A Linux-rendszermag titkosító API-ja

Új általános keretrendszerben érhetők el a titkosító szolgáltatások a rendszermag minden része számára.

Írásunkban a rendszermag új titkosító API-ját mutatjuk be röviden. Mindenkihez szólunk, akit izgat a Linux működése, például rendszergazda vagy olyan érdeklődő személy, aki szeretne bepillantani az API tervezésébe, megvalósításába és alkalmazásaiba.
A rendszermag működésének ismerete előnyt jelent, de ez nem szükséges ahhoz, hogy az API-t nagy vonalakban megértsük.
Ez az API csak rövid ideje létezik. Nem sokkal azután, hogy 2002 őszén a rendszermagot lezárták, vagyis az új szolgáltatásokat megvalósító kód beépítése nem volt lehetséges, a ***Dave Miller*** és ***Alekszej Kuznyecov*** által fejlesztett IPSec-megvalósítást jelölték a 2.6-os rendszermagba való beépítésre. Az IPSec a rendszermagon belül igényli a titkosítási szolgáltatásokat, és ez a rendszermag titkosító szolgáltatásaira való egyre növekvő általános igény mellett arra késztette a fejlesztőket, hogy új titkosító API-t fejlesszenek.

Tervezés

Bár eredetileg az IPSec támogatása volt a cél, az API-t általános célú szolgáltatásnak tervezték, amit a jövőbeli alkalmazások is kihasználhatnak, például az állománytitkosítók, a titkosított állományrendszerek, az állományrendszerek sértetlenségét felügyelő alkalmazások, a véletlenszám-előállító eszköz (***/dev/random***), a hálózati állományrendszerek biztonsága (például a CIFS) és egyéb hálózati szolgáltatások a rendszermagban, amelyek igénylik a titkosítást.
Az API tervezésekor kifejezetten szempont volt, hogy közvetlenül a lapvektorokon működjön. A lap a rendszermag által kezelt memória alapegysége. A lapvektorokon alapuló API lehetővé teszi a mély beépülést a rendszermag alszerkezeteibe, például a VFS-be vagy a hálózati verembe, valamint a szétszórás-összegyűjtés típusú műveletet is. Az IPSec esetében a titkosítási műveletek közvetlenül a hálózati csomagoknak megfeleltetett nem folytonos memórialapokra alkalmazhatók.
Az egyszerűség fontos tervezési szempont volt, ami általában is mindig jó, de különösen a rendszermag és a biztonsági szolgáltatások kódjánál lényeges.
A rugalmas használhatóság is a célkitűzések között szerepelt. Például az API rugalmasan kezeli az algoritmusokat: ezek magmodulként szükség szerint betölthetők, az API-nak semmit nem kell róluk előzetesen tudnia.
A jövőben a következőket szeretnék még megvalósítani:

- Titkosításgyorsító kártyák és az IPSec-es hálózati kártyák képességeinek a kihasználása.
- A felhasználó által előnyben részesített algoritmus kiválasztása, ha több lehetséges megvalósítás közül lehet választani (például hatékony gépi kódú megvalósítások vagy különböző alkatrészszintű megvalósítások).
- Az aszimmetrikus titkosítás (RSA) támogatása, amire a csoportszórásos IPSec és a magmodulok aláírása ellenőrzésének rendszermagbeli támogatásához lehet szükség. Ezen a területen sok vita várható, mert az aszimmetrikus titkosítás általában lassú és bonyolult – bármelyik a kettő közül elég okot ad arra, hogy kimaradjon a rendszermagból.
- Egységes API a felhasználói programok számára, amik az elérhető titkosítóeszközöket kívánják hasznosítani, például SSL, IPSec-kulcscsere, biztonságos útválasztó protokoll és DNSSEC.
- Az API memóriafoglalásának további csökkentése, hogy beágyazott eszközökben is használható legyen.

