A gyökér fájlrendszer titkosítása

Ha adataink fizikai biztonságát nem tudjuk garantálni, akkor nincs más lehetőségünk, mint a fájlrendszer titkosítása. Bár írásomban egy PowerPC alapú rendszer átalakítását fogom taglalni, az alapelvek más géptípusokon is változatlanok.

A *Linuxvilág* „Titkosított saját könyvtárak megvalósítása” (2003 október) című cikkében már ismertettem, hogyan lehet átlátszó módon titkosítani a kezdőkönyvtárakat. Ez alkalommal egy másik megoldást szeretnék ismertetni, a gyökér fájlrendszer titkosítását. Szó lesz a *GNU/Linux* rendszertöltési folyamatáról és a szükséges programokról, adok némi további útbaigazítást, megismerkedünk az *Open Firmware*-rel és néhány további figyelembe veendő tényezővel. A fogalmakat egy a *Fedora Core 3* előzetes kiadását futtató, *New World PowerPC* alapú *Apple iBook* alapján ismertetem. Természetesen az itt szereplő fogalmak és eljárások bármely készüléken, géptípuson és operációs rendszeren érvényesek, alkalmazhatók. Feltételezem, hogy rendelkezünk egy különálló *USB*-s flash memória egységgel, továbbá a gép belső programja képes a róla végzett rendszerindításra.

Feltételezem továbbá, hogy az olvasó már jártasságot szerzett a forrásfoltok telepítésében és a programok fordításában. A *Fedora Core 3 Test 3* terjesztés esetében az `mkinitrd` és az `initscripts` csomagokat kell megfoltozni a titkosított gyökér fájlrendszer támogatásával. Nem árt, ha a lemezkötetek kezelésével és a fájlrendszerek létrehozásával is tisztában vagyunk. A *Linux* terjesztések alapszintű telepítésével itt nem áll módomban foglalkozni.

Mielőtt rátérnénk a műszaki jellegű kérdésekre, meg kell ismernünk egy magasabb szintű fogalmat, a megbízás fogalmát. A megbízás kifejezést leginkább titkosítási és hitelesítési témák kapcsán hallhatjuk. Az elektronikus kulccsal rendelkező készülékeket alapesetben megbízhatókként kezeljük. Például, ha beírom a *PIN*-kódomat egy pénzkiadó automatába, akkor feltételezem, hogy nem fogja harmadik félnek kiadni azt. Hasonlóan, amikor beírok egy titkosítási kulcsot a számítógépembe, feltételezem, hogy nem fogja másnak átadni. Megbízom a számítógépemben, titkunk a kettőnké marad.

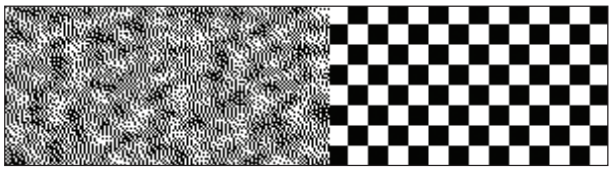
No de valóban megbízhatunk a számítógépünkben? Hacsak nem visszük mindenhova magunkkal, nyilván nem! Akkor is igaz ez, ha a lemezek titkosítva vannak. Vegyük a következő forgatókönyvet: míg alszunk, valaki ellopja a gépünket. A tolvaj – noha a visszafejtő kulcs hiányában egyelőre nem tud mi kezdeni vele – másolatot készít gépünk tartalmáról. Ezután a gép titkosított tartalmát lecseréli, majd visszateszi a gépet a helyére. Amikor másnap reggel felébredünk, a gép, ahogy minden nap, kéri tőlünk a kulcsot. Ez alkalommal azonban, amint megadjuk a kulcsot, elküldi azt a tolvajnak. Mivel most már adatainkkal és a kulcsunkkal egyaránt rendelkezik, el tudja olvasni dokumentumainkat.

