A fájlrendszerek titkosítása

Avagy hogyan rejtsünk el egész fájlrendszereket a kíváncsi szemek elől?

Számítógépünk használatba vételének pillanatától kezdve a lassan csordogáló idő folyamán rengeteg olyan adat gyűlik össze, amelyeket nem szívesen osztanánk meg másokkal. Gondolok ez alatt akár a személyes vagy üzleti levelezésünkre, üzleti kimutatásainkra, jelszavakra vagy bármilyen olyan adatra, melynek idegen kézbe kerülése kellemetlen perceket okozhatna számunkra. Néhányan abban a tévhitben élnek, hogy a gondosan megválasztott rendszergazdai jelszó elegendő biztonságot nyújt – őket, sajnos ki kell ábrándítanom. Adatainkat merevlemezünkön kódolatlanul tároljuk, amennyiben valaki megkaparintja, az teljes tartalmával, azaz összes adatunkkal együtt a birtokába kerül. Ilyenkor már nem véd meg sem a tűzfal, sem a rendszergazdai jelszó – kiszolgáltatottak leszünk.

Gondoltál már arra, hogy kódolni kellene azt a sok-sok adatot, hogy csak te férhess hozzá? Netán a jelszavaidat felejted el mindig? Mit szólnál hozzá, ha létezne egy megoldás, amelyhez mindössze egyetlen jelszót kell megjegyezned, és minden adatod biztonságban tudhatnád?

Létezik ilyen megoldás! Fájlrendszerszintű titkosításnak nevezik, és egész sor program létezik hozzá. Mi most kettőt fogunk alaposabban megvizsgálni: a CryptoAPI-t és a CFS-t. Az előbbi megoldás az úgynevezett ***international kernel patch***-ből nőtte ki magát, az utóbbi pedig egy rendszermag-támogatást közvetlenül nem igénylő, NFS-re épülő kiszolgáló. Mindkettő transzparens, azaz teljesen átjárható támogatást nyújt. Csupán használatuk megkezdésekor, a gép indítása után kell megadni egyetlen jelszót, akkortól kezdve a fájlrendszerben a csatolási pontjaik alá írt fájlok kódolva tárolódnak a lemezen, és csupán akkor fejtődnek vissza, ha valamilyen program szeretne hozzájuk férni.

Mielőtt belekezdenénk, egy fontos dologra még felhívnám a figyelmedet: ebben a titkosításban sem lehet vakon megbízni. A rendszeredre ezután is ügyelned kell! Amennyiben valaki, tegyük fel, betör a gépedre az adataidhoz ugyan nem fog tudni jelszó nélkül hozzáférni, viszont megfelelő jogosultságok megszerzésével a legközelebbi alkalommal, amikor beírod a jelszavadat, a betörő ellophatja tőled! Innentől kezdve pedig az összes adatod ismét szabad préda. Tökéletes biztonság tehát nem létezik, de minél jobban megpróbáljuk megközelíteni, annál jobban sikerül leszűkítenünk azoknak a körét, akik sikerrel pályázhatnának adatainkra.

A CryptoAPI

Ez a megoldás annak idején ***international kernel patch***-ként vált ismertté, és támogatottsága időről időre változik, szerencsére mostanában fejlesztője újra felkarolta a projektet. Jelenleg az összes friss változatú rendszermaghoz létezik támogatás, beleértve a 2.5.x-es fejlesztői sorozatot is. Az újabb CryptoAPI-k rendszermag-újrafordítást igényelnek, mivel használatukhoz egy új hurok (loop) eszköz szükségeltetik: a ***cryptoloop***.

A CryptoAPI alkalmazásával adatainkat különböző kódolási eljárásokkal titkosíthatjuk, például a Blowfish-, IDEA-, Serpent-, DES-, 3DES-eljárások segítségével.

