Grsecurity – Az én házam, az én váram

A Linux kernelen alapuló operációs rendszerek sokféle módot eszeltek ki a rendszer biztonságossá tételére, különféle tűzfalakat tartalmaznak, igyekeznek azonnal foltozni a hibás csomagokat, erre aztán egyre eszesebb csomagkezelőket gyártottak, egyszóval igyekeztek a rabló-pandúr versenyfutásban nem nagyon lemaradni. Most itt a kernelszintű biztonsági megoldásokról lesz szó, hogy miképpen tudjuk a Linux elemi lehetőségein túl teljességgel uralni rendszerünket.

A birodalom helyzete

Elsősorban most a kiszolgálói feladatot ellátó rendszereket tárgyaljuk, de az itt leírtak alkalmazhatók minden változtatás nélkül munkaállomásnak használt környezetekre is. Mi is az a birodalom, ahol várat kap egy frissen telepített ***Linux*** teljhatalmú ura? Nem más ez, mint az internet, ahol a viszonyok elég puskaporosak. Ha az offline világban valahol autókat gyújtogatnak fel, első oldalas hír lesz, ha viszont egy szervert vernek szét virtuálisan (ahol a befektetett munkaóra esetleg többszörösen meghaladja egy autó árát), akkor esetleg valamelyik internetes portál lehozza hírként. A virtuális vandalizmus mindennapos esemény. Így aztán a kacsalábon forgó várunkhoz nem árt, ha védőfal és testőrség is jár. Ez minden internetre kapcsolt számítógépre igaz, azonban azokra, melyek kiszolgálói alkalmazásokat is futtatnak, különösen érvényes. A pingvines rendszerrel pedig kimondottan előszeretettel valósítanak meg efféle feladatokat. A ***Linuxnak*** vannak ugyan elemi megoldásai a biztonság fokozására, de szerencsére ezeken túlmutató eszközök is rendelkezésre állnak. Itt van mindjárt a ***grsecurity***. Ha a váras analógiát még nem unja a kedves olvasó, akkor úgyszólván a ***grsecurity*** elsősorban arról gondoskodik, hogy a várfalakon belül minden rendben legyen.

Grsecurity dióhéjban

A ***grsecurity*** egy biztonsági lehetőségeket megvalósító kernelfolt (***patch***). Használatához új kernelt kell üzembe helyezni, cserébe sok új eszköz pottyan az ölünkbe. Kivédjük a telepítésével a legtöbb buffer overflow-n alapuló támadást, megerősítjük a ***TCP/IP***-t és kapunk egy nagyon finoman hangolható hozzáférés-vezérlést (***ACL***). Ezen túl persze számtalan változás lép életbe a kernel színfalai mögött, hogy nagyobb biztonságban legyünk. Az ***ACL*** rendszerrel folyamatokra lebontva meg lehet mondani, hogy ki mit tehet a rendszerben, az erőforrásokkal és a hálózaton. Például szabályozhatjuk folyamatonként a processzor és memóriahasználatot. Ahhoz hogy egy kernelpatchet vezérelni tudjunk, szükség van egy ***felhasználói térben (userland)*** futó alkalmazásra is. Ez jelen esetben a ***gradm***. A módosított kernellel és a ***gradm*** segítségével fogjuk egy példán keresztül megnézni, hogy miben is képes a ***grsecurity*** támogatni bennünket a biztonságért folytatott sziszifuszi küzdelemben. A kipróbálást gyorsítják azok a virtualizációs technikák, melyekről a szeptemberi számban kiváló cikkek jelentek meg (***qemu***, ***vmware***). Az itt leírtak előfeltétele egy tesztrendszer, akár valódi, akár egy virtuális számítógépen. Ez a tesztrendszer a ***Debian GNU/Linux 3.1*** volt, de más rendszerekre is átvihetők az alkalmazott megoldások. Lehetőség szerint disztribúciófüggetlen lépéseket teszünk.