Természetesen a példa kicsit erőltetett, de rendkívül szemléletes. A lényege az, hogy nem bízhatunk meg a számítógépünkben, hiszen nem tarthatjuk minden pillanatban szem előtt. Bármilyen jó is tehát a titkosítási rendszerünk, bizalmi alap nélkül épül fel.

Ha biztosítani akarjuk a számítógép rendszertöltési folyamatának megbízhatóságát, akkor el kell különítenünk tőle. A gondolat nem új, hiszen az autónknak is csak a kulcsát hordjuk magunkkal, magát az autót a parkolóban hagyjuk. A titkosítási kulcs természetes analógiája az autókulcsnak. A titkosítási kulcs könnyebben védhető, így a számítógépet nem kell mindenhová magunkkal vinnünk. Ennél egy kicsit tovább is lépünk, és a probléma megoldása céljából a számítógép indításához szükséges programot is a kulcson helyezzük el. A kulcs szerepét a flash meghajtó játssza. A rendszerindításért felelős program és a titkosítási kulcs együttes védelmével komolyan csökkenthetjük a rendszertöltő folyamat meghamisításának kockázatát.

Ehhez érteni kell a számítógép indításának folyamatát, ugyanis a titkosított gyökér fájlrendszer feloldása a rendszertöltő folyamat nélkülözhetetlen része. A jelenlegi üzembiztos, vagyis 2.6-os rendszermagok képesek az *initramfs* segítségével végzett rendszerindításra. (Lásd: *LWN.net*, *„Initramfs Arrives”*) Az *initramfs* egy `cpio` archív, a rendszermag most már tudja, hogyan bonthatja ki ezt egy *RAM*-lemezre. A kibontott fájlrendszer egy parancsfájlt tartalmaz, mely hagyományosan egy a gyökér fájlrendszer befűzéséhez szükséges rendszermagmodulokat betöltő parancsfájlt tartalmaz. Esetünkben ez a parancsfájl végzi majd a titkosított gyökér fájlrendszer visszafejtését is. A témáról bővebb tájékoztatást a *Linux* rendszermagforrások részeként elérhető *buffer-format.txt* és *initrd.txt* fájlokban lehet találni.

Linux alá számos fájlrendszer-titkosító felület létezik. *Jari Ruusu Loop-AES* megoldása is egy ezek közül. Számos *cryptoloop* változat is készült, ezek titkosított hurokeszközt biztosítanak. Írásomban a jelenlegi, 2.6-os rendszermagok



1. ábra Ha a fájlrendszer létrehozása előtt nem véletlenszerűsítjük a lemezrészt, akkor a támadó láthatja, mennyire van tele

2. ábra A lemezrész véletlenszerűsítésével eltitkolhatjuk annak kihasználtságát

által nyújtott ***dm-crypt*** felülettel foglalkozok. A ***Fedora Project*** jelenleg ezt a felületet helyezi előtérbe, és a ***dm-crypt*** modulok is szerepelnek a ***Fedora*** rendszermagcsomagjai között. Szükséges továbbá egy állandó jelleggel csatolt ***cryptsetup***. Segítségével leegyszerűsödik a ***dm-crypt*** eszközök kezelése. Végül, a rendszerindító fájlrendszer kezelésére a `parted` és a `hfsutils` szolgál.

Sajnos a ***Fedora Core anaconda*** telepítője alapesetben jelenleg nem támogatja a titkosított fájlrendszerre telepítést. Korlátain úgy léphetünk túl, hogy egy lemezrészt szabadon hagyunk, telepítjük a ***Fedorát***, megformázzuk titkosítottra a szabad lemezrészt, majd a telepítő által felmásolt fájlokat átmozgatjuk a titkosított részre. Az egyszerűség kedvéért feltesszük, hogy a ***Fedora*** telepítését két lemezrész használatával végezzük: a ***/dev/hda4*** a ***/home***, a ***/dev/hda5*** pedig a / elérési úttal van befűzve. Mivel a ***/home*** csak a ***Fedora*** telepítése után kerül feltöltésre, a ***/dev/hda4*** lesz a tartalék lemezrész, a ***/dev/hda3*** pedig a cseretár.

