a 8.1.5-ös Linuxos változatot alkalmazzák. A használt adatbázis mérete nagyjából 1,2 GB, tőzsdei adatokat tartalmaz.

Az adatok mentését is *Linuxon* oldották meg a multiplatformos rendszerben. Az Amanda nevű backup rendszer egy HP DAT24-esre, DDS2 kazettákra menti automatikusan 6-7 gép anyagát, amelyek között van Sun Solaris, Windows NT, *Linux*, sőt régebben SCO Unix is.

A Debian disztribúciót preferálják.

3.5. Medicontur Kft.

486DX4-120 MHz-es processzor, 16 MB RAM, 1 GB merevlemez az „otthona” annak a *Linuxnak*, amelyet levelezésre, és Internet gatewayként használnak egy ISDN vonalon keresztül. Az erre az „aprócska” gépre bízott feladatok:

- belső web-szerver – Roxen Challenger
- SQL-szerver – MySQL

- LDAP-szerver – OpenLDAP
- ftp-szerver – proftpd
- tűzfal – ipfwadm
- fájlserver – SaMBa
- levelező-szerver – sendmail (amely az Amavis víruskeresőt használva biztosítja a beérkező levelek vírusmentesítését); és még a telefonbeszélgetések időtartamát is rögzíti az ISDN vonalon keresztül. Körülbelül 15 felhasználót szolgál ki.

3.6. Philos Laboratories Kft.

166 MHz-es Pentiumtól kezdődően, 600 MHz-es Athlonig terjed a *Linuxot* futtató számítógépek skálája a cégnél. A rábízott feladatok: fájlserver – SaMBa, nfs; webszerver – Apache; levelező-szerver – sendmail; ftp-szerver – wu-ftp. A *Linuxos* gépek és a 40 felhasználó adminisztrálása NIS segítségével történik. A cég profilja miatt – játékszoftver-fejlesztés – használnak még fordítószervert, és a GNATS hibakövető rendszert (bug tracking system).

A Debian disztribúciót preferálják.

3.7. SZTAKI

2000. március 6-án átadták Magyarország legnagyobb teljesítményű számítógépét. A SZTAKI-ban 28 PC-t kapcsoltak össze 100 megabites gyorsaságú hálózattal, így a szuperszámítógépek árának töredékéért közel 30 ezer megaflopsra tudták emelni a gépek összteljesítményét. A klaszter jellemzői: teljes memóriakapacitás: 3,84 GB; teljes merevlemez-kapacitás: 290 GB; hálózati áteresztőképesség: 34 Gbps; csúcssebesség: ~30 Gflop. A gépek műszaki jellemzői (amelyekből a klaszter áll): DELL Precision 410M munkaállomás, 2 db Intel Pentium III 500 MHz-es processzor, 128 MB ECC SDRAM, 9,1 GB Ultra2 SCSI merevlemez, 100 Mbit-es Ethernet hálózati kártya, 3D gyorsító videokártya, 32 MB RAM-mal, 40-szeres sebességű SCSI CD-ROM olvasó, 15" DELL monitor. A hálózat 100 Mbit-es Ethernet, 48 portos Cisco 100 Mbit-es Ethernet kapcsolóval (full duplex, 24 Gbps). Többféleképpen lehet elérni a klasztert: sok felhasználó számára garانتál legalább egy munkaállomást; sok munkaállomást biztosít néhány felhasználó számára.

A SZTAKI eddig a Paksi Atomerőmű Rt.-vel és az Országos Meteorológiai Szolgálatnál tárgyalt már az együttműködés lehetőségeiről, de várják a szupergyors program felhasználása iránt érdeklődő nagyobb vállalatok érdeklődését is.

Alkalmazási területeik: univerzum vizsgálata, atomerőmű-blokkok működésének modellezése, meteorológiai előrejelzések, szemcsés anyagok keverése és szétválasztása, kémiai technológiai alkalmazások, anyagtani vizsgálatok, környezetvédelem.

A klaszter operációs rendszere *Red Hat Linux 6.1*.

3.8. TVNET Kft.

Compaq Proliant szerver, 733 MHz-es Pentium III-as processzorral, 2 darab 18 GB-os SCSI merevlemez RAID vezérlővel és 256 MB RAM – ezen a hardveren üzemel a <http://www.tvnet.hu/> címen elérhető webszerver. Tervbe van véve egy új, erősebb gép beszerzése: 833 MHz-es Pentium III processzorral és 6 darab 9 GB-os merevlemezzel, amelyeket RAID5-be kötve fognak használni. Ez lesz az új intranet-szerver, és ezen fog futni a levelező- és az adatbáziskezelő-szerver is.

A jelenlegi intranet-szerveren (333 MHz-es Celeron, 192 MB RAM-mal) egyszerre fut a Sybase SQLanywhere és a PostgreSQL adatbáziskezelő-szerver. Az egyik a számlázásért felelős, a másik a szolgáltatás iránt érdeklődőket tartja nyilván. A Sybase-hez csak windowsos kliensek vannak, a PostgreSQL-t az Apache és PHP3 párosításon keresztül érik el a kliensek. (A Sybase-hez is lehetséges PHP3-as felületet készíteni, mert létezik hozzá odbclib.) Ez a szerver fájlserverként is elérhető kétféle módon: a SaMBa csomag és nfs segítségével – kb. 30 felhasználót szolgál ki eképpen. Ez a gép bonyolítja a levelezést is.

A hálózatmenedzsmentben is szerepet kapott a *Linux*, snmp felhasználásával.

Kiseb feladatokra is *Linuxot* alkalmaznak, például irc, news-, ftp- és DNS-szerver.

A RedHat disztribúciót preferálják.

A Sybase-es klienseken, a menedzserek és marketingesek gépein kívül nincs Microsoft alapú szoftver az egész rendszerben.

3.9. Westel Rádiótelefon Kft.

Fujitsu Siemens, Digital Compaq gépeket használnak *Linux* futtatására – teljes internetszolgáltatást nyújtanak kb. 1000 felhasználónak.

A megoldásra használt szoftverek:

- levelezés – exim, endmail, imapd, imp
- webszerver – Apache
- ftp-szerver – proftpd

- proxy-szerver – squid
- adatbáziskezelés – PostgreSQL
- ssh – open-SSH

Saját fejlesztésű szkripteket használnak a szolgáltatások és szoftverek futásának ellenőrzésére, hiba esetén ezek figyelmeztető jelet képesek küldeni a meghatározott szakembereknek – sms-ben.

A 0660 sms rendszerét is *Linuxon* valósították meg.

Terveik között szerepel ISDN behívó router összeállítása.

A Debian disztribúciót preferálják.

4. Összefoglalás

Egy informatikai rendszer komponensekből épül fel, melyeknek együtt kell működniük egymással. A kompatibilitás megvalósítása nagyon nehéz, szinte lehetetlen feladat – különösen több gyártó esetén. Kivéve, ha a megoldás nyílt szabványokon alapul, és könnyű az átalakítása, testreszabása. Az látható a válogatásból, hogy leggyakrabban az olyan feladatok kerülnek át *Linuxra*, amelyek nyílt szabványokon alapulnak, mivel maga a *Linux* is nyílt szabványokat használ.

A példák azt mutatják, hogy az egymásra kölcsönösen támaszkodó kommersziális és nyílt forráskódú szoftverek révén jól működő üzleti informatikát lehet kialakítani.

Hosszan lehetne sorolni még a helyeket, ahol *Linuxot* használnak. Hozhattam volna példákat az egészségügyből, vagy ipari folyamatirányító rendszerekről is. Talán majd a következő előadáson.

Hozzáférésvezérlési modellek Linuxon

Magosányi Árpád <mag@lme.linux.hu>

2000.09.23.