A tesztkörnyezet

Tegyük fel, hogy van egy internetes kiszolgálónk, amin egyetlen ***Apache 2*** (a továbbiakban az egyszerűség kedvéért

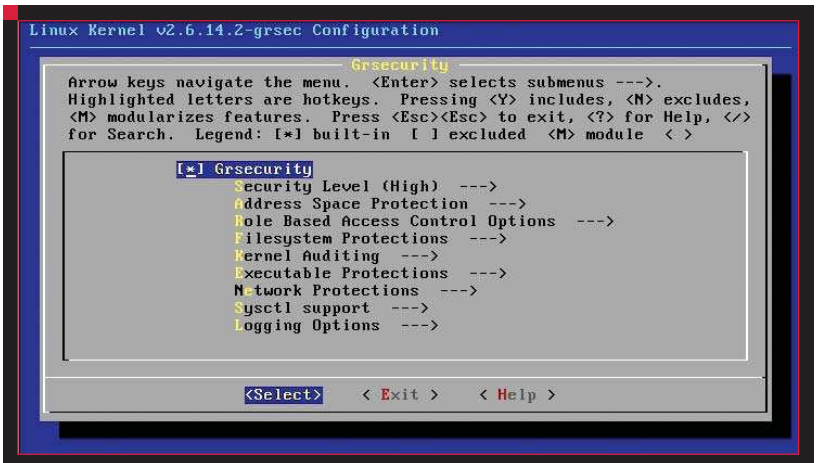
csak *Apache*) webkiszolgáló fut.
Ez a kiszolgáló képes *PHP* alkalmazásokat futtatni, de mivel a webmesteri feladatokat nem mi látjuk el, kiemelten fontos, hogy az Apache a legminimálisabb jogokkal bírjon. Hiszen ekkor, ha a webmester (aki az *Apache* által kiszolgált fájlokat/programokat gondozza) rosszindulatú kódot tölt fel, akkor a *www-data* felhasználó (amivel a webszerver fut), jogosultsága szerinti pusztítást tud végezni.
Ez ellen lehet védekezni, mondjuk *chroot* használatával, de az ellen például nem, ha a kiszolgált weboldalunk hallatlanul népszerű és például áldozatul esik a *Slashdot* effektusnak. Ekkor a gép minden erőforrását leköti a www szolgáltatás és esetleg egyéb kritikus alkalmazások leállhatnak.
Számtalan módon tudjuk szabályozni a folyamataink jogait a fájlrendszerhez, a hálózathoz és a gép erőforrásaihoz. A már előbb említett *Debian 3.1*-re az alaptelepítésen kívül csupán a *libncurses5* és a *libncurses5*, az *apache2* és a *php4* lett feltelepítve. Az utolsó kettő a feladathoz, az első kettő pedig a kényelmes majdani kernelfordításhoz (`make menuconfig` a `make config` helyett). Az *Apache 2* konfigurálva lett, hogy ténylegesen végrehajtsa a *.php* kiterjesztésű fájlokat. Ezek után felkerült a rendszerre a 2.6.14.2-es kernelforrás és az ahhoz való 2.1.7-es *grsecurity* kernelpatchnek és a *gradmnak* a fájljai.
Ezeknek a fájloknak a nevei letöltési címekkel a cikk végén megtalálható. Itt kapcsolódik be a cikk a történetbe. A cél egy olyan webkiszolgáló létrehozása, mely bár a *www-data* jogaival fut, egészen beszűkített jogai vannak, pontosan a feladathoz szabva. Igaz, hogy az *Apache*-hoz és a *PHP*-hez is számtalan olyan kiegészítés elérhető, mellyel a fentiek többé-kevésbé elérhetőek, viszont a most következők bármelyik alkalmazásra átvihetőek. Lássunk munkához!