Fűzzük be a ***/dev/hda4*** meghajtót a ***/home***, a ***/dev/hda5*** meghajtót pedig a / elérési út alá, és telepítsük a ***Fedora Core 3*** terjesztést. A root-on kívül ne adjunk hozzá felhasználókat, mert a ***/home*** tartalma a későbbiek során törlésre kerül.

Ezen a ponton kapunk egy teljes értékű ***Linux*** rendszert. Mielőtt létrehoznánk a titkosított fájlrendszert, véletlenszerűsítenünk kell az általa elfoglalandó lemezrészt. Ezzel megelőzhetjük a lemez tartalmával kapcsolatos adatok kiszivárgását. Az *1. ábrán* egy elvonatkoztatott lemez látható, mely félig van tele, és véletlenszerűsítése nem volt megfelelő. A *2. ábrán* egy olyan lemez látható, melynek véletlenszerűsítését helyesen elvégezték, mielőtt megformázták volna a titkosított fájlrendszerrel. Vegyük észre, hogy az 1. ábra lemezéről lehet némi információt szerezni, például meg lehet állapítani, hogy az adatok csak félig töltik ki. A 2. ábrán látható lemeznél ilyesmitől nem kell tartani, a lemez akár teli, akár üres is lehet. Egy lemezrész véletlenszerűsítése tartalmának véletlenszerű adatokkal való felülírásával történik:

```
dd if=/dev/urandom of=/dev/hda4
```

A művelet végrehajtása eltarthat egy ideig, ugyanis a véletlenszerű adatok előállítása nem egyszerű feladat.

A ***/dev/hda4*** meghajtón az alábbi lépéseket követve hozhatunk létre titkosított ***ext3*** fájlrendszert:

1) Ellenőrizzük, hogy az `aes`, a `dm-mod` és a `dm-crypt` modul be van-e töltve a rendszermagba.
2) Válasszuk le a ***/home*** elérési út alól a titkosított gyökér fájlrendszer tárolására kiszemelt meghajtót vagyis a ***/dev/hda4***-et:

```
# umount /dev/hda4
```

3) Hozzunk létre egy véletlenszerű, 256 bites titkosító kulcsot, és mentsük el ***/etc/root-key*** néven:

```
# dd if=/dev/urandom of=/etc/root-key bs=1c
count=32
```

A kulcsot később át fogjuk másolni a flash meghajtóra.

4) Hozzunk létre egy dm-crypt eszközt, mely az imént létrehozott kulccsal kerül titkosításra:

```
# cryptsetup -d /etc/root-key create root /dev/hda4
```

A ***/dev/mapper/root*** most egy titkosított réteget szolgáltat a ***/dev/hda4*** felett. Alapesetben a `cryptsetup` egy ***AES*** algoritmussal titkosított `dm-crypt` eszközt hoz létre, és 256 bites kulcstér használatát feltételezi.

5) A ***/dev/mapper/root*** alatt hozzunk létre egy ***ext3*** fájlrendszert:

```
# mkfs.ext3 /dev/mapper/root
```

6) Fűzzük be az új fájlrendszert:

```
# mkdir /mnt/encroot
# mount /dev/mapper/root /mnt/encroot
```

7) Kész a titkosított fájlrendszer, fel kell töltenünk a ***/dev/hda5***, azaz az eredeti gyökér fájlrendszer tartalmával:

```
# cp -ax / /mnt/encroot
```

8) Végül létrehozunk egy bejegyzést a /mnt/encroot/etc/crypttab fájlban, a különféle segédprogramok innen tudhatják meg, hogy melyek a fájlrendszer beállításai:

```
root    /dev/hda4       /etc/root-key    cipher=aes
```

