Titkosított saját könyvtárak megvalósítása

Használjuk ki a visszacsatoló eszközön keresztül titkosított fájlrendszerek előnyeit és a pam_mountot adataink erősebb védelme érdekében.

Amennyiben megfelelően használjuk a titkosított fájlrendszereket, kitűnő eszközt kapunk, amellyel a számítógépünkön található érzékeny adatokat elrejthetjük a kíváncsi tekintetek elől. Az általános titkosító eszközök, például a GNU Privacy Guard (GPG), tökéletesek arra, hogy elektronikus leveleinket titkosítsuk. De a használatuk kényelmetlen olyan fájlok esetében, amelyeket nap mint nap gyakorta módosítani szeretnénk. A GPG-vel ellentétben a titkosított fájlrendszer a felhasználó számára teljesen láthatatlan, ugyanis nincs szükség a fájl használat előtti visszakódolására, se az ezt követő újbóli titkosításra. Ilyen módon a felhasználói feledékenység kérdése is megoldódik. Miután bemutatunk néhány, Linuxon használatos titkosított fájlrendszert, megláthatjuk, hogyan hozhatunk létre olyan titkosított saját könyvtárakat, amelyek a felhasználó belépésekor fűződnek be, kilépéskor pedig önműködően leoldódnak. Végül pedig kiderül, milyen hátulütői lehetnek annak, ha a különféle titkosítási módszereket nem megfelelően használjuk. Miért lehet szükség arra, hogy valaki titkosítsa a számítógépén található adatokat? Hiszen a fájlok jogosultságai is pont erre szolgálnak, nem? Mindamellett, hogy a felhasználói jogosultságok hasznosak, értelmüket veszítik olyankor, ha egy támadó sikeresen átveszi a rendszer és a hozzá tartozó tárolóeszközök feletti irányítást, ilyen módon kerülve meg a jogosultsági rendszert. Tegyük fel, ha valaki ellopja a Linux-rendszert futtató Apple iBook hordozható számítógépemet – ekkor a fájlrendszerbeli jogosultságok értelmetlenné válnak, mivel a tolvaj akár egy saját CD-ről is újraindíthatja a rendszert. Ugyanez áll fenn, ha a laptopomat visszaküldöm az Apple-höz javításra, és egy megbízhatatlan munkatárs szándékosan belenéz a fájljaimba. Emiatt a számítógépen található fájlok titkosítása megfelelő védelemnek tűnhet, mivel a titkosítási folyamat nem függ az operációs rendszer épségétől, fájljainkat a titkosítás védi.
Az iBookomon csak a saját könyvtárakat kódolom le. A teljes fájlrendszernek a gyökértől kezdődően történő titkosítása az én esetemben a fellépő teljesítménycsökkenés és egyéb okok miatt nem járható út. Amennyiben mégis egy ilyesfajta titkosítást szeretnénk megvalósítani, az interneten bőven találhatunk leírást e témáról; az eljárás a Linux rendszerindító memórialemez (ramdisk) képességeire épül. Az a tapasztalatom, hogy x86-os rendszer esetén az általam választott titkosítási módszer körülbelül ötven százalékkal lassabb, mint egy titkosítatlan XFS fájlrendszer, ha a lemezre íráskor átmeneti tárazott (puffered) bevitelt, illetve kimenetet alkalmazunk. Ha pusztán a saját könyvtárakat titkosítjuk, a rendszerben lévő egyéb fájlok, mint például a rendszernapló, egyszerű szövegként tárolódnak, de az én esetemben ez nem minősül igazán érzékeny adatnak. Számomra elfogadható megoldásnak tűnik csak a saját könyvtárak titkosítása.
A Linux kínál néhány lehetőséget a fájlrendszerek titkosításához. Az olyan rendszerek, mint a Transparent Cryptographic File System (TCFS, vagyis átlátszó titkosított fájlrendszer), egy titkosító bővítményt (plugin) ad az NFS által kiszolgált `ext2`-es kötetekhez. Némely fájlrendszer beépített titkosítási lehetőséget is tartalmaz helyi használatra. Esetemben a legjobb megoldásnak a visszacsatoló (loopback) eszközön keresztüli titkosítás mutatkozott. Mint látni fogjuk, a visszacsatoló eszközön keresztüli titkosítással bármilyen, Linux által támogatott fájlrendszert titkosíthatunk, legyen az `ext2`, ReiserFS vagy XFS. A visszacsatoló eszköz segítségével Linux alatt fájlokat fűzhetünk be a rendszerbe olyan módon, mintha a fájl egy eszköz lenne, például a */dev/hda1*. Ez a szolgáltatás arra használható, hogy például CD-re történő égetés előtt a kiírandó ISO fájlt befűzzük a rendszerbe, és ellenőrizzük, valóban működőképes-e, illetve a segítségével megtekinthető a hajlékonylemez-lenyomatok (floppy lemezkép) tartalma is. ***Herbert Valerio Riedel*** GNU/Linux CryptoAPI rendszermagfoltjával és `util-linux` foltjaival rendszerünket – a visszacsatoló eszközön keresztül – titkosított kötetek (lemezrészek) befűzésére tehetjük képessé. Mielőtt mélyebben beleásnánk magunkat a részletekbe, nézzük meg, hogyan fűzhetünk be köteteket a visszacsatoló eszközön keresztül az eredeti rendszermagba. Először is hozzunk létre egy fájlrendszert tartalmazó fájlt, azaz a példában szereplő állományt, ami elég nagy ahhoz, hogy egy 20 megabájtos fájlrendszer elférjen benne.

```
dd if=/dev/zero of=simaszoveg.img bs=1M
➥count=20
```