A CryptoAPI telepítése

Első lépésben töltsük le a legújabb rendszermag forráskódját és a hozzá kapcsolódó CryptoAPI-foltot (lásd a ***Kapcsolódó címek*** részt). Miután a rendszermagot kibontottuk, lépjünk be a könyvtárába, és másoljuk oda a CryptoAPI-hoz tartozó foltot. Amennyiben a rendszermag bzip2-vel van tömörítve:

```
# tar Ixf kernelforrÆs.tar.bz2 -C /usr/src
```

Ha pedig gzippel:

```
# tar zxf kernelforrÆs.tar.gz -C /usr/src
```

Mindkét parancs létrehoz egy ***linux*** könyvtárat a ***/usr/src***-n belül, alatta a rendszermag forráskódjával. Ezt követően lépjünk be a ***/usr/src/linux*** könyvtárba és telepítsük a CryptoAPI-t a rendszermag forrására:

```
# cd /usr/src
```

2.4.16-os vagy újabb rendszermag esetén:

```
# zcat ~/patch-int-2.4.16.x.gz | patch -p1
```

Régebbi rendszermag esetén pedig:

```
# zcat ~/cryptoapi-2.4.x.x.diff.gz | patch -p1
```

2.4.16-os vagy újabb rendszermag esetén egy másik folt is szükséges:

```
# cat ~/loop-hvr-2.4.16.0.patch | patch -p1
```

(Ha a folt gzip helyett bzip2-vel lett tömörítve, `zcat` helyett a `bzcat` használható.)

Amennyiben a fenti parancsokat sikerrel végrehajtottuk, egy CryptoAPI-val felvértezett rendszermagforrást kapunk, ezt pedig csak le kell fordítanunk. A ***/usr/src/linux*** könyvtárban állva adjuk ki a `make menuconfig` parancsot. Ha a parancs hibát jelez, elképzelhető, hogy a futásához szükséges valamelyik csomag nincs feltelepítve, ez azonban a legtöbb esetben a ***libc6*** vagy az ***ncurses*** fejlesztői könyvtárainak a hiányát jelenti (***-dev***-csomagok). Amennyiben a parancs sikeresen lefut, egy menüben találjuk magunk. Itt lehet megadni, hogy rendszermagunkba mely összetevők forduljanak bele. Minket jelen esetben csak a CryptoAPI-val foglalkozó rész érdekel, így lépjünk be a ***Cryptographic options*** (régebbi változatú CryptoAPI esetén csak ***Crypto options***) menübe. Alapértelmezésben az összes elem modulként van megjelölve, ezeken szükségtelen változtatni. Ha a rendszermagunk egyéb tulajdonságait is beállítottuk, lépjünk ki ebből a menüből és folytassuk a fordítást. Futtassuk le a következő parancsokat:

```
# make dep && make clean && make install
# make modules && make modules_install
```

Ha minden jól ment egy új, működő titkosítási képességekkel bíró rendszermagot kaptunk, melyet a fentebbi `make install` lefuttatásával már működőképes állapotba is hoztunk. Nincs más dolgunk, mint újraindítani a rendszert és behúzni az új rendszermagot.