Üzembe helyezés

A

```
gunzip grsecurity_file
```

utasítás után a kitömörített kernelforrásra a `patch` parancs kiadásával lehet a *grsecurityt* telepíteni. Lépjünk be abba a könyvtárba, ahol



1. ábra A kernel grsecurity-specifikus beállításai

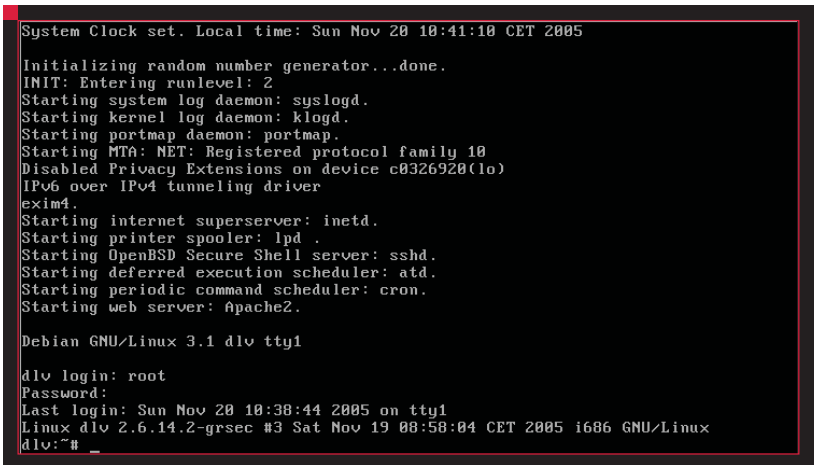
a *linux-2.6.14-2* könyvtár van, aztán hajtsuk végre:

```
patch -p0 < grsecurity_file
```

A `grsecurity_file` helyére a *grsecurity* kernelpatch teljes elérési útját írjuk. Ezután belépve a kernelforrás könyvtárába a

```
make menuconfig
```

következik. Bár a kernelfordítás minden csínja nem tartozik szorosan a cikk témájához, azért néhány dologgal érdemes tisztában lenni. Működő, éles rendszeren a kernel lecserélése nem várt problémákhoz vezethet, rosszabb esetben az új kernellel el sem indul a rendszer. A kernel beállításánál csak azzal foglalkozunk, ami ténylegesen pluszt jelent egy szokásos újrafordításhoz képest. Ha nem így tennénk, akkor a cikk címe „Hogyan fordítsunk kernelt?” lenne.
Mivel a *grsecurity* patchet alkalmaztuk, a `make menuconfig` által előcsalogatott menü kiegészül egy olyan alponttal, hogy *Security options* / *Grsecurity*. Itt kezdjük a munkát, miután a kernelforgatást saját szájunk íze szerint beállítottuk, ahogy egyébként is tettük volna. Az első opció a *Security Level*. A *grsecurity* nagyon összetett rendszer, akár minden elemét beállíthatjuk egyenként, vagy választunk az előre definiált szintek közül. A gyors beállítás érdekében egyelőre válasszuk a *High*-t, de azért nézzük át, miket lehet állítani! A *Role Based Access Control* alatt megadhatjuk, hogy hány elrontott jelszó után zárjon ki a rendszer, valamint hogy hány percig tartson a kitiltás, ha elérjük az előzőekben megadott küszöböt. A *Filesystem Protections* alatt le lehet tiltani, hogy a felhasználók lássák a */proc*-ban a nem hozzájuk tartozó folyamatokat és azok adatait, valamint szigorítani lehet a *chroot*-os környezeteket. A *Kernel Auditingban* számtalan naplózó/ellenőrző funkció kapcsolható be. Az *Executable Protections* / *Trusted Path Execution* által le tudjuk tiltani, hogy egyes felhasználók olyan programokat futtassanak, amik nem root tulajdonosú és csak root által írható könyvtárakban vannak. Ezzel nehezíthető rosszindulatú kód végrehajtása. Természetesen nem tökéletes a megoldás, hiszen egy nem megbízható felhasználó által felmásolt gonosz.pl-t a perl ezek után is vígan le fog futtatni. A *Networking Protections* alatt, ha a *High* biztonsági szintet választottuk, akkor csupán egy aktív lehetőség van, mégpedig a *socketek* korlátozására csoportazonosítók (*GID*) szerint. Ezen túlmenően is sok opció van, amire még érdemes vetni egy pillantást, az a *PaX*. Ez egy menüszinten található a *GrSecurityval*, a *grsecurity* patch-csel együtt felkerül. A *PaX* alapgondolata, hogy az általunk futtatott szoftverek hibásak, ezért ki kell védeni operációs rendszer szinten a gyakori hibalehetőségeket. Ide tartoznak a buffer overflow (túlcsordulás) hibák is. Többek közt ezekre gyógyír a *PaX*. Miután a menüpontokon átrágtuk magunkat (az egyes opciókon állva a *Help*-et választva egész kis dokumentációt



