Kiegészítő Linux kernel biztonsági megoldások összehasonlítása

A kiegészítő kernelbiztonsági megoldásokra irányuló vizsgálódásaim azzal kezdődtek, hogy egy SLES11 SP1 szerveren olyan formán szerettem volna üzemeltetni a cron démont, hogy a közönséges felhasználók időzített feladatai erősen korlátozott jogokkal fussanak, a rendszerfeladatok viszont nagyjából korlátoktól mentesen. Arra gondoltam, hogy ezt a célt egy mandatory access control (a továbbiakban MAC) rendszer használatával érem el. Használhatnám ugyan a kiterjesztett attribútumokra épülő POSIX ACL-eket is, amelyeket ha kiegészítünk az utóbbi időben egyre inkább használhatóvá váló Linux capability-kkel, akkor az ACL-ek néhány idegesítő gyengeségétől eltekintve elfogadható védelmi rendszert kaphatunk, de egy MAC rendszer használata mégis kézenfekvőbbnek tűnik.

Tulajdonképpen mi is egy MAC rendszer? A Wikipédia definíciója, és a hozzáférés-vezérlésben elfoglalt helye szerint kizárólagosan rendszergazdai irányítás alatt álló, a többi felhasználó belátására (discretion) nem alapozó, centralizál hozzáférés-vezérlési rendszer, és mint ilyen, a Discretionary Access Control (DAC), vagyis a hagyományos (pl. Windows/UNIX), elosztott felelősségvállaláson alapuló fájlrenszer-jogosultsági rendszer ellentétpárja. Már a Wikipédia is felhívja azonban a figyelmet arra, hogy a MAC név történelmi okokból szorosan egybeforrt a fenti definíció szerinti rendszerek egy konkrét típusával, nevezetesen a Multilevel Security (MLS, újabban MLS/MCS) rendszerekével, mégpedig oly mértékben, hogy több - általam MAC megoldásnak tartott - rendszer (pl. RSBAC, AppArmor) dokumentációja kizárólagosan ez utóbbi értelemben használja a MAC kifejezést. Az első definíció szerinti MAC rendszerek nem mindegyike nyújt azonban MLS/MCS szolgáltatást, sőt ez a háttérbe szorul, és inkább a Type Enforcement(TE)+Role Based Access Control(RBAC) alapú megoldások dominálnak.

A vonatkozó Wikipedia szócikkekből kiindulva nyilvánvalónak tűnik, hogy MAC megoldás címszó alatt nagyjából olyan rendszerre számíthatunk, ami a Linux kernelben azokon a pontokon, ahol az egyébként is hozzáférési döntést hoz a DAC és capability rendszer révén, további hozzáférés-vezérlési döntéseket hoz valamilyen központi, csak a rendszergazda (de nem a közönséges root account, hanem további jogosítványokkal rendelkező adminisztrátor(ok)) által módosítható adatbázis alapján. Az ilyen rendszerek közös jellemzője egyébként, hogy amit a szabványos DAC rendszer megtilt, azt nem bírálják felül, hanem csak annak a tiltását mérlegelik, amit a DAC egyébként engedélyezne. Más szóval a DAC szabályrendszerén enyhíteni nem lehet, csak szigorítani. A MAC rendszerek ideális esetben lehetővé teszik, hogy a lehető legegyszerűbb adatbázis segítségével olyan korlátozásokat vezethessünk be a Linux kernel hozzáférés-vezérlési rendszerében, amelyeket a DAC-ban csak bonyolult, átláthatatlan (rengeteg fájl jogosultságainak módosításával, sok POSIX ACL-lel, a felhasználók részére kiegészítő csoportok megadásával, filesystem capability-kkel, stb.) és megbízhatatlan módon (egy-egy frissítés után elkerülhetetlen újbóli jogosultságállítgatásokkal), vagy sehogy sem tehetnénk meg.

A MAC rendszerek alapvetően arról hoznak döntéseket, hogy mely subjectek (processzek) mely objecteken (fájlok, IPC csatornák, hálózati objeltumok, capability-k, resource-ok, más processzek, egyéb kernelszolgáltatások, stb.) milyen műveleteket végezhetnek el. A szabályrendszer átláthatóvá tétele érdekében, vagy más filozófiai meggondolásból néhány MAC rendszer korlátozásokat leíró adatbázisában a subjectek és és az objectek nem valamilyen természetes, a Linux kernelben egyébként is meglevő jellemzőjük alapján (pl. elérési út, UID, GID, szülőprocessz, stb.) szerepelnek, hanem absztrakt osztályokba sorolva, amely osztályokat valamilyen címke formájában külön össze kell rendelni az adott subjectekkel vagy objectekkel (mint pl. a SELinux egész fájlrendszert beterítő security.selinux nevű kiterjesztett fájlattribútumai esetében). Az RBAC

rendszereknél az ilyen absztrakt osztályok neve subjectek esetében általában domain, role vagy type, objectek esetében többnyire type.