Kivonat

A unix rendszereken hagyományosan a felhasználó-csoport-bárki más alapú hozzáférésvezérlési modellt implementálják. Ez a modell egy bonyolultabb igényeket támaztó rendszeren gyakran nem elégséges. Mint a legtöbb komoly Unix változaton, Linuxon is lehetséges ennél komolyabb hozzáférésvezérlést megvalósítani. Az előadás bemutatja az ismertebb hozzáférés-védelmi modelleket, és az azokat megvalósító szoftvermegoldásokat. Az előadás kitér a hálózati hozzáférésvezérlés aspektusaira is, és bemutat egy hálózati hozzáférésvezérlési modellt, amely a Bell-LaPadula modell szellemét követi, és a covert channelek problémáját próbálja meg kezelni.

Tartalomjegyzék

1. Hozzáférésvezérlés	37
1.1. Diszkracionális hozzáférési modellek	37
2. Kötelező hozzáférésvezérlési modellek	40
2.1. Egy egyszerű modell	40
2.2. Bell-LaPadula modell	41
2.3. Bell-LaPadula modell dinamikus címkékkel	42
2.4. Biba integritási modellje	43
2.5. Low Water Mark modell	43
2.6. Kínai fal modell	43
2.7. Clark-Wilson modell	44
2.8. Privacy Modell	44
2.9. Háló modellek	45

3. MAC implementációk Linuxon	46
3.1. RSBAC	47
3.2. Medusa	47
3.3. LOMAC	48
3.4. MAC	48
3.5. Egyéb megoldások	48
4. Hálózati alkalmazás	49
5. Összefoglalás	51

1. Hozzáférésvezérlés

Mint azt nagy gondolkodóink már többször megmondták, „az élet célja a küzdés maga”[1]. Persze azon hosszasan lehet vitázni hogy ennek a küzdelemnek mi a megnyilvánulási formája. Darwin szerint például a küzdelem az erőforrások megszerzéséért folyik. Ebből rögtön következik, hogy az erőforrásokat védeni kell. A három legfontosabb védendő tulajdonsága az erőforrásoknak pedig azok rendelkezésre állása, integritása és az a tény, hogy az erőforrás a *mi* rendelkezésünkre áll. Az utóbbit nevezzük az egyszerűség kedvéért bizalmasságnak.

Az egyik ilyen erőforrás az információ. Ezzel el is jutottunk az informatikai biztonság területére. Az adatok és egyéb informatikai erőforrások védelme pedig az azokhoz való hozzáférés szabályozásán alapul. Régen amikor a férfiak még igazi férfiak voltak, nem pedig hátulgombolós Pascal programozók[2], úgy tűnt hogy kiváló módszer a hozzáférésvezérlésre az hogy az erőforrást jelképező objektumokat (amiket a köznyelv a „fájl” szóval illet, de mi mindannyian tudjuk hogy a file-okról van szó), ellátják olyan attribútumokkal, amikből kiderül az hogy ki férhet hozzá milyen művelet céljából az egyes fájlokhoz. Persze valakinek képesnek kell lennie arra a műveletre hogy ezeket a jogosultságokat beállítsa. Ilyenkor az erőforrás tulajdonosára van bízva, hogy kinek milyen jogosultsága van az erőforráshoz. Az ilyen módszereket nevezzük diszkrecionális hozzáférési modelleknek.

1.1. Diszkrecionális hozzáférési modellek

A diszkrecionális hozzáférési modellekben általában felhasználók és felhasználói csoportok alapján osztanak ki elég sok féle jogot. Az erőforrás tulajdonosa leggyakrabban a felhasználó, de az sem ritka hogy a jogosultságokat az arra hivatott adminisztrátor osztathatja ki. Lássuk, milyen diszkrecionális hozzáférési modellek léteznek Linuxon, mire jók, és milyen problémákra nem adnak megoldást:

Szabványos unix hozzáférésvezérlés

A unix[3] rendszerek -így a Linux is- a hozzáférésvezérlésre a felhasználó-csoport-bárkimás hármast használják. Egy állományhoz egy felhasználó tartozik, aki egyúttal annak tulajdonosa is, és egy csoport. A műveletek aránylag kevesen vannak, név szerint írás, olvasás, végrehajtás. Ezt a modellt és működését mindannyian ismerjük, kár is több szót fecsérelni rá.

Hozzáférési listák

Másik nagyon elterjedt diszkrecionális hozzáférésvezérlési módszer a hozzáférési listák, külföldiül ACL-ek (Access Control List) használata[4]. Az ACL-ek is a felhasználó és felhasználói csoport alapján osztják ki a jogosultságot, de egy állományhoz több szabály is tartozhat, és általában nem csak olyan szabály van ami megad egy hozzáférést, hanem olyan is ami azt megvonja. Az ACL típusú rendszereken gyakran öröklődő szabályok is vannak, amikor egy könyvtár hozzáférési jogai öröklődnek az abban lévő file-okra is. Általában az írás, olvasás, végrehajtás műveleteken kívül további műveleteket is definiálnak.

A következő Linuxos implementációk léteznek állomány szintű diszkrecionális hozzáférésvezérlésre:

- Posix ACL project[5], amelyik ext2 állományrendszeren valósít meg ACL-eket.
- A Linuxon használható állományrendszerek közül az XFS[6] beépítve tartalmazza az ACL támogatást.
- Ha olyan ACL támogatást keresünk, ami független a használt állományrendszertől, a trustees[7] nevű implementáció lehet érdekes. Ez az implementáció arra is példa, amikor nem a felhasználó hanem a rendszeradminisztrátor osztja ki a jogosultságokat.
- Az RSBAC[8] nevű általános biztonsági csomag is tartalmaz állományrendszer-független ACL támogatást, de majd látni fogjuk hogy főként a kötelező hozzáférésvezérlési modellek támogatása a szakterülete.

Hálózati hozzáférésvezérlés

Amikor hálózati hozzáférésvezérlésről beszélünk, a felhasználót nem csak felhasználónév vagy csoport, hanem az általa használt gép IP címe alapján azonosítjuk. A hálózati hozzáférésvezérléssel kapcsolatban a gépünk játszhatja a kiszolgáló vagy a forgalomszűrő szerepet.

A valamilyen állományrendszert kiszolgáló alkalmazások, mint pl web- és ftp szerverek, nfs és samba szerverek általában saját hozzáférésvezérléssel rendelkeznek, hiszen igazán szabványos felhasználóazonosítási protokoll nem létezik, és ezek a kiszolgálók felhasználó alapján (is) döntenek el hogy egy file-hoz a hozzáférést megadják-e.

A szabványosnak tekinthető hálózati hozzáférésvezérlési módszer unixokon a tcpwrapper[9] használata, amely a hosts.allow és hosts.deny állomány alapján dönti el hogy egy adott IP cím a szolgáltatáshoz hozzáférhet-e.

Alacsonyabb szinten végzett hozzáférésvezérlés a csomagszűrés is. Csomagszűrést szokás a kiszolgálón és a forgalomszűrésre használt gépen is használni.

A Linux kernel már régóta tartalmaz csomagszűrő támogatást. De elég gyakran változik hogy pontosan milyen. A 2.0-ás sorozatú kernelek az ipfwadm, a 2.2-esek az ipchains, és a 2.4-esek a netfilter[10] nevű csomagszűrőt támogatják.

Persze a csomagszűrés mint hozzáférésvezérlési módszer igencsak szegényes; mivel alacsony szinten dolgozik, nagyon kevés információja van, és nagyon kevés műveletet ismer. A rendes hálózati hozzáférésvezérlésre a forgalomszűrő esetben tűzfalat szoktak használni. Linuxon több kereskedelmi tűzfalszoftver is fut, nagyon sok egy-egy protokollt ismerő proxyt írtak hozzá, és van néhány nyílt forrású tűzfalszoftver is. Ezek a következők:

FWTK Az FWTK[11] ugyan nem nyílt forrású, de ingyenesen letölthető szoftver. Noha kezd elavulni, a hálózati határvédelem történetében rendkívül fontos szerepe van, és ma is jól használható applikációs szintű tűzfal.

socks A socks tűzfalak a tűzfalproxyk egy másik ága mint az applikációs tűzfalak. A socks egy „áramkör szintű” tűzfal. Linuxra több implementációja létezik, a legismertebb a Dante[12].