2. ábra Elindult a Grsecurityval felvértezett új rendszer

lehet előcsalni, melyek jól érthetőek), nem kell mást tenni, mint elmenteni a beállításokat és elindítani a fordítást. Ha virtuális gépen dolgozunk, akkor a kernelfordítás ideje alatt, géptől függően egy kávé, tea, sütemény vagy az egész e havi *Linuxvilág* áttanulmányozása is belefér. Persze nem kötelező virtuális gépen fordítani, megtehetjük, hogy a valódi operációs rendszer fordítójával lefordított kernelt felmásoljuk a virtuális hálózaton keresztül. A kernel a

```
make
make modules_install
```

parancsokkal települ, ezek után *Debian 3.1* esetén a */boot* alá másoljuk be az *arch/i386/boot/bzImage*-t (a kernelforráshoz képesti relatív útvonal). A *LILO*-t illetve a *GRUB*-ot módosítsuk úgy, hogy az új kernelt is ajánlja fel rendszerindításnál. Ne vegyük ki a régit, hiszen nem biztos, hogy olyan kernel jött létre, ami el is indul az adott gépen. Persze ha virtuális géppel dolgozunk, az ebben rejlő kockázat elég csekély. Miután végeztünk ezzel, jó esetben az újraindítás után semmi változást nem észlelünk, éppen úgy be tudunk jelentkezni, mint eddig.

A rendszabályok életbe léptetése

Az előzőekben megtett lépésekkel már sokkal előrébb jár a rendszerünk biztonságosságban, azonban az egyes programok személyre szabott megregulázása csak most fog következni. Először is ki kell tömörítenünk a *gradm* csomagot, aztán a *gradm2* könyvtárba belépve a

```
make
make install
```

parancsokkal fel tudjuk telepíteni az adminisztrációs felületet. A folyamat közben meg kell adnunk egy jelszót, mely a *grsecurity* beállításait védi. Miután ezzel végeztünk, újabb jelszó beállítására van szükségünk, hogy bekapcsolhassuk a rendszabályainkat. Ezt a

```
gradm -P admin
```

paranccsal tehetjük meg. Ezután ha a

```
gradm -E
```

utasítást kiadjuk, akkor életbe lépnek a gyári *grsecurity* korlátozások, melyek elég szigorúak. Például az *Apache*-ot nem lőhetjük le még rootként sem és az */etc/grsec/* könyvtárhoz sem férhetünk hozzá, ahol a beállítások lakoznak. Ügyes! Ha meg akarunk szabadulni az őrmestertől, aki a *grsecurity* képében megszállta a rendszert, a

```
gradm -D
```