Úgy döntöttem tehát, hogy megvizsgálok néhány MAC rendszert, és kiválasztom, melyik lesz számomra a legmegfelelőbb. Első körben hármat tartottam további vizsgálatra érdemesnek, a SELinux-ot, az RSBAC-t és az AppArmort, végül azonban a Grsecurity-t is beiktattam.

1. SELinux

A SELinux policy-ja elég sok fájlban oszlik szét. Ezek közül a tulajdonképpeni bináris policy, ami a kernelbe töltődik, a `/etc/selinux/POLICYNÉV/policy/policy.VERZIÓ` fájlban található, de vannak itt adatok a fájlrendszer megcímkézésére vonatkozóan (amit mindenképpen el kell végezni, hogy használatba vehessük a SELinuxot), a UNIX felhasználók SELinux tulajdonságaira vonatkozóan, stb. A bináris policyn kívüli policy-összetevők menedzselését többnyire a semanage paranccsal végezhetjük.

A SELinux policyjának felépítése rendkívül elegáns. Minden subject és object egységes módon van osztályozva, az ún. security contexttel. A context négy összetevője közül csak a type és a range értékek számítanak a szűkebb értelemben vett hozzáférésvezérlésben, de ezek közül is a range-ben szereplő level egyik összetevője, a sensitivity sohasem bírt igazán gyakorlati jelentőséggel.

A SELinux kernelbeli része nem biztosít közvetlen összefüggést a UNIX UID/GID értékek és a processzek security contextje között, de mivel nyilvánvaló, hogy az UID/GID váltásnak sok esetben contextváltással kell járnia (pl. login, SSH, felhasználói crontabok), ezt úgy oldották meg, hogy néhány userland-beli programnak jogot adnak az explicit contextváltásra, amelyet a libselinux könyvtár segítségével a `/proc/PID/attr` API-n keresztül tehetnek meg. Ehhez hasonló módon userland támogatásra van szükség olyan esetekben, ahol a type_transition szabályok korlátai miatt a SELinux kernel-beli komponense nem tudja biztosítani, hogy bizonyos rendszeresen újragenerált fájlok (pl. `/etc/passw`, `/etc/shadow`, `/etc/group`, logrotate) megfelelő security contexttel legyenek felcímkézve. Mindezek miatt elég sok userland-beli programcsomagból módosított verzióra van szükség ahhoz, hogy használatba vehessük a SELinux-ot.

A SELinux kernel-beli része az LSM technológiára épül, ezért élvezi a vanilla kernel fejlesztőinek támogatását, sőt része a mainstream kernelnek. Így, ha SELinuxot akarunk telepíteni egy azzal még nem rendelkező disztribúcióra, akkor módosított kernelre nem, de a SELinux userland eszközeire és a fent említett módosított programcsomagokra szükségünk lesz. Ezen felül kell még legalább egy policy. Kiindulási alapnak jó lehet a Tresys Technology által fejlesztett referencia-házirend, ami az adott disztribúciónak az FHS-től (vagy más referenciától) való eltérésének függvényében módosításra, testreszabásra szorulhat. És itt jön a bökkenő: A SELinuxban még az ilyen "egyszerű", a sima DAC-hoz képest nem sok szigorítást tartalmazó házirendek is több százezer szabályból állnak, így a testreszabásuk nem annyira rendszergazdáknak, mint inkább disztribútoroknak való feladat.