A CryptoAPI használata

A CryptoAPI használatához szükséges a `losetup` parancs, mely lehetővé teszi a titkosított állományok befűzését. Futtassuk le a `losetup` programot kapcsolók használata nélkül, és ha a megjelenő szövegben nem tesz említést a `-e` vagy a `-encryption` kapcsolókról, minden bizonnyal egy új util-linux csomagot kell beszereznünk, mely a `losetup` egy újabb változatát is tartalmazni fogja, ami remélhetőleg már képes lesz titkosított kötetek kezelésére.
Most már csak azt kell kiválasztani, hogy milyen kódolási eljárást alkalmazzunk. Az egyik legnépszerűbb és legelterjedtebb titkosítás a ***Blowfish***. Ha már korábban is foglalkoztál titkosítással, biztosan ismerősen cseng ***Bruce Schneier*** neve, ő a Blowfish megalkotója és tollából született az alapműnek számító Alkalmazott kriptográfia (Applied Cryptography) című könyv. Alapértelmezésben a CryptoAPI úgynevezett CBC-módban kódolja a befűzött fájlokat. Ez a CBC-mód a ***Cipher Block Chaining***-et takarja, ez azt jelenti, hogy a kódolás során minden blokkot az előző blokk kódolt értékével is kódol. Emiatt később, ha egy blokk, vagyis láncszem elvész vagy megsérül, az azt követő összes többi blokk, azaz a lánc további része is használhatatlanná válik. Ennek köszönhető az is, hogy ha van egy hosszabb szövegünk vagy karaktersorozatunk, amely blokkról blokkra ugyanabból a szövegből áll minden egyes blokk másképpen kódolódik. Ha CBC helyett ECB-t (Electronic Code Book) használunk, a megegyező blokkok ugyanúgy kódolódnának, mivel az ECB minden egyes blokkot külön kódol. Ennek az a hátránya – vagy előnye, attól függően, mire szeretnénk használni –, hogyha a lánc egyik blokkja megsérül, az még nem feltétlenül vonja maga után az egész szöveg használhatatlanná válását.
Térjünk vissza azonban a CBC-hez! A CBC sem tökéletes, ha ugyanis nagy mennyiségű adatot kódolunk, akkor az adat hosszával exponenciálisan nő annak az esélye, hogy ugyanúgy kinéző kódolt blokkok keletkeznek. Minél több ilyen megegyező kódolt blokk jön létre, annál könnyebben törhetővé válik kódolásunk. Emiatt például 64-bites blokkhosszúságú kódolásnál, mint amilyen a Blowfish is, nem ajánlatos átlépni a 2-3 gigabájtos határt. Ha nagyobb mennyiségű adatot szeretnénk kódolni, ajánlatos valamilyen 128-bites blokkhosszúságú kódolást választani, amellyel ennél jóval nagyobb köteteket is biztonsággal kódolhatunk. A ***Serpent*** egy 128-bites blokkhosszúságú kódolási eljárás, melyet az egyik leggyorsabbnak és legkevésbé processzort megterhelőnek tekintenek, így példáinkban ezt fogjuk használni.
Első lépésben töltsük be a titkosítási kötetek kezelését végző modult:

```
# modprobe cryptoloop
```

Erre a modulra mindig szükségünk lesz, ha titkosított kötetekkel szeretnénk dolgozni.
Ezt követően hozzunk létre egy 10 megabájtos fájlt „kodolt” néven, amelyben majd kódolt fájlrendszerünk fog helyet foglalni:

```
# dd if=/dev/zero of=/root/kodolt bs=1M
➥count=10
```

Fűzzük be a fájlt a hurok eszközre:

```
# losetup -e serpent /dev/loop0 /root/kodolt
```

Ezt követően a parancs végrehajtása során rákérdez, hogy milyen hosszúságú kulccsal szeretnénk dolgozni. Válasszuk ki a 128-at, bőven elegendő lesz. Jelszónak pedig adjunk meg egy tetszőleges karaktersorozatot.
A következő lépésben hozzunk létre egy fájlrendszert az eszközön:

```
# mke2fs /dev/loop0
```

Természetesen nemcsak `ext2`-t használhatunk, hanem bármilyen más egyéb, Linux által támogatott fájlrendszert is.
Ha az `mke2fs` elkészült, fűzzük be az eszközt a */**mnt*** alá:

```
# mount /dev/loop0 /mnt
```

Ha belépünk a */**mnt*** könyvtárba láthatjuk, hogy máris ott van egy ***lost+found*** könyvtár, melyet az `mke2fs` hozott létre. Készítsünk egy fájlt kedvenc szövegszerkesztőnkkel és írjunk bele néhány sort, majd mentsük a fájlt és miután kiléptünk a könyvtárból, fűzzük ki a kötetet:

```
# cd /
# umount /dev/loop0
# losetup -d /dev/loop0
```

Ha a `grep`-pel rákeresünk az előbbi szövegre a fájlban, azt tapasztalhatjuk, hogy a fájl nem tartalmaz ilyen szöveget. Ellenben ha a */**dev**/**loop0***-n grepelünk a fájl befűzött állapotában, akkor természetesen a `grep` találatot jelez:

```
# grep -c  sz veg  /root/kodolt /dev/loop0
```

Következő alkalommal, amikor szeretnénk újra hozzáférni a kötethez, csak a `losetup` parancsot kell kiadni ugyanúgy, mint az előbb:

```
#  losetup -e serpent /dev/loop0 /root/kodolt
```

Figyeljünk arra, hogy kulcsnak és jelszónak ugyanazt adjuk meg, mint az első használatnál, máskülönben a kötet rosszul kódolódik és a `mount` paranccsal sem leszünk képesek befűzni egészen addig, míg a kötetet nem fűzzük újra a helyes jelszóval.