kiadásával és a jelszó beírásával megtehetjük. Jó tudni, hogy engedélyezett *RBAC* (ez a szabályozó rendszer neve) mellett a gyári beállításokkal a `reboot` parancs sem működik rendesen, mert fájlrendszereket leválasztani sincs jogunk például. Talán már látszik, hogy a *grsecuritynek* megadott jelszavak mennyire fontosak. Egyfelől ha elvesztettük, és az *RBAC* engedélyezve van, akkor nagy bajban vagyunk, másrészt, ha kitalálható, akkor majdnem ott vagyunk, mintha fel se raktuk volna a *grsecurityt*. A teszt kedvéért készült két apró *PHP* szkript: az egyik a paraméterül kapott értéket beírja a */var/www/adat* fájlba, a másik pedig a */tmp/zagyva* fájlba ír véletlen számokat a végtelenségig. Nyilvánvaló, hogy az első szkripttel nincs semmi baj, de a másodikat jobb lenne letiltani. Egyrészről megtehetjük, hogy csak azokat a helyeket engedjük írni a webszervernek, amit majd ténylegesen kell neki, másrészről pedig letilthatjuk, hogy a */var/www* könyvtárban lévő dolgok közül csak azokat olvashassa el, amik ténylegesen rá tartoznak. Akár megtehetjük, hogy azt is, hogy az egész fájlrendszert csak olvashatónak lássa az *Apache* és mondjuk a *MySQL* socketet engedélyezni neki. A telepítés után a tesztrendszer /var/www könyvtárába bekerült a két bugyuta *PHP* szkript (*1. Lista*). A *grsecurity* rendszabályok nélkül és a *grsecurity* alapértelmezett rendszabályaival mindkét szkript kifogástalanul működik. A *rossz.php* 30 másodperc múlva leáll, mivel a *PHP*-ban az az alapértelmezett maximális futási idő, de addig kíméletlenül próbálja megtölteni(nem túl okos módszerrel) a */tmp* lemezrészt. Ezután a gyári */etc/grsec/policy* fájl a következő bejegyzéssel egészült ki:

```
subject /usr/sbin/apache2
        / r
        /var/www x
        /var/www/jo.php r
        /var/www/adat rw
        +CAP_NET_BIND_SERVICE
        /etc/ x
        /etc/apache2 rx
        RES_NPROC 2 3
```

Mit is mondanak ezek a sorok? A / könyvtár és minden ami alatta van (a tulajdonságok öröklődnek) csak olvasható az `/usr/sbin/apache2` számára, kivéve a `/var/www`, ebbe a könyvtárba beléphet ugyan, de nem olvashatja. A `/var/www/jo.php`-t olvashatja, de nem írhatja például. A `/var/www/adat`-ot írhatja és olvashatja. A `+CAP_NET_BIND_SERVICE` sor

1. Lista Egy jó és egy kevésbé jó PHP szkript

jo.php

```
<?
$fajl = fopen("adat", "w");
fwrite($fajl, $_GET["adat"]);
fclose($fajl);
?>
```

rossz.php

```
<?
$fajl = fopen("/tmp/zagyva",
➥w);
while(1)
{
  fwrite($fajl, rand(-500,
  ➥500));
  fflush($fajl);
}
fclose($fajl);
?>
```

mondja meg, hogy a folyamatnak lehetősége van a hálózaton szolgáltatásként üzemelni: lefoglalhat egy portot, címet magának. Az `/etc x` azért szükséges, hogy ne tudja olvasni a folyamat az */etc* alatt lévő fájlokat, hiszen ott lakozik például az */etc/passwd*, mely nem lenne jó, ha a tréfás kedvű webmester miatt publikussá válna. Viszont az */etc* alatt vannak az ***Apache 2*** beállítófájlok, ezért a következő sorban mindjárt megmondjuk, hogy ott szabad a gazda. Végül, mivel egy kis forgalmú teszthelyről van szó, a maximálisan elindítható ***Apache*** folyamatok számát 3-ban határozzuk meg. Minden ilyen `RES_VALAMI` korlátozás két számot vár el. Egy puha (első szám) és egy kemény korlátot. A puha korlát elérésekor a folyamat kiadhatja a `setrlimit` hívást, hogy többet szeretne az adott erőforrásból, amit egészen a kemény korlátig növelhet. A példából látszik egy bejegyzés általános szerkezete, annyit érdemes még hozzátenni, hogy a `subject` vonatkozhat könyvtárra és végrehajtható állományra egyaránt. Az */etc*-re vonatkozó bejegyzésnek következtében, ha a webmesterünk gonosz tréfát akarna velünk űzni, akkor a `print_r (file ("/etc/passwd"));` sorral semmire sem megy, hiszen az ***Apache*** ezt nem olvashatja. Egyébként is szerencsés, hogy a rendszerbeállításokat olvasni sem tudja egy ***PHP*** szkript. Feltűnhet, hogy az Apache most a saját naplóit sem tudja írni. Ez így igaz, a naplózási elérési utak mindenütt */dev/null*-ra lettek állítva. Valódi webkiszolgáló alkalmazásnál a naplózás elengedhetetlen, így megírhatjuk azt a sort is, ami engedélyezi, hogy a */var/log/apache2/access.log* illetve *error.log* írható legyen a folyamatnak. Jelen pillanatban a folyamatban ott tartunk, hogy sikerült a *rossz.php*-t működésképtelenné tenni, valamint a *jo.php* is csak a */var/www/adat* fájlba tud írni, sehova máshova.