A SELinux előnye (az alább ismertetett RSBAC-hoz hasonlóan), hogy rendkívül finomfelbontású a biztonsági modellje, és a fájlok felcímkézésével (amelynek ötletét talán nem is annyira a UNIX inode-onkénti fajljogosultságaiból, hanem több évtizedes USA kormányzati MLS/MCS hagyományokból merítették) olyan jellegű jogosultságelkülönítést tudnak megvalósítani, amit az elérési út-alapú MAC megoldások (AppArmor, Grsecurity) nem. Véleményem szerint van néhány hátránya, amelyek talán az eleganciára való túlzott törekvésből adódnak: Az egyik az, hogy rendkívül bonyolult a karbantartása, embert próbáló feladat a konkrét rendszerhez való illesztése. A Red Hat pl. kétféle policyt szállít az oprendszereivel:

- Az alapértelmezett "targeted" nevűt, amelynél a rendszer nagy része korlátozások nélkül fut, és csak néhány kiválasztott program/démon jogkörét szűkíti a SELinux, de már ez a policy is több százezer szabályból áll.
- Van egy "strict" nevű policy is, amely jóval teljesebb körű korlátozást, így nagyobb biztonságot nyújt, de ennek használata a bonyolultsága okán "unsupported", és mivel előregyártott policyról van szó, nyilván testreszabást igényel egy-egy konkrét rendszer esetében.

A másik fő hátránya szerintem az, hogy amint feljebb már említettem, a kernel szemszögéből a processzek security contextje teljesen független az UID és GID értékektől, ezért tipikus esetben a felhasználókat csak a Linux DAC tudja megkülönböztetni egymástól jogosultság tekintetében, és kiterjedt módosításra szorul a userland (amint szintén említettem már fentebb).

2. RSBAC

A SELinux után az Amon Ott által szinte egyszemélyes projektként futtatott RSBAC rendszerrel kezdtem foglalkozni. Az RSBAC moduláris felépítésű. A kötelező AUTH és az ajánlott RC modulok együtt a SELinux TE/RBAC megoldásához rendkívül hasonló filozófiát képviselnek. Az rsbac_menu a Linux kernel ncurses alapú `make menuconfig` rendszeréhez hasonló külalakú konfiguráló program, és rendkívül szemléletes képet ad az RSBAC policy felépítéséről. Az opcionális MAC modul a SELinux MLS/MCS rendszeréhez hasonlít, de ennek nem igazán szenteltem figyelmet, mint ahogy a még elvontabb/elborultabb PM modulnak sem. A praktikusabb, rendszerközelibb ACL, CAP és RES modulok viszont egyszerűnek de hasznosnak tűnnek számomra.

Az RSBAC a SELinuxhoz hasonlóan inode-onként címkézi fel a fájlokat, de attól eltérően nem magában az inode-ban tárolja a metaadatokat kiterjeszett attribútum formájában, hanem - a fájlrendszerkvóták adatbázisához hasonló módon - minden fájlrendszer gyökérkönyvtárában létrehoz egy /rsbac.dat könyvtárat, és ezekben különféle bináris adatbázisfájlokban tárolja el a teljes policyt minden modulra vonatkozóan. A fájlokra inode-szám alapján hivatkozik, és mivel Linux (de legalábbis ext3 fájlrendszer) alatt könnyen előfordulhat, hogy egy fájl törlése után ugyanabban a könyvtárban egy frissen létrehozott fájl megkapja a törölt fájl inode-számát, felmerülhet az az aggály, miszerint az RSBAC alatt a törölt fájlok jogait megörökölhetik más, egész más célokat szolgáló később létrehozott fájlok, azonban ez a félelem alaptalan mindaddig, amíg nem mountoljuk fel a fájlrendszereinket nem RSBAC-képes kernellel, mert az RSBAC minden fájl törlésekor minden - az adott fájlra vonatkozó - adatot azonnal töröl az adatbázisából. Az RSBAC egyébként a számos biztonsági modul révén sokkal több metaadatot rendel a fájlokhoz, mint a SELinux, ezért kényelmetlen is lenne azokat xattr formájában tárolni. A SELinux esetében módosított init binárisra van szükség a policyt betöltéséhez, az RSBAC esetében azonban maga a kernel végzi el ezt a munkát, és semmilyen userland programnak nem engedi meg, hogy hozzáférjen a /rsbac.dat könyvtárakban tárolt adatbázisokhoz. Az userland és a kernel RSBAC kódja között az API egyetlen "security" nevű rendszerhívásból áll (legalábbis bizonyos architektúrákon). A SELinux és az RSBAC esetében egyébként azért kell a policyt a bootolási folyamat rendkívül korai szakaszában és

speciális módon betölteni a kernelbe, mert amint már említettem, ezek a rendszerek a subjecteket (processzek) és az objecteket (fájlok, IPC csatornák, stb.) speciális metainformációkkal (pl. security context) ruházzák fel, és ezeknek a MAC-specifikus címkéknek - egyebek mellett - a rendszer állapottároló mivoltából kifolyólag a kezdetek kezdetétől fogva jelen kell lenniük a kernel adatstruktúráiban.