Munka titkosított kötetekkel

Ha rendszeresen erre a titkosított fájlrendszerre szeretnéd menteni a dolgaid, a */**mnt*** helyett létrehozhatsz neki más könyvtárat is, illetve a */**etc**/**fstab*** fájlban beállítható néhány alapértelmezés, hogy kevesebbet kelljen gépelni az állandó használat során.
Nézzük meg, miként fest egy általános ***fstab***-bejegyzés kódolt fájlrendszerre:

```
/root/kodolt  /mnt  ext2  defaults,noauto,
➥loop=/dev/loop0,encryption=serpent  0  0
```

Az első oszlopban adjuk meg, hogy hol található a kódolt fájlrendszert tartalmazó fájl, a második oszlopban azt adjuk meg, hogy melyik könyvtárba fűződjön be, az `ext2` pedig értelemszerűen a fájlrendszer típusát jelenti. A `noauto` azt jelenti, hogy a rendszer ne fűzze be önmagától a fájlt rendszer-

induláskor. A `loop` kapcsolóval a hurkoló eszközt választjuk ki, az `encryption`-nel pedig a kódolás típusát. A következő két érték közül a második `0` azt jelenti, hogy rendszerinduláskor a fájlrendszert nem kell ellenőrizni.
Ha ezzel megvagyunk, a következő két paranccsal minden beállítást elvégezhetünk:

```
# mount /root/kodolt
# umount /root/kodolt
```

Ilyen esetekben a kötet jelszavát a `mount` parancs fogja kérni. Főleg nagyobb fájlrendszer esetén nem árt, ha időnként lefuttatunk valamilyen fájlrendszer-ellenőrző programot a titkosított kötetre is. Először fűzzük be a kötetet a `losetup` parancscsal a már tanult módon, majd ha `ext2`-t használunk, futtassuk le a következőt:

```
# e2fsck -f /dev/loop0
```

Ismerkedés a CFS-sel

A CFS egészen más elven működik, mint a CryptoAPI. Annyiban hasonlítanak csupán, hogy egyikük sem köt minket egy bizonyos fájlrendszer használatához. Ráadásul a CFS képes az éppen működő fájlrendszeren dolgozni, és egyszerű, felhasználó is tudja kezelni. A rendszer lényege, hogy a saját könyvtárunkban létrehoz egy könyvtárat, amelybe a titkosított fájlok kerülnek. Ebbe a könyvtárba nem szabad közvetlenül írni, helyette a CFS által létrehozott könyvtárba kapunk írási jogot, mely valójában egy NFS kötet, és a saját könyvtárunkon kívül található.

A CFS használa

A CFS-t a ***Kapcsolódó címek*** részben található címről tölthetjük le, vagy ha Debian Linuxot használunk, az `apt-get install cfs` paranccsal is feltelepíthetjük. Telepítés után azonnal megkezdhetjük a munkát, akár egyszerű felhasználóként. Először hozzunk létre egy könyvtárat a CFS-sel, ezt a `cmkdir` paranccsal tehetjük meg:

```
# cd ~
# cmkdir proba
```

Ezután a rendszer megkérdezi, hogy milyen jelszót szeretnénk a könyvtárhoz rendelni. A jelszónak legalább 16 karakteresnek kell lennie és nem árt, ha alaposan az emlékezetünkbe vésük. A létrejövő „proba” könyvtárban egy sor különös ponttal kezdődő bejegyzést találunk, melyet a CFS hozott létre saját belső használatra. Ezzel a könyvtárral a továbbiakban nem is kell törődnünk, végképp nem szabad fájlokat létrehozni benne, mert azok ilyen módon nem kódolódnak le. Helyette, ha mégis szeretnénk használni, fűzzük be a könyvtárat a `cattach` paranccsal:

```
# cattach proba
```