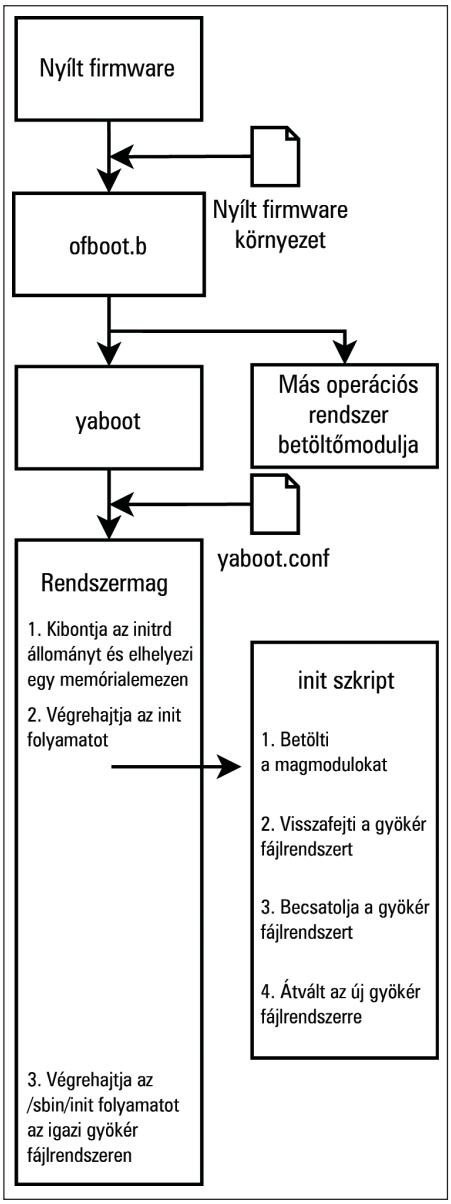
A titkosított fájlrendszer is készen áll, a továbblépéshez azonban ismernünk kell gépünk rendszertöltési folyamatát. A számítógépek általában rendelkeznek valamilyen belső programmal, ez adja át a vezérlést a rendszertöltést végigvivő programnak. A belső program védelmének tárgyalása túlmutatna a cikk témáján, ezért feltételezzük, hogy a rendszer belső programja megbízható. A legtöbb olvasó számára valószínűleg ismerős a ***BIOS***, a ***PC*** alapú gépek által használt rendszertöltő belső program. Itt az ***Open Firmware*** rendszertöltővel foglalkozok, ezt több gyártó is alkalmazza, például az ***Apple***, a ***Sun*** és az ***IBM***.

A *NetBSD/macppc* telepítési útmutatója kiváló ismertetőt tartalmaz az *Open Firmware*-hez. Célunk az, hogy az *Open Firmware* parancssoros felületének segítségével eltávolítható flash meghajtóról végzett rendszerindításra állítsuk be a számítógépet. Az *Open Firmware* lehetővé teszi a számítógéphez csatlakoztatott készülékek megtekintését, továbbá a belső program változóinak megtekintését és módosítását.
Az *Open Firmware* parancssorát a *New World* (*G3* vagy újabb) *Apple* gépeken az OPTION-COMMAND-O-F billentyűkombináció rendszertöltés közben való lenyomásával lehet elérni.
Azt, hogy a rendszer melyik eszközt használja a rendszertöltésre, a `boot-device` változó határozza meg.
A változó értékét a `printenv` paranccsal vizsgálhatjuk meg:

```
> printenv
[...]
boot-device      hd:,\\:txbi
➥hd:,\\:txbi
```