T.REX A T.REX[13] egy kb 1 éves tűzfalszoftver. Ez egyrészt már meglévő proxy szoftverekből, másrészt a gyártó által készített proxykból áll. Applikációs szintű tűzfal.

Zorp A Zorp[14] egy új generációs, moduláris, valódi applikációs szintű hozzáférésvezérléssel rendelkező tűzfal, amely -mint azt látni fogjuk- kötelező hozzáférésvezérlési modelleket is támogat.

A diszkrecionális hozzáférésvezérlési módszerek hátulütője

Mint mondtam vala, régen úgy gondolták, hogy a diszkrecionális hozzáférésvezérlés mindenre elég. De sajnos a trójaiak, a rosszindulatú felhasználók és a rosszul megírt applikációk ellen nem véd. Példaképpen vegyük azt az esetet[15], amikor András elkészít egy titkos dokumentumot. A dokumentum annyira titkos, hogy csak Béla láthatja, de Cecília, a titkárnő már nem. András beállítja a dokumentum hozzáférési jogait úgy hogy csak Béla láthassa. De Béla nyúl, és a dokumentumról készít egy másolatot Cecília részére, hogy helyette ő végezze el a dokumentummal végzendőket. Igen ám, de Cecília tulajdonképpen a versenytárs cég felbérelt ügynöke, és huss, elküldi a dokumentumot az APEHnek.

Ilyen és hasonló esetekre találták ki a kötelező hozzáférésvezérlési modelleket.

2. Kötelező hozzáférésvezérlési modellek

A kötelező hozzáférésvezérlés (MAC, Mandatory Access Control) olyan szabályokat definiál, amiket az informatikai rendszer minden elemének és felhasználójának be kell tartania. Ezek a modellek azon alapulnak, hogy az adatoknak és a felhasználóknak címkéi vannak, amik megmondják hogy az adat mennyire titkosított, illetve milyen jellegű, és a felhasználó mennyire titkos és milyen jellegű adatokhoz férhet hozzá.

Kötelező hozzáférésvezérlési modell több féle van, attól függően hogy az adatoknak melyik tulajdonságát (bizalmasság, vagy integritás) kell jobban védeni, illetve hogy milyen lehetőségeket nyújt a használt informatikai rendszer.

Egy MAC modellt matematikusan egy hármassal lehet leírni[15]:

$$\langle SC, \Rightarrow, \oplus \rangle$$

ahol:

- SC a biztonsági osztályok (lehetséges címkék) halmaza
- $\Rightarrow \subseteq SC \times SC$ egy „folyhat” reláció az SC halmazon
- $\oplus : SC \times SC \Rightarrow SC$ egy „osztálykombinációs” operátor az SC halmazon.

A dolog nagy lényege az, hogy a \Rightarrow reláció megmondja azt hogy milyen irányba folyhat az információ, és a \oplus operátor megmondja hogy ha két adatot kombinálunk, azoknak mi lesz a besorolása.

2.1. Egy egyszerű modell

Most a példa kedvéért képzeljünk el egy Linuxos rendszert, amit úgy építettünk fel, hogy a base rendszer installálása után a /titkos és a /nemtikos könyvtárakba ismét kicsomagoltunk egy-egy base rendszert, majd a /etc/inittab-ot átírtuk úgy, hogy a getty-eket tartalmazó sorok így nézzenek ki:

```
1:2345:respawn:/usr/sbin/chroot /titkos /sbin/getty 38400 tty1
2:23:respawn:/usr/sbin/chroot /titkos /sbin/getty 38400 tty2
3:23:respawn:/usr/sbin/chroot /titkos /sbin/getty 38400 tty3
4:23:respawn:/usr/sbin/chroot /nemtikos /sbin/getty 38400 tty4
5:23:respawn:/usr/sbin/chroot /nemtikos /sbin/getty 38400 tty5
6:23:respawn:/usr/sbin/chroot /nemtikos /sbin/getty 38400 tty6
```

Az így kialakított környezeteket „chroot-olt környezet”-nek[16] vagy „sandbox”nak nevezzük. Most képzeljük el, hogy a root sandboxból csak a két sandboxba bechrootolva indítjuk el az ott lévő rc scripteket, valamint azt hogy maximum az egyik külső sandboxban fut bármilyen hálózati szolgáltatás, a root sandboxban pedig (az ineten kívül) semmi. Valamint álmodjuk azt, hogy a chrootolt környezetből nem lehet kitörni[17].

Matematikusul a fenti helyzetet így mondjuk:

- $SC = \text{root, titkos, nemtitkos,}$
- $\Rightarrow = \{(\text{root}, \text{root}), (\text{titkos}, \text{titkos}), (\text{nemtitkos}, \text{nemtitkos})\},$
- $a \oplus b = \{a, \text{ ha } a = b \text{ egyébként nem definiált } \}$

Ezzel implementáltunk is egy MAC modellt, jaj de jó nekünk.

Persze ennél a modellnél léteznek jóval használhatóbbak is, lássunk néhányat:

2.2. Bell-LaPadula modell

A legismertebb MAC modell a Bell-LaPadula[18] modell, ami a következő módon néz ki: A modell objektumokat és szubjektumokat definiál; egy objektum egy file vagy bármilyen más passzív dolog a rendszerben. A szubjektum pedig a felhasználó, és az ő nevében futó processz, job. $\lambda(o) \in SC$ jelöli az „o” objektum biztonsági osztályát, vagy röviden címkéjét. A címkék között egy $\leq_C SC \times SC$ rendezési reláció van értelmezve. A felhasználókhhoz rendelt címkét úgy kell értelmezni, hogy a felhasználó beléphet bármilyen biztonsági címkével, ami kisebb vagy egyenlő mint az ő címkéje.

- Egyszerű biztonsági tulajdonság: Az s szubjektum csak akkor tud az o objektumból olvasni, ha $\lambda(s) \geq \lambda(o)$.
- \star -tulajdonság: Az s szubjektum csak akkor tud az o objektumba írni, ha $\lambda(s) \leq \lambda(o)$.

Könnyen látható hogy a fenti szabályok, ha nem teszünk különbséget szubjektum és objektum között, megfelelnek a következő szabálynak: $\lambda(s_1) \Rightarrow \lambda(s_2)$ ha $\lambda(s_1) \leq \lambda(s_2)$, vagyis az információ csak „felfelé” folyhat.

A modell kiegészítései

Természetesen az olvasási és írási műveleten kívül más műveletek is léteznek. Ilyen például a létrehozás és a törlés. Egy kis képzelőerővel ezeket a műveleteket is be lehet sorolni a két alapvető művelet közé, vagy ki lehet találni nekik hasonló, az eddigiekkel konzisztens szabályokat.

A Bell-LaPadula modell csak az információ bizalmasságával törődik, az integritásával nem. Ez azt jelenti, hogy egy alacsony szinten lévő felhasználó nyugodtan írhat egy nagyon fontos file-ba, akár felülírva annak tartalmát. Ezért gyakran a \star -tulajdonságnál csak az egyenlőséget szokták megengedni. Az információ ilyenkor is csak felfelé folyik, viszont azt csak „húzni” lehet, „tolni” nem.