Az RSBAC TE/RBAC rendszerében a SELinux sokösszetevős security contextje helyett a subjectek egy egyszerű "role" paraméter alapján vannak osztályozva, az objectek pedig "type" alapján. A processzek, mivel a subject mellett object szerepet is betöltenek, rendelkeznek külön role és type értékkel is. Ennek a TE/RBAC rendszernek a működése leginkább abban tér el a SELinuxétól, hogy itt a kernel egy rendkívül elmés ötlet, nevezetesen a fájlok rc_initial_role és rc_force_role attribútumai segítségével megoldja - mégpedig igen hatékony módon - azt, hogy a processzek UID váltása role-váltást idézzen elő. Mivel tehát a SELinux-szal ellentétben az RSBAC kernel nem csak bizonyos binárisok végrehajtásakor (a SELinuxnál ezt hívját type transition-nek), hanem (R)UID váltáskor is lehetővé teszi a processzek role-váltását, és amint feljebb írtam, a policyt sem egy módosított init, hanem maga a kernel tölti be, így az RSBAC jóval kevesebb támogatást igényel az userlandtól, mint a SELinux. Ez azt jelenti, hogy az userland-et a pam_rsbac.so modul bevezetésén kívül egyáltalán nem kell RSBAC specifikus módon megváltoztatni. Az SELinuxéhoz hasonló filozófia miatt itt is fennáll a fent említett címkézési probléma az `/etc/passw`, `/etc/shadow`, `/etc/group`, és logfájl jellegű fájlok újragenerálásakor, de a felhasználói adatbázis esetében ezt megoldották egy huszárvágással, nevezetesen a szokásos shadow fájlok helyett a kernelben (illetve a gyökérfájlrendszer /rsbac.dat könyvtárának egyik adatbázisában) tárolt UNIX felhasználói adatbázissal (ennek az eléréséhez szükséges az említett `pam_rsbac.so` modul), a logrotate vonatkozásában viszont nincs tudomásom hasonlóan egyszerű megoldásról, itt talán a különböző típusú logfájlok elkülönített alkönyvtárakban való tárolása segítene.

Az RSBAC a Grsecurity-hoz hasonlóan (de a SELinuxszal és az AppArmorral ellentétben) nem az LSM infrastruktúrára épül. Ennek oka többek között talán az, hogy az LSM integrációhoz mind az LSM-et magát, mind a szóbanforgó MAC megoldásokat erőteljesen meg kellene reformálni, és ezek az "egyszemélyes" projektek ezt nem tudják, nem is akarják felvállalni. Linus Torvalds részéről – úgy érzem – jogos az az elvárás, hogy az ilyen speciális területeken működő rendszerek egy keskeny, jól körülhatárolható felületen (ilyen az LSM) keresztül érintkezzenek a kernel többi részével, ne pedig több száz különböző helyen beavatkozva, főként talán azért, hogy karbantartó híján könnyen leválaszthatóak, kikapcsolhatóak legyenek, és egy-egy hibájuk miatt ne kelljen váratlan helyeken problémáktól tartani a kernel működésében és viszont. Úgy néz ki tehát, hogy ezek a "renitens" biztonsági projektek örökké a vanilla kernelfán kívül fognak fejlődni, és ez talán jól is van így.

Az RSBAC amellett, hogy teljes értékű MAC rendszer, további – a MAC szabályrendszerével egyszerűen (vagy sehogyan) le nem írható – praktikus biztonsági megoldásokkal szolgál, mint pl. a symlinkek átirányítása (pl. saját `/tmp` könyvtár mindenkinek és minden démonnak), egy szigorúbb – biztonsági megoldásként is használható – chroot jellegű rendszerhívás, az rsbac_jail hozzáadása, opcionális PaX integráció (ún. enhanced RSBAC kernel használata esetén), vírusvédelmi megoldások támogatása, speciális fájl flag-ek, stb.

Mindezen előnyök ellenére, bár az RSBAC tanulmányozásának korai szakaszában egyértelmű befutónak tartottam ezt a rendszert, később elvetettem a használatát a SELinuxéval összemérhető bonyolultsága okán, valamint amiatt, hogy nem találtam hozzá sem automatikus policy generáló eszközt (a másik három rendszerhez különböző képességekkel rendelkezésre állnak), sem előre gyártott jól kidolgozott policy-kat (igaz, a kernel maga generál hozzá egy nagyon egyszerű alap policyt).