Ezt követően fűzzük be a fájlt a visszacsatoló eszközhöz:

```
losetup /dev/loop0 simaszoveg.img
```

Majd pedig hozzunk létre egy fájlrendszert a befűzött fájlban:

```
mkfs -t ext2 /dev/loop0
```

Végül fűzzük be a fájlrendszert, ahogy azt egy egyszerű eszköz esetében tennénk:

```
mount /dev/loop0 csatolási_pont
```

Most nézzük meg, hogyan működik mindez titkosított fájlrendszerek esetében. Ehhez azonban a rendszermagnak tartalmaznia kell a hozzá kapcsolódó bővítményt, de sajnos a legtöbb linuxos terjesztés ezt alapesetben nem támogatja, így egy saját építésű rendszermagra lehet szükségünk. Ezt a titkosító felületet a 2.4-es rendszermagokhoz a ➲ http://www.kerneli.org címen érhetjük el, a 2.5-ös rendszermagok viszont már eleve tartalmazzák ezt a bővítést. Ha sikerült feltenned a `patch-int` és `loop-hvr` foltot, a rendszermagod beállítópanelje egy ***Cryptographic options*** (Titkosítási beállítások) almenüvel bővül; ebben a következő beállításokat kell engedélyezned:

- cryptographic API support (`CONFIG_CRYPTO`)
- generic loop cryptographic filter (`CONFIG_CRYPTOLOOP`)
- cryptographic ciphers (`CONFIG_CIPHERS`)

A CONFIG_CIPHER_ részből legalább egy titkosítást engedélyezned kell. Én az AES-t választottam (ezt eredetileg Rijndaelnek hívták), és a példákban is ezt fogom használni.
Ezt követően fordítsd le a rendszermagot a kívánt beállításokkal. A titkosítási részek modulként is fordíthatók. Ha modulokat használsz, mielőtt ténylegesen használatba vennéd, ne felejtsd el betölteni őket. Ne felejtsd el az util-linux foltjait is feltenni, majd fordítsd és telepítsd fel. Az ehhez kapcsolódó util-linux folt a ➲ http://www.kernel.org/pub/linux/kernel/people/hvr/util-linux-patch-int címről tölthető le. Ha sikerül feltelepítened az új változatokat, tapasztalni fogod, hogy a mount és losetup parancs működése némileg megváltozik. Most már készen állunk arra, hogy titkosított fájlrendszer hozzunk létre, a korábbi példában látottakhoz hasonlóan, amikor egy egyszerű visszacsatoló eszközön keresztül fűztük be a fájlrendszert. Először is, hogy nehezen felismerhetővé tegyük a szabad és a foglalt területeket, az állomány tartalmát a */dev/urandom* segítségével véletlen adatokkal töltjük fel, a */dev/zero* helyett.

```
dd if=/dev/urandom of=ciphertext.img
➥bs=1M count=20
```