Másik kiterjesztése a Bell-LaPadula modellnek az, amit a TCSEC[21] B osztálya, illetve a Common Criteria[22] (CC) Labeled Security Protection Profile-ja[23] (LSPP) megkövetel, illetve a szerzők is így írják le később. Nevezzük ezt a modellt az egyszerűség kedvéért DoD modellnek: A biztonsági osztályok hierarchiába rendezett kategóriái („sensitivity label”) mellett a sorrendbe nem rakott „integrity label”-eket vezeti be amik címkék halmazai. Az integrity labelekre a rendezési operátor a \subseteq . Így a modell a következően néz ki:

- $SC = SC_s \times SC_i$ ahol \leq teljes rendezési reláció SC_s -en, és \subseteq részleges rendezési reláció SC_i -n.
- $\lambda(o) = (\lambda_s(o), \lambda_i(o))$ ahol $\lambda_s(o) \in SC_s$ és $\lambda_i(o) \subseteq SC_i$
- $s_1 \Rightarrow s_2$ definiált ha $\lambda_s(s_1) \leq \lambda_s(s_2)$ és $\lambda_i(s_1) \subseteq \lambda_i(s_2)$.
- $\lambda(o_1) \oplus \lambda(o_2) = (max(\lambda_s(o_1), \lambda_s(o_2)), \lambda_s(o_1) \cup \lambda_s(o_2))$

2.3. Bell-LaPadula modell dinamikus címkékkel

Az egyszerű Bell-LaPadula modellnél a felhasználónak belépéskor valamilyen módon közölni kellett a rendszerrel, hogy éppen melyik biztonsági szinten van, biztonsági szintjének váltásához pedig újra be kellett lépnie. Ennek elkerülésére találták ki a dinamikus címkéket: a processzekhez nem egy, hanem két címkét rendelnek: az egyik címke azt mondja meg, hogy milyen az a minimális biztonsági osztály, amibe írhat, a másik pedig azt, hogy milyen az a maximális osztály, amiből olvashat. Ha pedig valamilyen állományhoz hozzányúl, a címkehalmaza a megfelelő módon összeszűkül. Lássuk mindezt matematikusan is (noha kissé pongyolán):

- $\lambda_{obj}(o) \in (O \rightarrow SC)$
- $\lambda_{minwrite}(s) \in (S \rightarrow SC)$
- $\lambda_{maxread}(s) \in (S \rightarrow SC)$
- $\lambda_{subj} \in (S \rightarrow SC \times SC), \lambda_{subj}(s) = (\lambda_{minwrite}(s), \lambda_{maxread}(s))$
- $read \in S \times O \rightarrow SC \times SC, read(s, o) = (\lambda_{obj}(o), \lambda_{maxread}(s))$
ha:
 $\lambda_{maxread}(s) \leq \lambda_{obj}(o)$

- $write \in S \times O \rightarrow SC \times SC, write(s, o) = (\lambda_{minwrite}(s), \lambda_{obj}(o))$
 ha:
 $\lambda_{minwrite}(s) \geq \lambda_{obj}(o)$

A read és write operáció kimenete a szubjektum új címkéit reprezentálja.

2.4. Biba integritási modellje

Mint láttuk, a Bell-LaPadula modell csak a bizalmasságra helyezi a hangsúlyt. Természetesen van olyan eset, amikor az adatok bizalmassága nem is érdekes, de az fontos hogy a rendszer elemeit ne tudják illetéktelenek változtatni. A Biba-féle[19] modellnek pontosan ez a lényege. Ugyanazokat a szabályokat definiálja, mint a Bell-LaPadula modellt, csak megfordítja a \geq jelet, és λ helyett ω az objektum címkéjét meghatározó operátor.

- Egyszerű integritási tulajdonság: Az s szubjektum csak akkor tud az o objektumból olvasni, ha $\omega(s) \leq \omega(o)$.
- integritási \star -tulajdonság: Az s szubjektum csak akkor tud az o objektumba írni, ha $\omega(s) \geq \omega(o)$.

Könnyen belátható, hogy egy olyan rendszeren ahol a Bell-LaPadula modell implementálva lett, egyszerűen a címkék sorrendjének megcserélésével megkapjuk a Biba modell implemetációját.

2.5. Low Water Mark modell

Ez a modell[24] az adatok integritásának védelmére szolgál. A lényege az, hogy az objektumokhoz és szubjektumokhoz rendelt címke azok „tisztaságát” jelöli. Minél kisebb a címke, az adat annál koszosabb. Az új egyedek (objektumok és szubjektumok) a címkéjüket attól az egyedtől öröklik, amelyikből származnak. Az objektumok címkéje nem változik, a szubjektumok címkéje pedig a legkisebb címkéjű eddig olvasott objektum címkéje. Írni csak ugyanolyan vagy kisebb címkéjű objektumba lehet. Ha egy szubjektum címkéje csökken, az összes olyan objektumot, amit éppen írásra nyitva tart, „eldob”.

2.6. Kínai fal modell

A kínai fal modell[26] arra a helyzetre keresi a megoldást, amikor pl egy konzultáns cég munkatársai több egymással versengő cégnek is végeznek munkát. Ilyenkor

nem szabad hogy ugyanaz a munkatárs több, ugyanabban a szektorban tevékenykedő cég számára is dolgozzon. Ezt a kihívást COI (Conflict Of Interest) csoportok definiálásával oldja meg a modell. Minden szubjektumhoz tartozik egy n-elemű címke, amely minden COI csoporthoz maximum egy elemet tartalmazhat. Egy címkézett objektumhoz akkor lehet hozzáférni olvasásra, ha a szubjektum címkéjében az objektum címkéi közül minden COI classnál ugyanaz, vagy üres címke van. Ha egy szubjektum hozzáfért olvasásra egy objektumhoz, a szubjektum címkéje a továbbiakban a két címke uniója lesz. A szubjektum címkék két bejelentkezés között megőrződnek.

2.7. Clark-Wilson modell

A Clark-Wilson[33] modell lényege, hogy a védett adatokhoz csak védett procedúrákon keresztül lehet hozzáférni.

Vannak védett adatok (Constrained Data Item; CDI) és nem védett adatok (Unconstrained Data Item; UDI) UDI-ből vagy a CDI-k egy osztályából egy másik CDI osztályba való transzformációt Transzformációs Procedúrák (Transformation Procedure; TP) végzik. A TPknek *igazoltaknak* kell lenniük ahhoz hogy a tranzakciót elvégezhessek. Az operációs rendszernek biztosítani kell, hogy a CDI-k csak az azokat manipuláló TP-ken keresztül legyenek elérhetőek, csak azok számára a felhasználók számára, akik jogosultak a TP-t, és azon keresztül a CDI-t elérni. Ez valami olyasmi, mint egy webszerver, ami tele van szórva PHP scriptekkel: A webszerver mögötti adatbázisban és a filerendszeren vannak a CDI-k, a PHP scriptek pedig a TP-k. A felhasználói input pedig az UDI. Jó esetben ezeket a PHP scripteket megvizsgálják, hogy valóban azt csinálják-e minden inputra amit elvárunk tőlük.

2.8. Privacy Modell

A privacy modell egy kicsit hasonlít a Bell-LaPadula DoD kiegészítéséhez, de az integritási címkéket teljesen másra használja, nevezzük hát őket privacy címkéknek: Ezek a címkék egy-egy felhasználási körét írják le az adatoknak. A felhasználóknak van egy engedélyezett halmaza a privacy labelekből, de egyszerre csak egy ilyen címkét használhat. Hogy éppen melyiket, egy TS-el adhatja meg. A dolog lényege a privát információ megőrzése.

Nézzük egy példán a dolog lényegét: Van egy kórház, ahol a betegek személyes adatait őrzik. Ezek három kategóriába tartoznak: pénzügyi adat, a kezeléshez használható adat, kutatáshoz használható adat. A betegek megmondhatják, hogy az adataikat milyen célokra lehet használni. Tegyük fel hogy egy orvos kutatni akar: beállítja, hogy ő most kutat, és ekkor csak a kutatásra használható ada-

tokhoz férhet hozzá. Ha gyógyítani akar, akkor meg csak a gyógyításhoz használhatóakhoz.