A fenti sor értelmezése a következő: *„a rendszerindítást az első IDE lemezen található, HFS típusú txbi fájl futtatásával kell elvégezni”*. A második kettőspont, mely a `txbi` előtt található, a zseton *HFS* fájltípusként való értelmezését váltja ki. Egyébként a program a `txbi`-t egy fájl elérési útjaként értelmezné. Esetemben a `hd` zseton valójában egy álnév a jóval bonyolultabb `/pci@f4000000/ata-6@d/disk@0` eszközhöz. A karakterlánc az első *IDE* merevlemez különféle alrendszereken keresztül vezető elérési útját adja meg. Azt, hogy egy álnév melyik eszközhöz tartozik, az *Open Firmware* `devalias` parancsával tekinthetjük meg.
A `boot-device`, a rendszerindító eszköz beállításához meg kell tudnunk, hogy az *Open Firmware* milyen névvel azonosítja flash meghajtónkat. Az ls paranccsal kapott eszközfát megvizsgálva azonnal fény derül a flash meghajtó elérési útvonalára:

```
> dev / ls
[...]
        /pci@f2000000
                [...]
                /usb@1b,1
                        [...]
                        /disk@1
[...]
```

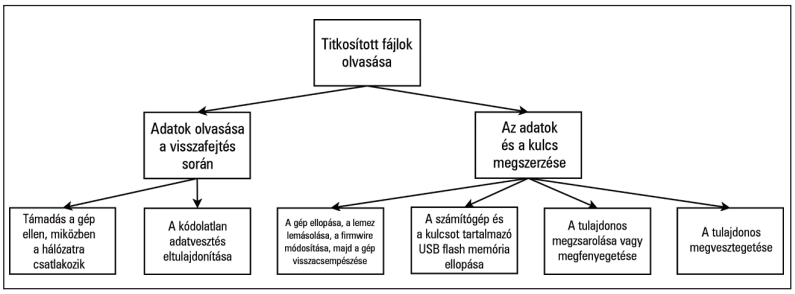


3. ábra Open Firmware-rel ellátott, PowerPC alapú rendszer indításának folyamata

Szereztünk némi betekintést a belső programnak a számítógépről alkotott képébe, ezért térjünk át a belső program által kezdésként futtatott program, vagyis a rendszertöltő vizsgálatára. Az *Apple PowerPC* gépein futó Linux rendszerek általában a yaboot nevű program közreműködésével indítják a gépet. A `yaboot` hasonló a *LILO*-hoz vagy a *GRUB*-hoz, és két kulcsprogramot tartalmaz, az *ofboot.b*-t és a *yabootot*. Az *ofboot.b* a rendszertöltés első fázisáról gondoskodik. Lényegében az *ofboot.b* feladata az indítandó operációs rendszer kiválasztása. Ha például a rendszerre a *Mac OS X* és a *Linux* is telepítve van, akkor az *ofboot.b* indítja el a *Mac OS X* vagy a *Linux* rendszertöltőjét. Ha a felhasználó a *Linux* betöltése mellett dönt, az *ofboot.b* a *yabootot* indítja el, mely a rendszertöltési folyamat második fázisát képviseli. A *yaboot* ezután betölti a *Linux* rendszermagot, és jelen esetben az *initrd*-t.
A *3. ábra* a *Linux* titkosított gyökér fájlrendszerről való elindítását szemlélteti (*PowerPC* géptípuson).
Eltávolítható rendszerindító eszközünkre az *ofboot.b* és a *yaboot* programot, egy *Linux* rendszermagot, valamint a titkosítási kulcsot tartalmazó *initrd*-t kell rámásolnunk. Az *Apple* jelenlegi *PowerPC* alapú gépei a rendszerindításra használt adathordozón *HFS* fájlrendszert várnak.