Algoritmusok

Jelenleg az API az alábbi háromféle algoritmust támogatja:

1. ***Kivonatolók*** (egyirányú hasítófüggvények) – ezek bemenete tetszőleges üzenet, kimenete rövid, adott méretű üzenetkivonat. Az egyirányúsághoz az szükséges, hogy a kimenet egyszerűen előállítható legyen, de az eredeti üzenetet ne lehessen könnyen visszaállítani belőle. A titkosítási célokra olyan hasítófüggvények felelnek meg, amik különböző bemenetekre ritkán adnak azonos kimenetet. A lehetséges alkalmazási területek között van az adatok sértetlenségének ellenőrzése és üzenethitelesítő kódok létrehozása hálózati protokollokhoz. Kivonatoló algoritmusokra példa az MD5 és az SHA1.
A HMAC nevű üzenethitelesítő séma (RFC2104) beépült az API-ba, és ez minden szabványos kivonatoló algoritmussal működik. Jelenleg az IPSec-csomagok hitelesítési adatainak előállítására használják.
2. ***Titkosítók*** – ezek az algoritmusok szimmetrikus kulcsú titkosítást valósítanak meg, ahol az üzenetet egy kulcs segítségével alakítják át titkosított üzenetté. Általában ugyanez a kulcs szolgál a titkosított üzenet visszafejtésére is. Követelménye, hogy az üzenetek titkosítása és visszafejtése a kulcs birtoklása esetén könnyű legyen (amit viszont titokban kell tartani), anélkül viszont nehéz. A lehetséges alkalmazási területek között szerepel az adattitkosítás és az üzenethitelesítő kódok létrehozása. Titkosító algoritmusokra példa a TripleDES, és a Blowfish, valamint az AES.
Kétféle titkosító létezik: a tömbtitkosítók adott hosszúságú adattömbökkel dolgoznak (például 16 bájt), a folyamtitkosítók kulcsfolyamot használnak, ami az adat egyetlen bitjén dolgozik egy lépésben. A titkosítók számos üzemmódban működhetnek, például ECB (Electronic Codebook) módban, amiben az üzenet minden tömbje egyszerűen van titkosítva a kulccsal, vagy CBC (Cipher Block Chaining) módban, amiben az előzőleg titkosított tömb a következő tömb titkosításában kerül felhasználásra.
3. ***Tömörítés*** – gyakran használják együtt a titkosítással, hogy nehezebb legyen kihasználni az eredeti üzenet gyengeségeit, és hogy gyorsabb legyen a titkosítás (ugyanis a tömörített üzenetek rövidebbek). A titkosított adatot lényegéből adódóan nem nagyon lehet tömöríteni, és ez hátrányosan befolyásolja a teljesítményt az olyan átviteli csatornák esetében, amelyek kihasználják a tömörítést. Ha az ada-

tokat még a titkosítás előtt tömörítik, sok esetben elkerülhető a teljesítménycsökkenés. Tömörítő algoritmusokra példa az LZS és a Deflate.
Eddig a jól ismert forrásokból származó algoritmusokat tették alkalmassá az API-val való használatra, mivel ezeket valószínűleg jobban átnézték és többen próbálták ki. A rendszermag főágába általában csak olyan algoritmus kerülhet be, ami nem áll szabadalmi oltalom alatt (azaz az IDEA 2011-ig nem kerülhet be), nyílt és elismert szabványokon alapul, és kipróbálására tesztcsomag áll rendelkezésre.

Lapvektorok

Mielőtt az API szerkezetének tárgyalásába kezdenénk, röviden tekintsük át a memórialapok és a lapvektorok témakörét. Mint az fentebb említettük, a lap a rendszermag által kezelt memória alapegysége (i386-on a lapok mérete 4 KB). Képzeljünk el egy tárolót, amiben mondjuk 1460 bájt felhasználói adat van. Ez a rendszermag adott lapjához tartozik, a lap kezdetétől számított bizonyos eltolási értéktől kezdődik, és 1460 bájt hosszú. Ezt a tárolót a következő lapalapú szerkezet írja le: `{ lap, eltolÆs, hossz }`.
A lapokkal közvetlenül dolgozó csatolónak, például a titkosító API-nak ezzel a szerkezettel, más néven a lapvektorral kell foglalkoznia. A megvalósítás egy már meglévő rendszermagbeli adatszerkezetet, az úgynevezett szórási listát használja. A szórási listában lapvektorok vannak, és általában szétszórásos–összegyűjtéses közvetlen memória-hozzáférési műveleteknél használatos.
A titkosító API arra használja a szórási listát, hogy nem folytonos lapvektorokon végezzen műveleteket. A rendszermagban a szétszórás–összegyűjtés elsődleges célja az adatok felesleges másolásának elkerülése. A tapasztalat szerint ettől a forráskód is áttekinthetőbb lesz. Számos olvasónak ismerős lehet a szétszórásos–összegyűjtéses I/O a `readv()` és a `writev()` rendszerhívások formájában. A rendszermag titkosító API-ja ugyanezt az általános elvet használja fel, de egyszerű memóriaterületek helyett lapokon működik.