2.9. Háló modellek

Mint az első példán láthattuk, a \Rightarrow operátornak egy informatikai rendszerben reflexívnek kell lennie; egy biztonsági osztályon belül nem érdemes megiltani az adatáramlást.

András, Béla és Cecília példáján láttuk azt is, hogy ha $A \Rightarrow B$ és $B \Rightarrow C$ akkor $A \Rightarrow C$, vagyis a \Rightarrow operátor tranzitív.

Ugyanazon a példán azt is láthattuk, hogy a \Rightarrow relációnak érdemes olyannak lennie, hogy ha $A \Rightarrow B$ és $B \Rightarrow A$ akkor $A = B$, hiszen ellenkező esetben nem „fogja” az információáramlást.

Ezt a tulajdonságot antiszimmetriának nevezzük, a reflexív, tranzitív és antiszimmetrikus relációkat pedig részleges rendezési relációknak. Tehát a \Rightarrow reláció a biztonsági osztályokon olyasmi mint a \leq reláció az egész számokon (pontosan olyan akkor lenne ha lennének olyan egész számok amiket nem lehet összehasonlítani).

Mint tudjuk, az informatikában minden információ amivel dolgozunk, véges. Így a biztonsági osztályok száma is véges lesz.

A sandboxos példában csak futólag említettük a hálózatot. Általában az interneten elérhető adatok publikusak; az ő biztonsági osztályuk kisebb mint bármely más biztonsági osztály. Tehát a biztonsági osztályok halmazán értelmezünk legkisebbet.

Két adat kombinációjának az eredménye a józan paraszti ész szabályai szerint a két adat biztonsági osztálya közül a nagyobbik kell hogy legyen Mivel a \Rightarrow operátor nem minden két biztonsági osztályra van értelmezve, ezért úgy mondjuk, hogy ez egy legkisebb felső határ határoz meg.

A fenti okoskodásból a következő szabályok jönnek ki, amelyeket Denning axiómáinak[20] nevezünk:

- Az SC halmaz véges.
- $A \Rightarrow$ operátor egy részleges rendezés SC -n.
- SC -nek van alsó határa a \Rightarrow operátorra nézve.
- $A \oplus$ operátor egy teljesen definiált legkisebb felső határ operátor.

Megmutatható, hogy ezeknek a szabályoknak a teljesülése esetén a hozzáférésvédelmi modell egy véges hálót alkot[15][20].

A hozzáférésvezérlési modellek nagy része véges hálóval leírható vagy közvetlenül, vagy némi matematikai trükközés után.

Például ilyenek a következők:

- Bell-LaPadula modell
- Biba modell
- Kínai fal modell
- Low Water Mark modell

Létezik még a Clark-Wilson modellnek is olyan implementációja, amely véges hálós modell (a Bell-LaPadula DoD kiterjesztése) segítségével történik. Namármost meg lehet mutatni, hogy a véges hálóval leírható modellek egymásba alakíthatóak, sőt két ilyen modell együttes használata (amikor mindkét modell szabályai érvényesülnek) is véges hálós modell. Ez nem feltétlenül jelenti azt hogy bármely implementáció képes bármely implementációt emulálni.

3. MAC implementációk Linuxon

A Linuxos hozzáférésvezérlési implementációk vizsgálatánál a következő szempontokat lehet például figyelembe venni:

a kompatibilitás ára Mennyire kell átírni a már meglévő programokat ahhoz, hogy működjenek a modell által felállított keretek között is[24].

az implementáció megbízhatósága Mennyire pontosan valósítja meg a modellt az implementáció[22], és maga az implementáció mennyire stabil. Általában a MAC implementációk kernel térben futnak, tehát egy kis programozási hiba kernel pánikot, fagyást eredményezhet.

A kompatibilitás mértékét megnöveli az, ha a namespace-eket virtualizálni lehet: a különböző biztonsági szinteken futó programok pl különböző /tmp könyvtárat használnak. A kompatibilitás problémája az interprocessz-kommunikáció során jön elő még igen markánsan. Itt is gyakran segíthet a namespace virtualizációja, illetve kemény fejtörést okozhat az, hogy egy nevezetlen pipe-ot hogyan értelmezzünk a modell keretei között.

Fontos dolog a Linux capability rendszerével való együttműködés Az az implementáció, amely valamely módon nem számol a capabilityk létezésével, sok problémát okozhat.

Szempon a modelltől „kilógó” információáramlások kezelése is. Szükség lehet arra pl, hogy egy „secret” dokumentumot „public” besorolásúra minősíthesünk vissza adott esetben. Ezt általában a Clark-Wilson modell TP-inek megfelelő

programokkal szokás elérni. Ezeknek a programoknak adnak egy `MAC_OVERRIDE` capabilityt, és persze csak bizonyos felhasználók futtathatják őket.

3.1. RSBAC

Ez az implementáció[8] egy általános hozzáférésvezérlési keretrendszer. A tárgyalt modellek közül az alábbiakat ismeri:

MAC MAC alatt gyakran a Bell-LaPadula modellt értjük. Itt is ez van implementálva, méghozzá a DoD változat dinamikus címkékkel

Privacy Modell Ez a Fischer-Hübner féle modell[27] mintaimplementációja.

ACL Mint feljebb láttuk, az RSBAC egy állományrendszer független ACL implementáció.

A nem tárgyalt modellek közül pedig az alábbiakat:

- Functional Control (FC)
- Security Information Modification (SIM)
- Malware Scan (MS)
- File Flags (FF)
- Role Compatibility (RC)
- Authentication (AUTH)

Az rsbac nagyon pontosan, szépen implementálja a modelleket, de kevés figyelmet fordít a kompatibilitási árra, hogy egy már meglévő környezetre a programok átírása nélkül lehessen a modellt rávarrni. Nem veszi figyelembe a capabilityket, nincs namespace virtualizációs lehetősége, és az unnamed pipe-ok miatt a MAC modellje csak erős rendszerbeli változtatásokkal tehető használhatóvá. Pl egy RSBAC-ot használó rendszert csak „rc” shelllel sikerült megfelelően használni[28].

3.2. Medusa

A Medusa[25] a hozzáférésvezérlést a „virtual space”-ekre alapozza. Minden objektum rendelkezik egy bitmappal, ami megmondja hogy ő melyik virtual space-ekben van benne. A processzek ezen kívül még három ilyen bitmappal rendelkeznek, amik megmondják, hogy valamely processz melyik virtual space-be írhat, melyikből olvashat, és melyik processzekkel végezhet IPC-t. Ezen kívül a

medusa támogatja a rendszerhívások átírását és a file redirekciót. Érdekes tulajdonsága, hogy a virtual space alapján képes „eltüntetni” állományokat. A Medusa segítségével bármely háló modellre épülő hozzáférésvezérlési modellt ki lehet alakítani, amihez a 32 bit elég. Az lme.linux.hu-n[29] pl egy Bell-LaPadula modellhez hasonló rendszer került kialakításra. Az első példában bemutatott rendszerhez hasonlóan van kialakítva több sandbox, amelyek több helyen egymásba vannak ágyazva. A \Rightarrow operátor által adott szabályokat a virtual space-ek segítségével tartatjuk be. A downgrade-hez és a címkeváltáshoz a TP-k egy-egy bash instancia segítségével vannak megoldva, amikor is az elindított program megkapja azokat a jogosultságokat, amik a művelethez kellenek.

Az ezen az előadáson használt laptopon is egy hasonló rendszer fut.