Ezt követően, ha belépünk a */**crypt**/**proba*** könyvtárba (Debian esetén */**var**/**cfs**/**proba***), az ott létrejövő fájlokat a rendszer kódolja, és titkosított névvel a saját könyvtárunkban lévő „proba” könyvtárba menti. Fontos tehát, hogy nem a saját könyvtárunkban lévő könyvtárban, hanem a */**crypt*** (illetve */**var**/**cfs***) alatt lévő könyvtárban kell dolgoznunk.
Ha végeztünk a munkával a könyvtárat a `cdetach` paranccsal fűzhetjük ki:

```
# cdetach proba
```

Már létező könyvtár jelszavát a `cpasswd` paranccsal lehet megváltoztatni, amely példánkban így működne:

```
# cpasswd proba
```

A CFS-ről bővebben

A CFS a kódoláshoz különleges DES-t vagy 3DES-t használ ECB-vel. Mivel a CFS az állományok visszafejtéséhez felhasználja a fájlleírójának (i-node) számát és a felhasználó UID-ját is, ezért a befűzött könyvtárakhoz csak az a felhasználó férhet hozzá, aki eredetileg létrehozta azokat, illetve visszafejtés nélkül a fájlokat másolni sem lehet, mivel az új fájloknak más lesz a fájl leírószáma és ezáltal megfejthetetlenné válnak. Ez azt jelenti, hogy még a könyvtár befűzött állapotában sem fér hozzá senki a fájljainkhoz, még a rendszergazdát sem. Ez nem egy életbiztosítás, mivel mint tudjuk, a rendszergazda jelszó nélkül válhat tetszőleges felhasználóvá, úgy pedig már akadály nélkül beletekinthet bármibe, ha a könyvtár éppen be van fűzve. Nem árt tudni azt sem, hogy a `cdetach` parancs nem tartalmaz semmilyen védelmet, így bárki leválaszthat bármely más felhasználó által használt könyvtárat.

Összegzés

Az eddigiekben a CryptoAPI-val és a CFS-sel ismerkedhettünk meg. Mindkettőnek egyaránt megvannak az előnyei és a hátrányai, ki-ki maga döntse el, melyik felel meg jobban az ő igényeinek. Ha nem elégedtél meg ennyivel, elárulom, létezik még néhány választható megoldás, mint például a BestCrypt vagy a LoopAES, melyekkel lehet, hogy a későbbiekben még foglalkozunk.

Gludovátz Gábor
(ggabor@sopron.hu)
1996 óta foglalkozik Linux-rendszerekkel. Egyik kedvenc időtöltése a programozás, jelenleg éppen egy C++-ban írt KDE-s játékon dolgozik, de szívesen kódol Pythonban és PHP-ben is. Honlapja ➲ http://www.sopron.hu/~ggabor/

Kapcsolódó címek

A CryptoAPI a 2.4.16-nál régebbi rendszermagokhoz
➲ ftp://ftp.hu.kernel.org/pub/linux/kerne/people/hvr/
A frissítésekhez csak próbaváltozatú CryptoAPI van
➲ ftp://ftp.hu.kernel.org/pub/linux/kerne/people/hvr/testing/
IV-számoló (initialization vector) eljárás javítása, mely a 2.4.16 vagy újabb rendszermagokhoz szükséges
➲ ftp://ftp.hu.kernel.org/pub/linux/kerne/people/hvr/testing/loop-hvr-2.4.16.0.patch
A CFS-t innen töltheted le
➲ http://computing.ee.ethz.ch/sepp/cfs-1.4b2-to.SEPP/
Linux-Crypto levelezési lista
➲ http://mail.nl.linux.org/linux-crypto/
Cryptography-FAQ
➲ http://www.faqs.org/faqs/cryptography-faq/
A *Handbook of Applied Cryptography* című könyv innen tölthető le ➲ http://cacr.math.uwaterloo.ca/hac/