Az API szerkezete

Az API két elsődleges objektummal foglalkozik:

- ***Algoritmusmegvalósítások*** – magmodulok, amik a szükséges algoritmusok kódját tartalmazzák.
- ***Átalakítók*** – objektumok, amik példányosítják az algoritmusokat, kezelik a belső állapotot és a közös megvalósítási logikát. Az átalakítókat a `crypto_alloc_tfm()` és a `crypto_free_tfm()` segítségével kezelhetjük. Az API-hoz tartozó burkolók segítségével leegyszerűsödik az átalakítók használata, és lehetővé válik az átalakító alatt meghúzódó algoritmus tulajdonságainak a lekérdezése.

A következő példakód az átalakító jellemző használatát mutatja be, ez a Blowfish-kódolóval ECB módban bizonyos rendszermagkód adatokat szeretne titkosítani:

```
tfm =
➥crypto_alloc_tfm("blowfish",
CRYPTO_TFM_MODE_ECB);
crypto_cipher_setkey(tfm,
➥key, keylength);
crypto_cipher_encrypt(tfm,
➥&scatterlist, numlists);
crypto_free_tfm(tfm);
```

Az ábrán (48. CD Magazin/Crypt könyvtár) látható, hogy az API réteges felépítésű, azaz a mag a titkosítás felhasználói és az algoritmusok megvalósítói elől rejtve van. Ez a mag tartalmazza az általános átalakítókezelést, a szórási lista kezelését és az algoritmusok elvonatkoztatását. Még lejjebb az algoritmustípus szerinti logika – például a titkosítók feldolgozási üzemmódja és a kivonatok használata az üzenethitelesítő kódok előállítására – található.
Az algoritmuskezelő réteg tartalmazza az algoritmusok megvalósításait felkutató, betöltő és a rájuk való hivatkozást számláló logikát. Az utóbbi azért szükséges, hogy elkerüljük csúnya dolgok bekövetkezését, amik akkor történnének, ha egy használatban levő algoritmusmodult törölnénk a memóriából. Létezik egy futásidejű algoritmuslekérdező felület is, amin keresztül a hívó kód meghatározhatja a rendszeren elérhető algoritmusokat. Ezt elsősorban a kulcstárgyalási protokollokhoz, például az ISAKMP/IKE-hez tervezték használni. Végül az algoritmusbejegyző felület a modulok számára lehetővé teszi, hogy egy vagy több algoritmust jegyezzenek be – különféle tulajdonságokat, mint például az algoritmus nevét, tömbméretét, a legkisebb és legnagyobb kulcshosszt megadva. Az adott pillanatban bejegyzett algoritmusok és tulajdonságaik a */proc/crypto* könyvtárban tekinthetők meg.

Adatok

A 2.5-ös rendszermagfában lásd:
- Documentation/crypto/
- include/linux/crypto.h
- crypto/

A ➲ http://samba.org/~jamesm/crypto címen található weboldal további tudnivalókat tartalmaz a fejlesztők számára, például a tennivalók listáját.

Összegzés

A titkosító API még fiatal, de lehet belőle valami, főleg ha a felsorolt jövőbeli tervezési célok egy részét megvalósítják.

Köszönetnyilvánítás

Köszönöm ***David Miller*** és ***Nancy Chan*** segítségét, akik átnézték ezt a cikket.

Linux Journal 2003. április, 108. szám

James Morris
Programfejlesztő – a Netfilter, az LSM, az SELinux és a Linux-rendszermag-titkosító API-projektekben dolgozik. Sydneyben független tanácsadó.