3. Grsecurity

A Grsecurity az RSBAC-hoz hasonlóan csaknem egyszemélyes projekt Brad Spengler irányítása alatt. Ugyancsak az RSBAC-cal rokon módon nem LSM alapú, valamint szintén tartalmazza a zseniális PaX patchet, az RSBAC-cal ellentétben azonban nem opcionálisan, hanem kötelező módon, tehát nincs Grsecurity PaX nélkül.

A PaX a MAC rendszerekkel ellentétben nem a kernel egyébként is meglevő hozzáférésvezérlési döntéshozatalát fejleszti tovább, hanem a futó programoknak a hackerek általi "eltérítését" próbálja megakadályozni, pontosabban a programok memóriatérképének korrupcióját nehezíti meg. Kezdetben ezen cél elérése érdekében a virtuális memória bizonyos lapjainak végrehajtási tilalmával (PAGEEXEC/SEGMEXEC) és memóriatérkép véletlenszerűsítéssel (ASLR) küzdött a PaX, mára azonban több hatékony kiegészítő védelmi módszert vezetett be, pl. a PAGEEXEC/SEGMEXEC módszereknek a felhasználói processzeken túl a kernelre történő kiterjesztését (KERNEXEC), bizonyos típusú kernel exploitok (pl. null pointer dereference) elleni ötletes védelmet (UDEREF), és más szigorításokat a memóriakezelés területén. A PaX védelmi módszerei közül mára már a vanilla kernelben is meghonosodott a PAGEEXEC jellegű végrehajtási tilalom (de csak az ezt hardveresen támogató CPU-kon), és az ASLR, de a PAX megoldásai ezeken a területeken is radikálisabbak és hatékonyabbak.

A Grsecurity a SELinuxtól és az RSBAC-tól eltérően, de az AppArmorral megegyező módon egyrészt nem absztrakt címkék (pl. security context) alapján osztályozza a subjecteket és objectetek, hanem a vanilla kernel által már egyébként is biztosított metainformációk alapján (pl. (R)UID/(R)GID, elérési út/fájlnév), másrészt nem igényli a fájlok felcímkézését sem, mert a fájlokhoz való hozzáférést nem valamilyen inode-hoz köthető (vagy azokban tárolt) metainformáció, hanem egyszerűen a fájlok elérési útja/neve alapján végzi. Ennek két fő előnye van: egyrészt a Grsecurity (és az AppArmor) policy-ja összehasonlíthatatlanul egyszerűbb és átláthatóbb, mint a SELinuxé és az RSBAC-é, másrészt ezek a MAC megoldások a bootolási folyamat tetszőleges pontján (vagy akár a rendszer aktív futása közben) aktiválhatóak, deaktiválhatóak és tetszőleges mértékben átkonfigurálhatóak.

Mindkét típusba tartozó MAC rendszerre jellemző, hogy amíg aktívak, mindent megtesznek az ellen, hogy a gyengeségeiket ki lehessen használni: Csak a megbízhatónak minősített programok szűk körének engedélyezik a mountolást, hard linkelést, névtér létrehozást, security context címke-módosítást, stb., egy szóval a lehetőségeikhez mérten közelítenek a legkisebb jogosultság elvének betartásához. És ez az, amiben a négy vizsgált rendszer közül átlagosan talán a Grsecurity teljesít a legjobban éles környezetben.

A Grsecurity MAC megoldásának, az ún. RBAC rendszernek két hatalmas előnye van: a nagyon egyszerű, ugyanakkor praktikus és erőteljes policy szintaxis, és egy rendkívül hatékony tanuló mód (learning mode). Ezen előnyöket kihasználva speciális IT biztonsági szakismeretek vagy hosszas tanulás nélkül is könnyedén készíthetünk olyan policyt, amely nem csak bizonyos szolgáltatásokat szorít szigorú keretek közé (mint a SELinux általánosan használt targeted policyja vagy az AppArmor disztribútor által szállított gyári házirendjei), hanem a SELinux strict policyjához hasonlóan (illetve a tökéletes testreszabottsága okán még szigorúbban) érvényesíti a legkisebb jogosultság elvét.

A Grsecurity RBAC rendszere a processzeket (R)UID/(R)GID alapú role-okba vagy domain-ekbe, és ezen belül (a processzekek binárisainak elérési útja alapján) subjectekbe szervezi. Ez a kétszintű hierarchia a subjectek execve híváson keresztüli átörökítését lehetővé tevő szabályokkal kiegészítve egyszerű, de rendkívül hatékony és flexibilis policy struktúrát eredményez, ráadásul olyat, aminek a generálása jól automatizálható, és ezt a Grsecurity ki is aknázza a gradm parancs segítségével. A policy hatékonyságára jellemző, hogy semmiféle userland támogatást vagy nem igényel! A Grsecurity userland-je a PaX specifikus parancsokat is beleértve két-három binárisból és egyetlen

konfigurációs könyvtárból áll. A policy egyetlen szöveges fájlban van, a teljes userland kb. tíz fájlt tesz ki.