A medusa nagy problémája az, hogy még nem kiforrott, gyakran nem azt csinálja amire az ember a dokumentáció alapján következtetne, sőt heisenbugok vannak benne. Mivel a redirekció könyvtárakra és symlinkekre nem működik, valamint a pty-k és pts-ek nincsenek különlegesen kezelve, elég sok covert channel van az így felépített rendszerben.

3.3. LOMAC

A LOMAC[24] egy Low Water Mark modell implementáció, amelynek elsődleges szempontja az hogy a kompatibilitás árát csökkentse. Ezt úgy éri el, hogy az implementáció kernel modulban van, és a rendszerhívásokat wrappeli meg, valamint a modell alkalmazásánál úgy választották meg a modellben és a Linuxban található objektumok hozzárendelését, hogy az minél kevesebb problémát okozzon.

3.4. MAC

Ez is egy Bell-LaPadula DoD modell implementáció[30], de csak az integritás-cimkéket implementálja, így az általa kialakított biztonsági szintek teljesen diszjunktak. Viszont jó hálózattal kapcsolatos támogatása van: pl minden „compartmentnek” saját routing táblája van.

3.5. Egyéb megoldások

Mint azt érezhetjük, a Clark-Wilson modellt aránylag egyszerű setuid-os programokkal és különleges userid-kkel nagyjából implementálni[36]. Erre pedig csaknem bármilyen modellt rá lehet húzni, persze a modell megbízhatósága a TP-k megbízhatóságától erősen függ. Erre S.N. Foley ráépített egy elméleti keretet[34], amivel elég sok modellt lehet implementálni, és megmutatta, hogy pl egy kínai fal modellt hogyan lehet[35].

4. Hálózati alkalmazás

Azokat a tűzfalszerű eszközöket, amelyek kötelező hozzáférésvezérlést hajtanak végre, guardnak nevezzük. A Guardok implementálásával a következő problémák vannak:

A Bell-LaPadula modell az írás és olvasás műveletét mellékhatások nélküli, atomikus műveletnek tekinti. Sajnos az élet nem ilyen egyszerű, gondoljunk csak arra, hogy mi történik akkor amikor http protokollon lekérünk egy oldalt. Ezt a műveletet mindenki olvasásnak tartja, mégis azzal kezdődik, hogy *írunk* a hálózati kapcsolatra, és általában nem is keveset. De nézhetjük azt az esetet is, amikor írni akarunk egy file-t egy ftp szerverre. Amíg eljutunk addig hogy a „STOR” parancsot kiadhassuk, több üzenet jön a szervertől, feltöltés közben folyamatosan jönnek a TCP csomagjaink vételét elismerő csomagok, és a tranzakció végén annak befejezését elismerő feedback. Ezeknek az „ellenirányú” információáramlásoknak a kihasználásával igen nagy sávszélességű, úgynevezett *covert channel*eket[31] lehet kialakítani. Most képzeljük el azt az esetet, amikor valaki ezt a covert channelt arra használja, hogy információkat juttasson ki vele egy védett szerverről, vagy egy magát web böngészőnek kiadó programon keresztül shell parancsokat hajtasson végre a tűzfal mögött lévő védett rendszeren.

A másik tipikus probléma az, amikor egy alkalmazás valamilyen adatot vár, például egy imap szerver egy felhasználói nevet. De a csúnya gonosz crackerek úgy állítják össze a felhasználónevet, hogy az az imap szerver stackjén átcsordulva egy általuk megadott programot futtassanak az imap szerver nevében[32].

A probléma tehát az, hogy az általában vezérléshez tartozónak tekintett információt fel lehet használni információ továbbítására, és az általában információnak tekintett információt fel lehet használni vezérlésre.

A mostanában elterjedt protokolloknál sajnos ezeket a mellékhatásokat nem lehet kiküszöbölni, de sokat lehet tenni azok minimalizálása érdekében. Az első feladat az adat megkülönböztetése a vezérléstől, a második pedig a fennmaradó covert channelek minimalizálása.

A covert channelek felderítése és minimalizálása önmagában is szép feladat, most maradjunk az adat és vezérlés megkülönböztetésének a hozzáférési modellre gyakorolt hatásánál.

Ha a vezérlést leválasztjuk az adatról, az olvasás és az írás művelete kezd hasonlítani arra az eszményképre, amire a Bell-LaPadula modell épít. Viszont megjelenik két új művelet, a vezérlés és a visszajelzés. Ez a két művelet ideális esetben adatot nem hordoz, tehát a bizalmasság szempontjából nem kell velük foglalkozni. Itt a lényeg az integritásban van, tehát a Biba modellhez hasonló szabályok megfelelően mutatkoznak. Ha megengedünk dinamikus címkeképzést és bevezetjük a DoD integritási címkét, megkapjuk a következő hozzáférési modellt:

- $SC = SC_s \times SC_i$ ahol \leq teljes rendezési reláció SC_s -en, és \subseteq részleges rendezési reláció SC_i -n.
- $\lambda_{obj}(o) \in (O \rightarrow SC)$
- $\lambda_{minwrite}(s) \in (S \rightarrow SC)$
- $\lambda_{maxread}(s) \in (S \rightarrow SC)$
- $\lambda_{subj} \in (S \rightarrow SC \times SC)$, $\lambda_{subj}(s) = (\lambda_{minwrite}(s), \lambda_{maxread}(s))$
- $read \in S \times O \rightarrow SC \times SC$, $read(s, o) = (\lambda_{obj}(o), \lambda_{maxread}(s))$
ha:
 $\lambda_{maxread}(s) \leq \lambda_{obj}(o)$
- $write \in S \times O \rightarrow SC \times SC$, $write(s, o) = (\lambda_{minwrite}(s), \lambda_{obj}(o))$
ha:
 $\lambda_{minwrite}(s) \geq \lambda_{obj}(o)$
- $control \in S \times O \rightarrow SC \times SC$, $control(s, o) = (\lambda_{minwrite}(s), \lambda_{obj}(o))$
ha:
 $\lambda_{minwrite}(s) \geq \lambda_{obj}(o)$
- $feedback \in S \times O \rightarrow SC \times SC$, $feedback(s, o) = (\lambda_{obj}(o), \lambda_{maxread}(s))$
ha:
 $\lambda_{maxread}(s) \leq \lambda_{obj}(o)$

A fenti modell röviden azt mondja, hogy a control és a feedback műveleteket úgy értelmezzük, mintha az adatáramlás pontosan az ellenkező irányba folyna mint amerre ténylegesen megy a vezérlőinformáció, vagy másképpen kifejezve: az írás és olvasás műveletekre a Bell-LaPadula, a vezérlés és feedback műveletekre a Biba modellt alkalmazzuk.

Az hogy adott esetben mi számít adatnak és mi vezérlésnek, az applikációs proxy feladata eldönteni. Mint láttuk, ez nem triviális, alkalmazásfüggő feladat. Ehhez kell egy olyan apparátus, amely jelenleg csak a Zorp[14] tűzfalszoftverben van.

A modell érdekes tulajdonsága, hogy mindig szubjektum-objektum kapcsolatról beszél, noha az interprocessz kommunikációban a szubjektum-szubjektum kapcsolat a szokásos. Szubjektumnak mindig a kezdeményező hálózati entitást értelmezzük, objektumnak pedig azt a hálózati entitást, amely felé a tranzakció kezdeményeződött. Ezek a szerepek elméletben egy összetettebb tranzakció során megfordulhatnak, ez a modellt nem befolyásolja. Példaként tekintsük azt a tranzakciórészletet, amikor egy ftp szerverre jelentkezünk be:

user ftp

331 Anonymous login ok, send your complete e-mail address as password.

Az első sor egy control operáció, tehát csak akkor engedélyezett, ha az ftp kliens „magasabb vagy egyenlő” mint a szerver. A második sor egy feedback, tehát akkor engedélyezett ha a szerver magasabb vagy egyenlő. (Ezt a proxy úgy oldja meg, hogy nem az eredeti, hanem egy szabvány üzenetet küld.) Viszont ha a második sort úgy értelmezzük hogy a szerver volt a szubjektum, és a kliens az objektum, akkor ez egy control operáció, tehát csak akkor engedélyezett, ha a szerver magasabb vagy egyenlő. Tehát a helyzet ugyanaz.