1) A `parted` programmal hozzuk létre a megfelelő, indításra is használható lemezrészt a flash meghajtón. (Saját meghajtóm 64 MB-os, és a */dev/sda* eszközcsomóponton keresztül érhető el.):

```
# parted /dev/sda
(parted) mklabel mac
(parted) print
Disk geometry for /dev/sda: 0.000-62.500 megabytes
Disk label type: mac
Minor   Start      End    Filesystem  Name    Flags
1       0.000      0.031       Apple
(parted) mkpart primary hfs 0.031 62.500
(parted) print
Disk geometry for /dev/sda: 0.000-62.500 megabytes
Disk label type: mac
Minor   Start      End    Filesystem  Name    Flags
1       0.000      0.03        Apple
2       0.031     62.500    untitled
```



2. ábra Hogyan tudja egy támadó olvasni a titkosított fájlrendszert?

```
(parted) set 2 boot on
(parted) name 2 Apple_Boot
(parted) quit
```

2) Hozzuk létre a ***HFS***-t a rendszertöltő lemezrészen:

```
# hformat /dev/sda2
```

3) A /mnt/encroot/etc/yaboot.conf módosításával állítsuk be a yabootot a megfelelő eszközről végzett rendszerindításra. Az alábbiakban egy minimális beállításkészlet szerepel:

```
boot=/dev/sda2
ofboot=/pci@f2000000/usb@1b,1/disk@1:2
partition=2
install=/usr/lib/yaboot/yaboot
magicboot=/usr/lib/yaboot/ofboot
default=linux
image=/vmlinux
        label=linux
        root=/dev/hda4
        initrd=/initrd.gz
        read-only
```

A `/pci@f2000000/usb@1b,1/disk@1:2` érték az ***Open Firmware*** eszközfájának korábbi vizsgálatából származik, a `/pci@f2000000/usb@1b,1/disk@1` pedig a `f2000000` címen elérhető ***PCI*** busz első ***USB*** buszának első lemeze. A bennünket érdeklő eszköz egy lemez (disk), a `:2` jelölés pedig a második lemezrészre utal.

4) Telepítsük a rendszertöltő programokat és a rendszermagot a */**dev**/**sda2***-re:

```
# ybin –config /mnt/encroot/etc/yaboot.conf -v
# mount /dev/sda2 /media/usbstick
# cp /boot/vmlinux /media/usbstick
```

A következő lépés a titkosításra képes initrd telepítése a flash meghajtóra. A Fedora tartalmaz egy mkinitrd nevű eszközt, mely alkalmas initrd létrehozására. Sajnos cikkem születésekor az mkinitrd még nem volt képes titkosított gyökérkönyvtárat befűzni. A ***https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=124789*** címen elérhető folttal viszont felülkerekedhetünk ezen a problémán is. A folt telepítése után az mkinitrd képes lesz a */**etc**/**crypttab*** olvasására, illetve a megfelelő ***initrd*** létrehozására:

1. `mkinitrd –authtype=paranoid -f /media/usbdisk/initrd.gz <rendszermagváltozat>`
2. `umount /media/usbstick`

Át kell írni a /mnt/encroot/etc/fstab fájlt is, követve a módosításokat:

```
/dev/mapper/root /      ext3    defaults      1 1
```

A cseretár titkosítása vagy teljes elhagyása a titkosított fájlrendszer használatának természetes velejárója. Ennek okai a „Titkosított saját könyvtárak megvalósítása” című cikkben és a ***BugTraq*** levelezési lista ***„Mac OS X stores login/Keychain/FileVault passwords on disk” (A Mac OS X lemezen tárolja a bejelentkezési/Keychain/FileVault jelszavakat)*** című beszélgetésében találhatók meg. Ha a ***https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=127378*** címről letölthető foltot feltesszük az ***initscripts*** csomagra, a ***Fedora*** lehetővé teszi a felhasználók számára, hogy véletlenszerűen létrehozott munkamenetkulccsokkal titkosítsák a cseretárként használt lemezrészt. Mivel általában a cseretárhelynek az egyes rendszerindítások alkalmával nem kell megőriznie tartalmát, a munkamenetkulcs a rendszer leállításakor nem kerül mentésre. A titkosított cseretár engedélyezéséhez a következő lépéseket kell végrehajtani:

