

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 címkékkal	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 sokféle jogot. Az erőforrás tulajdonosa leggyakrabban a felhasználó, de az sem ritka hogy a jogosultságokat az arra hivatott adminisztrátor oszthatja 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 hoz-

záférésvezérléssel 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 titkos, 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 initen kívül) semmi. Valamint álmodjuk azt, hogy a chrootolt környezetből nem lehet kitörni[17].

Matematikusul a fenti helyzetet így mondjuk:

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

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 felhasználóhoz 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. Nevezük 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 labelre 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-lemű címke, amely minden COI csoporthoz maximum egy elemet tartalmazhat. Egy címkézett objektumhoz akkor lehet hozzáférni olvasásra, ha a szubjektum címkéjében az objektum címkei 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 *igazol*taknak 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 olyasmint 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.

Szempontra a modelltől „kilógó” információáramlások kezelése is. Szükség lehet arra pl, hogy egy „secret” dokumentumot „public” besorolásúra minősíthessünk 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 rávarni. 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” shellel 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 megfelelőnek mutatkoznak. Ha megengedünk dinamikus címkeképzést és bevezetjük a DoD integritási címkéket, 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 folyrna 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://wwwrsbac.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 (Supercedes 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.