5. Összefoglalás

Áttekintettük a legfontosabb hozzáférésvezérlési modelleket, azok tulajdonságait. Megnéztük, hogy melyik modellnek milyen Linuxos implementációja van. Végül bemutattunk egy hozzáférésvezérlési modellt, amely a hálózati forgalomszabályozással kapcsolatos problémákat próbálja meg kezelni[14].

Hivatkozások

- [1] Madách: az ember tragédiája (<http://www.mek.iif.hu/porta/szint/human/szepirod/magyar/madach/tragedia/>)
- [2] Az igazi programozó (<http://www.date.hu/zsguthy/humor/igazi.htm>)
- [3] Brian W. Kernighan and Rob Pike: The Unix Programming Environment Prentice Hall, Inc., 1984. ISBN 0-13-937681-X (paperback), 0-13-937699-2 (hardback).
- [4] Trusted unix working group (trusix) rationale for selecting access control list features for the unix (r) system (<http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-020-A.txt>)
- [5] Extended attributes and ACL for Linux (<http://acl.bestbits.at>)
- [6] XFS: A high-performance journaling file system (<http://oss.sgi.com/projects/xfs/>)
- [7] Trustees project (<http://www.braysystems.com/linux/trustees-dw.mhtml>)
- [8] Rule Set Based Access Control (RSBAC) for Linux (<http://www.rsac.de/>)

- [9] W.Z. Venema, „TCP WRAPPER, network monitoring, access control and booby traps”, UNIX Security Symposium III Proceedings (Baltimore), September 1992. (ftp://ftp.porcupine.org/pub/security/tcp_wrapper.ps
ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6.tar.gz)
- [10] Negyedik generációs tűzfal Linux-on Kadlecsik József <kadlec@sunserv.kfki.hu> KFKI RMKI Számítógép Hálózati Központ (<http://nws.iif.hu/NwScd/docs/nevjegy/nj54.htm>)
- [11] TIS Internet Firewall Toolkit (<http://www.tis.com/research/software/>)
- [12] Dante homepage (<http://www.inet.no/dante>)
- [13] T.Rex homepage (<http://www.opensourcefirewall.com>)
- [14] Zorp homepage (<http://www.balabit.hu/zorp/>)
- [15] Ravi S. Sandhu. Lattice-based access control models. IEEE Computer, 26(11):9–19, November 1993 (<http://heavenly.nj.nec.com/did/87719>)
- [16] chroot(1) man page
- [17] An Evening with Berferd In Which a Cracker is Lured, Endured, and Studied by Bill Cheswick (<http://www.securityfocus.com/data/library/berferd.ps>)
- [18] Bell, D.E. and LaPadula, L.J. „Secure Computer Systems: Mathematical Foundations and Model.” M74-244, Mitre Corporation, Bedford, Massachusetts (1975). (Also available through National Technical Information Service, Springfield, Va., NTIS AD771543.)
- [19] Biba, K.J. „Integrity Considerations for Secure Computer Systems.” Mitre TR-3153, Mitre Corporation, Bedford, Massachusetts, (1977). (Also available through National Technical Information Service, Springfield, Va., NTIS AD-A039324.)
- [20] Denning, D.E. „A Lattice Model of Secure Information Flow” Communications of ACM 19(5):236-243 (1976).
- [21] DoD Trusted Computer System Evaluation Criteria, 26 December 1985 (Supersedes CSC-STD-001-83, dtd 15 Aug 83). (<http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html>)
- [22] Common Criteria (<http://www.radium.ncsc.mil/tpep/library/ccitse/ccitse.html>)
- [23] Labeled Security Protection Profile (Version 1.b) (http://www.radium.ncsc.mil/tpep/library/protection_profiles/LSPP-1.b.pdf)

- [24] LOMAC: Low Water-Mark Integrity Protection for COTS Environments, Timothy Fraser NAI Labs tfraser@nai.com 3060 Washington Road Glenwood, MD 21738, USA (<ftp://ftp.tislabs.com/pub/lomac/>)
- [25] Medusa homepage (<http://medusa.fornax.sk/>)
- [26] Brewer, D.F.C and Nash, M.J. „The Chinese Wall Security Policy.” Proceedings IEEE Symposium on Security and Privacy, 215-228 (1989).
- [27] Simone Fischer-Hübner, Amon Ott: „From a Formal Privacy Model to its Implementation” for the National Information Systems Security Conference (NISSC 98) (<http://www.rsbac.de/niss98.htm>)
- [28] Személyes beszélgetés Wagner Endrével <wagner@balabit.hu>
- [29] Linux-Felhasználók Magyarországi Egyesülete (<http://lme.linux.hu>)
- [30] MAC30 implementáció (<http://users.ox.ac.uk/~mbeattie/linux/>)
- [31] Covert Channels — Here to Stay? (1994) Reprint Department Navy Naval Research Laboratory Ira S. Moskowitz Myong H. Kang (<http://citeseer.nj.nec.com/moskowitz94covert.html>)
- [32] Elias Levy (Aleph1): Smashing The Stack For Fun And Profit, Phrack 49 Volume Seven, Issue Forty-Nine, File 14 of 16 (<http://www.codetalker.com/whitepapers/other/p49-14.html>)
- [33] D. C. Clark and D. R. Wilson, „A comparison of Commercial and Military Computer Security Policies”, IEEE Symposium on Security and Privacy, 1987.
- [34] S.N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In 4th ACM Conference on Computer and Communications Security. ACM Press, 1997. (<http://citeseer.nj.nec.com/foley97specification.html>)
- [35] S.N. Foley Implementing Chinese Walls in Unix. Computers and Security Journal. 16(6):551-563, 16(6):551-563, December 1997. (<http://www.cs.ucc.ie/~s.foley/pubs/cwunix.ps.gz>)
- [36] W. Polk. Approximating Clark-Wilson access triples with basic UNIX controls. In Unix Security Symposium IV, pages 145–154, 1993.

A tűz, avagy mi ellen véd a tűzfal? (kivonat)

Mátó Páter

2000. szeptember 27.

Egyre gyakrabban hallhatunk számítógépes rendszerek ellen elkövetett gonosztettekről. Ezeknek az akcióknak a célja leggyakrabban az erőfitogtatás, néha tiltakozás esetleg rosszindulat. Egy azonban biztos: ahogy egyre fontosabbak lesznek számítógépes rendszerek, úgy válnak egyre inkább egy-egy szervezet vagy ember Achilles-sarkává.

A rendszerek gyakran esnek áldozatául belső ember vagy szervezet akciójának. Ha a támadók az ellopni vagy megsemmisíteni szándékozott információkhoz legálisan hozzáférhetnek, akkor a támadás megakadályozása szinte lehetetlen. Ha a megtámadni szándékozotti rendszerhez a támadóknak hozzáférése van ugyan, de nem elég magas szintű, akkor beszélhetünk lokális (számítógépen vagy hálózaton belüli) támadásról. Tételezzük fel, hogy a számítógéphez jogosan hozzáférő felhasználók jóindulatúak. Ebben az esetben a támadók a rendszert a számítógépes hálózaton keresztül igyekeznek hatalmukba keríteni.

Hogyan lehetséges, hogy egy nagy energiával felállított rendszer támadható? Hogyan kerülhetnek a programokba olyan hibák, melyek a teljes rendszer kompromittálódásához vezethetnek? Mi a tűzfal és mi ellen véd? Mire kell figyelni egy szerver telepítésénél? Ezekre a kérdésekre kísérel meg választ adni ez az előadás.