1) A */**mnt**/**encroot**/**etc**/**fstab*** fájlhoz adjuk hozzá az alábbi sort, lecserélve vele a korábbi swap (cseretár) bejegyzést:

```
/dev/mapper/swap swap   swap    defaults      0 0
```

2) Adjuk hozzá az alábbi sort a */**mnt**/**encroot**/**etc**/**crypttab*** fájlhoz, ezzel közöljük a rendszerrel, hogyan kell elvégeznie a titkosítást:

```
swap    /dev/hda3       /dev/urandom    swap
```

Elértünk oda, hogy újra tudjuk indítani a rendszert, és használatba tudjuk venni a titkosított fájlrendszert. Ismét tartsuk lenyomva az OPTION-COMMAND-O-F billentyűkombinációt, és lépjünk be az ***Open Firmware*** parancssorába. Korábban már láttuk, hogy a flash meghajtó második lemezrészének elérési útja `/pci@f2000000/usb@1b,1/disk@1:2`. Ennek tudatában összeállíthatjuk a `/pci@f2000000/usb@1b,1/disk@1:2,\ofboot.b` elérési útvonalat. A vessző a lemezrész számát és a fájlrendszer elérési útját választja el egymástól. A `\ofboot.b` a fájlrendszerbéli elérési út, a \ pedig a ***UNIX*** / fájlrendszer gyökér jelöléséhez hasonlóan az eszköz gyökerét adja meg:

```
> dir /pci@f2000000/usb@1b,1/disk@1:2,\
Untitled            GMT                     File/Dir
   Size/      date       time   TYPE         Name
   bytes   9/ 3/ 4   21:44:41  ???? ????    initrd.gz
 2212815   8/28/ 4   12:24:21  tbxi UNIX    ofboot.b
    3060   9/ 3/ 4    2:21:20  ???? ????    vmlinux
```

```
  141868   9/28/ 4   12:24:22  boot UNIX  yaboot
     914   9/28/ 4   12:24:22  conf UNIX
yaboot.conf
```

Ezzel ellenőriztük, hogy az ***Open Firmware*** képes a rendszer elindításához szükséges fájlok olvasására. Ha a `boot-device` változót a `/pci@f2000000/usb@1b,1/disk@1:2,\ ofboot.b` értékre állítjuk, a rendszer a flash meghajtóról fog indulni:

```
 setenv boot-device
/pci@f2000000/usb@1b,1/disk@1:2,\ofboot.b.
```

Miután a gép sikeresen elindult a titkosított fájlrendszerről, semmisítsük meg a ***/dev/hda5*** által tárolt adatokat. Erre a célra a gyökér fájlrendszer lemezrészének véletlenszerűsítésénél már kipróbált módszert alkalmazzuk:

```
 dd if=/dev/urandom of=/dev/hda5
```

Ha gondoljuk, a felülírást többször is elvégezhetjük. A lemezek tartalmának kiirtására szabványt például az ***Amerikai Védelmi Minisztérium „National Industrial Security Program Operating Manual" (Nemzeti ipari biztonsági program üzemeltetési kézikönyve)*** dokumentumának 8. fejezetében találunk. Megfelelő törlés után a ***/dev/hda5*** például ***/home*** könyvtárként használható. A ***/home*** fájlrendszert szintén titkosítani kell. Szerencsére ennek folyamata jóval egyszerűbb, hiszen a gép nem a ***/home*** könyvtárból indul. Magát a fájlrendszert a gyökér fájlrendszerhez hasonló módon hozhatjuk létre.

1) Ellenőrizzük, hogy az ***aes***, a ***dm-mod*** és a ***dm-crypt*** modul be van-e töltve a rendszermagba.