A szabályok létrehozása nagy munka összetett alkalmazás esetén, ezért a ***gradm*** képes helyettünk összeállítani szabályokat! A

```
gradm -F -L /root/tanul
```

élesíti a tanuló módot és az eredményt a `/root/tanul` fájlba rakja. Ide azok az események fognak kerülni, melyeket a jelenlegi szabályrendszer megtiltott volna. Így aztán a parancsot kiadva érdemes végrehajtani az összes műveletet, futtatni az összes olyan programot, amire szükség van. De vigyázat, a szolgáltatások leállítása/elindítása nem ide tartozik, hiszen azt feltételezzük, hogy a támadó már rendszergazdai jogokkal bír a rendszer felett és éppen azon ügyködünk, hogy minél kevesebb kárt tehessen. Miután végeztünk a tevékenységekkel, adjuk ki a

```
gradm -F -L /root/tanul -O
➥/etc/grsec/acl
```

parancsot, így a ***gradm*** legyártja a naplókból a nekünk szükséges szabályokat. Amit végrehajtottunk, azokat a későbbiekben is futtatni tudjuk így. Nincs más teendő ezután, mint élesíteni az új szabályokat. Ha kiadjuk a

```
gradm -E
```

parancsot, akkor kapunk egy valamivel biztonságosabb Apache kiszolgálót. Be lehet állítani azt is, hogy a `gradm -E` már a futási szint részeként elinduljon. Az engedélyezéshez nem kell jelszó, így le kell gyártani a disztribúciónak megfelelő indítószkriptet, mely a `gradm -E` parancsot kiadja a betöltési folyamat azon pontján, miután a szigorítások nem akadályozzák a rendes üzemet. A cikkben beállítottakat (alapbeállítások és az ***Apache***-ra vonatkozó sorok) nem lehet a rendszerindítás során alkalmazni, a webkiszolgáló el sem indul úgy. Látható, hogy a legnagyobb munka a ***grsecurityval*** egy alkalmas szabályrendszer felállítása, amely aztán jobb esetben megvédi a fontos rendszert. Nem csodaszerről van szó: az alattomos szoftverhibákkal a továbbiakban is számolni kell, az emberi tényező pedig kikerülhetetlen. Ha rossz szabályrendszert dolgozunk ki vagy a ***gradm***-hoz használt jelszó gyenge, egyezik a root jelszóval vagy a monitor oldalára van felírva, akkor hiába dolgoztunk.

A részletes dokumentáció megtalálható a ➲ ***www.grsecurity.net*** oldalon a ***Papers*** menü alatt. A ***grsecurity*** most már a várunk őrzője, hatékonysága csak rajtunk múlik! Találjuk meg az arany középutat, amely a használhatatlan, de biztonságos (vö. kihúzott ***UTP*** csatlakozó) és az átjáróház (vö. ***open-relay*** levelezőszerver) között van valahol. Ennek feltérképezése végképp egyedi, minden rendszergazda saját környezetében kerülhet egyre közelebb az ***aurea mediocritashoz***.

Novák Áron
(aaron@szentimre.hu)
BME-VIK-es gólya, műkedvelő rendszergazda. Jelenleg leginkább a NetBeans-szel és mindenféle hordozható eszközzel foglalkozik, legalábbis mindazokkal amelyeket meg lehet szólaltatni Linux alatt.

KAPCSOLÓDÓ CÍMEK

A szükséges fájlok letölthetők a következő helyekről:

- ➲ http://www.hu.kernel.org/pub/linux/kernel/v2.6/linux-2.6.14.2.tar.bz2
- ➲ http://www.grsecurity.net/grsecurity-2.1.7-2.6.14.2-200511150641.patch.gz
- ➲ http://www.grsecurity.net/gradm-2.1.7-200511041858.tar.gz