A felvetett témákat a következő kérdéskörök tárgyalásán keresztül járja körül:

- A tipikus biztonságot veszélyeztető programozási hibák
 - Nyelvspecifikus programozási hibák
 - Nyelvfüggetlen programozási hibák
- A számítógépes hálózatokról
 - a hálózat felépítése
 - alacsony szintű támadások

- a TCP/IP protokoll biztonsági kérdései
- A támadások fajtái és eszközei
 - félrevezetés (social engineering)
 - lehallgatás (sniffing)
 - címhamisítás (spoofing)
 - portscan
 - brute force
 - exploit-ok
 - DoS, DDoS
- A vég
 - várható maradványok
 - Újratelepítés
- Megelőzés
 - szolgáltatások szűrése
 - a naplók szerepe
 - csomagszűrő
 - folyamatos frissítés
 - wrapper-ek
 - alkalmazás tűzfalak

Template-rendszerek PHP alatt, a Prim template-parser rendszere (kivonat)

Noll János

2000. szeptember 27.

A résztvevők között elsősorban olyanokat várok, akik jelenleg is programoznak PHP-ben, vagy valamilyen CGI nyelven. Az ő tapasztalataikkal és hozzászólásaikkal, ötleteikkel még érdekesebb lehet a workshop.

A workshop egy része PHP specifikus, azonban nagyrészt bármilyen CGI nyelvben használható.

Témák:

- Mi az a template? Miért használunk template-eket? Hogyan válasszuk el a kódot a sablontól?
- Template szintaxisok, problémák, megoldások, ötletek.
- Pár szó a FastTemplate rendszerről, miért nem ezt használjuk a Prim-nél?
- Egy egyszerű parser: Prim template-parsere (LGPL-es). Szintaxis, mit tud, mit nem (és miért), tapasztalatok ...

Hálózati határvédelem eszközei (kivonat)

Scheidler Balázs

2000.09.16.

Napjainkban sajnos egyre kevésbé „biztonságos” hely a Net. Míg néhány évvel ezelőtt az Internet még félig-meddig bizalmi alapon működött, ma már mindenki gyanakvással figyel a „külvilágba”. Gyakran hallani tizenévesek „akcióiról”, mely során nagynevű cégek hálózataiba hatolnak be látszólag különösebb probléma nélkül. Ezen csínyek során súlyos milliók kerülhetnek, kerülnek veszélybe.

Az írott és íratlan sajtó persze igyekszik nagy lufit varázsolni mindenből, a sikerek legnagyobb oka viszont általában az elégtelen, rosszul kivitelezett vagy rosszul üzemeltetett hálózati határvédelem.

Előadásomban igyekszem bemutatni a rendelkezésre álló védelmi technológiákat:

- bastion host: egy egyszerű munkaállomás, két hálózati interfésszel, a külső hálózatot a belső hálózatról közvetlenül nem-, csak a bastion hostra való belépéssel lehet elérni.
- csomagszűrő router: olyan számítógép (vagy célszámítógép), mely a fejlécre vonatkozó szabályok alapján enged át, vagy tilt meg csomagokat.
- állapottartó csomagszűrő (stateful packet filter): olyan csomagszűrő, mely képes a csomagok között állapotot tartani, azaz megvizsgálni a csomagok tartalmát, és ez alapján döntést hozni.
- proxy tűzfal: csomagok továbbítása helyett kapcsolatok továbbítását végzi, és biztosítja, hogy egy kapcsolaton csak megadott protokoll haladhasson át, és annak is csak az engedélyezett részei.

A BalaBit IT Biztonságtechnikai Kft jelenlegi fejlesztése, a GNU/GPL alatti Zorp proxy tűzfalrendszer ez utóbbi csoportba tartozik, és célja, hogy minden területen felvegye a versenyt a jelenleg kapható tűzfalszoftverekkel.

A Zorp keretrendszerének felépítése lehetővé teszi az átengedett protokoll finomhangolását, a beágyazott protokollok kezelését, valamint az outband autentikációt. Ezeket a lehetőségeket mutatom be egy életszerű példán keresztül, mely egy képzeletbeli cég hálózati határvédelmét mutatja be.

Könyvtári karton feldolgozó rendszer

Szentpétery Ferenc <szefe@kvif.hu>

2000.09.26.

Kivonat

A rendszer a legnagyobb német könyvtár katalóguscéduláit dolgozza fel, viszi számítógépre. A továbbiakban olvasható az előadás vázlata.

Tartalomjegyzék

1. A feladat	61
2. Követelmények	61
3. Hardver	62
4. Szoftver és kiválasztása	62
5. Elrendezés	62
6. Működés	62
7. Tapasztalatok	63
8. Összefoglalás	63

1. A feladat

A programrendszerrel a Deutsche Bücherei katalógusállományának egy részét kell feldolgozni. A munka két fázisból áll: az 1. fázisban minden karton besorolásra kerül kategória (pl. disszertáció, monográfia, stb) és írástípus szerint. A rögzítők ezenkívül egy célprogrammal megnézik, hogy az adott karton szerepel-e már a könyvtár adatbázisában. A találat vagy nemtalálat tényét is rögzítik. A 2.

fázisban történik a kartonok szövegének a rögzítése. A katalóguscédulák szkennelve tiff formátumban érkeznek Németországból. Mindkét fázis két műveletet tartalmaz: rögzítést és ellenőrzést.

2. Követelmények

A rendszernek a következő követelményeknek kellett megfelelnie:

- kb. 5 millió rekord
- kb. 5 millió képfájl, 120 db CD-n,
- 40 db rögzítői munkaállomás
- rövid (1 mp) válaszidő, amikor az ellenőrök visszakeresik a kartonokat
- alacsony költségek

3. Hardver

A hardver kiválasztásánál a költségek alacsonyan tartása volt a legfontosabb szempont, a feladathoz szükséges minimumkonfiguráció: szerver: Pentium II 350 MHz, eleinte 128, most 512 MB RAM, UW SCSI winchester. Kliensek: Celeron 400 MHz, 32 MB RAM, 100 Mbit-es hálózat, 64 kbit-es bérelt vonal.

4. Szoftver és kiválasztása

Szerveroldalon Slackware 7.0, MySQL 3.22. Kliensoldalon jelenleg Windows, de hamarosan megkezdődik az átállás Linuxra. A szerverprogram C-ben, a kliens Javában készült. Mi alapján lettek az egyes szoftverelemek és programnyelvek kiválasztva?

5. Elrendezés

A szerveren fut az adatbáziskezelő és a szerverprogram (backend). A klienseken fut a rögzítőprogram (frontend) és az általa vezérelt képnéző. A szerver és kliens között egy egyedi protokoll szerint zajlik a kommunikáció. A kliens gépeken tároljuk a képeket. Miért előnyös ez a megoldás?

6. Működés

- a kommunikációs protokoll
- a szerver program működése,
- a kliens program működése,
- segédprogramok, scriptek
- különleges karakterek kezelése
- mentés
- a programrendszer outputja: a konvertált file

7. Tapasztalatok

- adatbáziskezelő problémák
- időnkénti teljesítményproblémák és megoldásuk
- memóriaigény
- a max. 16 index használata
- váltás az első időszakban használt Mini SQL-ről MySQL-re
- winchester meghibásodás és isamchk

8. Összefoglalás

Mint látható. a Linux és a rajta futó szoftverek sikeresen megállják a helyüket egy olyan feladat esetében, amit a szoftveres cégek szinte kivétel nélkül drága kereskedelmi szoftverekkel oldának meg, a hozzájuk szükséges igen erős és ennek megfelelően méregdrága hardverrel együtt.