2) Válasszuk le a ***/home*** elérési út alól a titkosított kezdőkönyvtár-fájlrendszer tárolására kiszemelt meghajtót, vagyis a ***/dev/hda5***-öt:

```
# umount /dev/hda5
```

3) Hozzunk létre egy véletlenszerű, 256 bites titkosító kulcsot, és mentsük el ***/etc/home-key*** néven. A szükséges parancs:

```
# dd if=/dev/urandom of=/etc/home-key bs=1c
count=32
```

4) Hozzunk létre egy ***dm-crypt*** eszközt, mely az imént összeállított kulccsal kerül titkosításra:

```
# cryptsetup -d /etc/home-key create home /dev/hda5
```

5) A ***/dev/mapper/home*** alatt hozzunk létre egy ***ext3*** fájlrendszert:

```
# mkfs.ext3 /dev/mapper/home
```

6) Fűzzük be az új fájlrendszert:

```
# mount /dev/mapper/home /home
```

7) Hozzunk létre egy bejegyzést a ***/etc/crypttab*** fájlban, a különféle segédprogramok innen olvashatják ki a fájlrendszer beállításait:

```
root    /dev/hda5       /etc/home-key    cipher=aes
```

8) Végül a ***/home*** könyvtárhoz tartozó bejegyzéssel bővítsük a ***/etc/fstab*** fájlt:

```
/dev/mapper/home /home   ext3    defaults      1 2
```

Szép munkát végeztünk, nekiláthatunk a további, nem root felhasználói fiókok hozzáadásához. A titkosított gyökér fájlrendszer létrehozása befejeződött.
Ha az összes adatunk titkosítva van, annak bizonyos veszélyei is lehetnek. Ha elveszítjük a titkosító kulcsot, vele együtt minden adatunk elvész. Éppen ezért ne feledjünk biztonsági másolatokat készíteni a kulcsot tartalmazó flash meghajtóról. Fontos továbbá, hogy a titkosított adatokról nyílt szövegű mentéseket is készítsünk. Ha rendszerindításra alkalmas vészlemezt is összeállítunk, jól gondoljuk át, hogy milyen összetevőket helyezünk el rajta. A gyökér és a kezdőkönyvtár-fájlrendszer kulcsára mindenképpen szükség lesz, de nem hiányozhat a `parted`, a `hfsutils`, a titkosítással kapcsolatos rendszermagmodulok és a `cryptsetup` sem.
Mennyire hatékonyan lehet ezzel a módszerrel megvédeni az adatokat? ***Secrets and Lies (Titkok és hazugságok)*** című könyvében ***Bruce Schneier*** ismertet egy olyan technikát, mellyel érdemi választ adhatunk erre a kérdésre. A fenyegetések modellezésére egy úgynevezett támadásfát lehet használni. A ***4. ábra*** titkosított fájlrendszerünk támadásfájának elejét tartalmazza. Mindenképpen ki kell emelni, hogy ez a támadásfa nem teljes, és talán soha nem is lesz az.
A cikkemben szereplő megoldást követve, némi kreativitással bárki erőteljes védelemmel láthatja el adatait bizonyos típusú lopások ellen. Nem szabad azonban elfeledkezni azokról a támadástípusokról, amelyek ellen ezek a védelmi megoldások hatástalanok. Nyilvánvaló, hogy a hálózati és egyéb támadásokkal szemben más módszerekkel kell védekezni, ám az itt szereplő fogások sokban hozzájárulnak a rendszer általános biztonságának megalapozásához.

Linux Journal 2005. január, 129. szám

Mike Petullo jelenleg tesztmérnök a WMS Gamingnél. 1997 óta ismeri a Linuxot. Bárki véleményét szívesen fogadja az lj@flyn.org címen.

KAPCSOLÓDÓ CÍMEK

- lwn.net/Articles/14776
- sourceforge.net/projects/loop-aes
- www.dss.mil/isec/nispom_0195.htm