Miután létrehoztuk a gazdafájlt, először ideiglenesen hozzá kell kapcsolnunk egy visszacsatoló eszközhöz, csakúgy, mint az előbb. Ezúttal azonban meg kell mondanunk a losetup-nak, hogy az eszköz titkosítva legyen, esetünkben AES kódolással:

```
losetup -e aes /dev/loop0 ciphertext.img
```

Ha a losetup kérdezi, írd be a szükséges jelszót, és ha kéri – a kódolástól függően – a kulcsméretet, amellyel az adott kötet titkosítva lesz. A fájlrendszer létrehozása ugyanúgy történik, mint ahogyan az előző példában is láthattuk. A titkosítást már beállítottuk, és a visszacsatoló eszközre bíztuk, így azzal már nem kell foglalkoznunk:

```
mkfs -t ext2 /dev/loop0
```

A losetup módosításán kívül a util-linux folt a mount parancsot is felkészíti a titkosított kötetek kezelésére. Ezáltal a titkosított kötetek befűzése meglehetősen egyszerűen történik:

```
mount -o loop,encryption=aes ciphertext.img
➥csatlakoztatási pont
```

Ezután a mount parancs megkérdezi a jelszót, és ha szükséges, a kulcsméretet is.
Most pedig, hogy már magadtól is kezelni tudod a titkosított köteteket, rátérünk a pam_mount működésére. A pam_mount egy PAM-bővítmény, ami még jobban leegyszerűsíti a kötetek kezelését, és csak akkor fűzi be a szükséges kötetet, ha a felhasználó bejelentkezik a rendszerbe. Segítségével befűzhetjük a Samba vagy Novell alapú megosztásokat is, akár titkosított fájlrendszereket is. ***Elvis Pfützenreuter*** a pam_mount eredeti szerzője; ***Mukesh Agrawal*** pedig azt a foltot írta, amely először tette lehetővé a titkosított kötetek kezelését is. Jelen pillanatban e cikk szerzője a pam_mount karbantartója, mely bővítmény a ➲ http://www.flyn.org címen érhető el.
Ahelyett, hogy a titkosított köteteket magunknak kellene befűznünk, a rendszergazda a pam_mount-ot beállíthatja oly módon, hogy az magától fűzze be a fájlrendszert, ha a felhasználó bejelentkezik, illetve leoldja, amikor a felhasználó kilép a rendszerből.

Ez úgy állítható be, hogy a belépési jelszó egyúttal a titkosított kötetet is elérhetővé teszi, így hozva létre egy teljesen átlátszó titkosított fájlrendszert.
A pam_mount három különféle módszert használ arra, hogy hozzáférjen a titkosított kötetekhez. Az első elég unalmas. Ha a titkosított kötet jelszava nem áll kapcsolatban a belépési jelszóval, akkor a helyes jelszót a felhasználótól kéri be. E módszer használatához a ***pam_mount.so***-t és a pmhelper-t kell helyesen beállítani. A szokásos ./configure, make, és make install parancsok segítségével a pam_mount és a beállításokat tartalmazó fájlok a helyükre másolódnak. A gyári ***pam_mount.conf*** a */etc/security* könyvtárban található. Vess rá egy pillantást, és állítsd be a te rendszerednek megfelelően – a beállításfájlban található temérdek megjegyzésnek köszönhetően ez nem okozhat gondot. A legfontosabb változtatás, hogy a kötetek befűzéséhez kapcsolódó hivatkozásokat hozzáadjuk a fájl végéhez. Ahogyan a beállításokat tartalmazó fájlban látható, a titkosított köteteket kezelő beállítások körülbelül így festenek:

```
volume user local ignored
befűzendő_eszköz
csatlakoztatási_pont mount_kapcsolói
kódolás_típusa
kulcs_elérési_útja
```

Alább egy példát láthatunk arra, hogyan fűzzünk be egy AES-szel kódolt fájlrendszert a */home/mike* alá, amikor Mike bejelentkezik a rendszerbe:

```
volume mike local - /home/mike.img /home/mike
➥loop,user,exec,encryption=aes,keybits=256 - -
```

Ezt követően a megfelelő PAM-beállításfájlhoz adjuk hozzá a következő sorokat:

```
auth required pam_mount.so try_first_pass
session required pam_mount.so try_first_pass
```