A Grsecurity az RSBAC-hoz hasonlóan az előbbiekben felvázolt RBAC rendszeren felül további praktikus, a `/proc/sys` felületen keresztül ki- és bekapcsolható védelmi "trükkökkel" rendelkezik. Itt is teljes értékű biztonsági megoldást faraghatunk a chroot rendszerhívásból, aktiválhatjuk az erőteljes Trusted Path Execution védelmet, szigoríthatjuk a `/proc` fájlrendszeren keresztül elérhető adatokhoz való hozzáférést, stb., de symlink átirányítás sajnos nincs.

4. AppArmor

Legutoljára maradt az AppArmor, amely a SELinux-hoz hasonlóan az LSM infrastruktúrára épül, és része a vanilla kernelnek (igaz, csak a 2.6.36-os verzió óta). Kezdetben az Immunix, később a Novell, jelenleg pedig az Ubuntut kiadó Canonical a fő fejlesztője. Az AppArmor a Grsecurityhez hasonlóan elérési út/fájlnév alapú biztonsági megoldás, de a MAC rendszeren felül nem tartalmaz kiegészítő védelmi mechanizmusokat.

A dokumentációjából hamar kitűnik, hogy az AppArmor policy szintaxisa is kísértetiesen hasonlít a Grsecurity-ére, de sajnos sajnos hiányzik a role/domain fogalma, bár a 2.7-es verziótól tervezték egy erre emlékeztető, de valószínűleg kevésbé erőteljes eszköz bevezetését, de úgy tűnik, nem sikerült teljesíteni a tervet, mert a 2.7-es kiadás még mindig a 2.5-ös verzió kernel kódját használja.

A grsecurityben subjectként megismert fogalom itt "profile"-ként köszön vissza. A role-ok hiánya sajnos azzal jár, hogy a SELinuxhoz hasonlóan a kernel-beli MAC kód nem tesz semmit annak érdekében, hogy a különböző UID-del/GID-del futó processzek különböző jogokat kapjanak. Ezt a gyengeségét az AppArmor a SELinux módjára az userland közreműködésével kompenzálja az aa_change_hat és aa_change_profile rendszerhívások valamint a pam_apparmor.so modult mozgósítva.

Az AppArmor profilok a Grsecurity subjecteknél flexibilisebb módon rendelhetőek hozzá a processzekhez (ami nem meglepő, hiszen részben rájuk hárul a role-ok hiányának kompenzálása). Az objectekre megadható végrehajtási jogot (x) négyféle - az öröklést szabályozó - módosító flag-gel láthatjuk el, ráadásul a négyféle flag közül három lehetővé teszi az AT_SECURE glibc paraméter explicit átadását a vermen keresztül a környezeti változók biztonsági célokat szolgáló kigyomlálásának érdekében.

Az AppArmor általában a disztribútorok által előre gyártott, a SELinux targeted policy-jéhez hasonló módon csak bizonyos kiválasztott rendszerszolgáltatásokat korlátozó policy-vel érkezik, de ezek egyszerűen szerkeszthetőek, sőt policy-generáló segédeszköz is rendelkezésre áll. Az AppArmorban egyébként a többi MAC megoldásnál természetesebb módon valósítható meg a kiválasztott szolgáltatások korlátozása és a rendszer többi részének korlátlan jogosultságokkal történő futtatása, ugyanis a másik három rendszer szigorú "minden tilos, amit nem szabad" elvével szemben itt korlátoktól mentesen fut minden processz, amihez nem rendelünk profilt (a profilokon belül azonban már itt is érvényesül a "minden tilos, amit nem szabad elv").

Összességében az a benyomásom, hogy a biztonsági profilok processzekhez kapcsolása leginkább a SELinux hasonló funkciójával rokonítható, az objektumok osztályozása viszont a Grsecurity módszerére emlékeztet.

Jelen előadás anyaga kibővített, hivatkozásokat is tartalmazó, közösségileg megvitatott formában megtalálható az alábbi címeken:

`http://hup.hu/node/108217`
`http://hup.hu/node/105652`
`http://hup.hu/node/107363`