Ez a beállításfájl ahhoz a szolgáltatáshoz tartozzon, amelyikhez a titkosított kötetkezelést szeretnéd kötni. Noteszgépemen a */etc/pam.d/login* fájl így néz ki:

```
auth      requisite  pam_securetty.so
auth      requisite  pam_nologin.so
auth      required   pam_env.so
auth      required   pam_unix.so nullok
account   required   pam_access.so
account   required   pam_unix.so
session   required   pam_unix.so
session   optional   pam_lastlog.so
session   optional   pam_motd.so
session   optional   pam_mail.so standard noenv
password  required   pam_unix.so nullok
                     ➥obscure min=4 max=8 md5
auth      required   pam_mount.so try_first_pass
session   required   pam_mount.so try_first_pass
```

Végül pedig hozd létre a felhasználóhoz tartozó titkosított kötetet, úgy, ahogy az előbb a példában láthattuk.
A kötetek befűzésére a második pam_mount módszer a felhasználó számára egy kicsit kényelmesebb. Ha ugyanúgy hozzuk létre a titkosított fájlrendszert, mint az előző módszer esetében, akkor a titkosított kötet jelszava megegyezhet a

felhasználó belépési jelszavával is, így a pam_mount ugyanezzel a jelszóval fér hozzá a kötethez.
A harmadik példa a legrugalmasabb, és egyúttal ez igényli a legrészletesebb magyarázatot. Először is tisztázzunk néhány fogalmat, hogy megértsük, pontosan hogyan működik ez a módszer:

- sk: a rendszerjelszó. Ez az a kulcs vagy jelszó, amellyel a felhasználó belép a rendszerbe.
- fsk: a fájlrendszer jelszava. Ez az a kulcs, amivel a pam_mount lehetővé teszi, hogy a titkosított kötetet befűzd.
- E és D: egy OpenSSL alapú kódoló, illetve dekódoló módszer, ilyen például a bf-ecb.
- efsk: a titkosított fájlrendszer kulcsa, efsk=E_sk(fsk), ami valahol a fájlrendszerben található (például */home/fhnév.key*).

A pam_mount beolvassa az efsk-t a fájlrendszerről, végrehajtja az fsk = D_sk(efsk) műveletet, és az fsk segítségével befűzi a fájlrendszert. Ennek a módszernek az az előnye, hogy olyan módon is megváltoztathatjuk a belépési jelszavunkat, hogy a titkosított fájlrendszert nem kell újból létrehoznunk. Ha a belépési jelszó megváltozik, egyszerűen csak újból létre kell hozni az efsk-t (vagy a */home/fhnév.key* fájlt) az efsk = E_newsk(D_oldsk(efsk)) segítségével – a pam_mount-ban található egy ***passwdehd*** nevű parancsfájl ennek kezelésére.
A harmadik módszernek a kivitelezéséhez először hozzunk létre egy fájlt, ami a fájlrendszert fogja tárolni (csakúgy, mint az előző példákban):

```
dd if=/dev/urandom of=/home/user.img bs=1M
➥count=méret_megabájtokban
```

Ezután hozzunk létre egy fájlt (efsk), ami a kötet jelszavát tartalmazza (fsk) a */dev/urandom* segítségével, ez a felhasználó bejelentkezési jelszavának megfelelően van kódolva:

```
dd if=/dev/urandom bs=1c count=keysize / 8
➥| openssl enc -bf-ecb >/home/user.key
```

Ezután hozzunk létre egy titkosított fájlrendszert a visszacsatoló eszközön. A fájlrendszer jelszava fsk legyen (ezt a */dev/urandom*-ból nyertük, és a */home/fhnév.key* fájlban a második lépésnek megfelelően titkosítva tároljuk):

```
openssl enc -d -bf-ecb -in /home/fhnév.key
➥| losetup  -e aes -k keysize -p0
➥/dev/loop0 /home/user.img
mkfs -t ext2 /dev/loop0
umount /dev/loop0
losetup -d /dev/loop0
```

Végül a ***pam_mount.conf***-ban állítsd be a kódolás típusát arra, amivel a kulcsfájlt titkosítottad, ez esetünkben a bf-ecb eljárás, majd a kulcs elérési útját állítsd be az efsk elérési útjára, ami esetünkben a */home/fhnév.key* állomány.
Bruce Schneier alapműnek számító könyvében, az Applied Cryptographyben azt állítja, hogy a „programalapú titkosítás ijesztő". Ez alatt azt érti, hogy rendkívül nehéz igazán megbízható titkosítási eljárásokat olyan általános célú operációs rendszereken létrehozni, mint amilyen a Linux is. Például az ilyen korszerű operációs rendszerek a memória tartalmát bármikor kiírhatják a lemezre, és így titkosított és titkosítatlan jelszavak egyaránt a lemezre kerülhetnek. Egy titkosított kötet lényegét veszti, ha a használatához szükséges kulcsot az operációs rendszer kiírja a lemezre. Egy lehetséges módszer ennek elkerülésére az, ha titkosítjuk a cserememéjának használt lemezrészünket. Ennek biztonságos végrehajtására még a CryptoAPI sem képes, de a fejlesztés már zajlik. Egy hasonló kezdeményezés – a LoopAES – azonban már képes titkosítani a rendszer csereterületeit.
Vegyük újra azt a példát, amelyikben a meghibásodott iBookomat elküldtem az Apple-nek javításra. Ebben az esetben bár a saját könyvtáram titkosítva van, az adataim még sincsenek teljes biztonságban. Egy megbízhatatlan munkatárs elindíthatja ördögi CD-ROM-ját a gépemen, és így kicserélheti mondjuk a login programomat az ő saját tervezésű változatára. Így ha visszakapom a noteszgépemet, és bejelentkezem rá, akkor az átjavított login program elküldheti a jelszavamat egy távoli számítógépre. Egy betörésérzékelő rendszer minden bizonnyal kiszúrná ezt a módosítást.
Egy további gyenge pontja a pam_mount-tal befűzött titkosított köteteknek a rendszer bejelentkezési jelszava. Mivel adatainkat a bejelentkezési jelszó segítségével titkosítjuk – akár közvetlenül, akár közvetve –, ennek a jelszónak erősnek kell lennie.
Ahelyett, hogy vakon növelni kezdenénk a szükséges jelszavak minimális hosszát, pillantsunk inkább bele Bruce Schneier „Titkok és hazugságok" című könyvébe. Az az erős jelszó, amit leírva egy pénztárcában tárolnak, biztonságosabb, mint egy fejben tartott, de könnyen kitalálható jelszó. Valamilyen fizikailag azonosító módszer alkalmazása is megfontolandó. Emlékezzünk, ha a rendszered jelszava nem biztonságos, akkor a titkosított fájlrendszered is csak annyit ér, mint a jelszó.
A titkosított jelszavak kétélű kardként is működhetnek. Mi történik, ha elfelejted a jelszavadat? Mi lesz, ha a harmadik megoldást használod, és véletlenül letörlöd a titkosított kulcsodhoz tartozó fájlt? Mi van akkor, ha a programom vagy valaki másnak a titkosító programja hibás? A felsoroltak mindegyike azt eredményezheti, hogy 2128-féle vagy még több különböző titkosított kulcsot kell kipróbálnod, hogy visszakapd a fájlrendszeredet. Az adataidról mindig gondoskodnod kell: állandóan készíts róluk mentést, akár egy jó rendszergazda. Alapesetben ezeket a mentéseket nem kell titkosítani, fizikailag azonban jól el kell zárni valamilyen biztonságos helyre.
Egy szó, mint száz, rengeteg erőfeszítésbe kerül, hogy egy valamennyire is biztonságos rendszert karbantartsunk, azon túl, hogy adataink biztonságát valamilyen korszerű titkosító eljárás szavatolja, például az AES. ***Matt Blaze*** a következőket fűzte hozzá az Applied Cryptography című könyvhöz:
„A kiváló minőségű titkosító eljárások és protokollok nagyon fontos eszközök, de magukban csak a valóság gyenge utánzatai – rendkívül fontos meggondolni azt, hogy mit védünk valójában, és jó tudni, hogy a védelmi rendszerünk hogyan válhat működésképtelenné (a behatolók ritkán tartják magukat a tiszta, jól meghatározott, tanított támadási modellekhez)."
Átrágtad magadat az írásomon, most már tehát tisztában kell lenned a titkosított fájlrendszerek működésével. A megfelelően titkosított saját könyvtárakkal ugyanis nagyon megbízható védelemhez jutunk.

Linux Journal 2003. augusztus, 112. szám

Mike Petullo (lj@flyn.org)
Az amerikai hadsereg Németországban állomásozó szakaszvezetője. Nappal repülőgépekből ugrál ki, éjszaka C-kódokkal hadakozik, és a Linuxot javítgatja 1997 óta.