Kódolatlan szövegekkel dolgozó alkalmazások feltámasztása az Stunnel segítségével

Régi alkalmazásainkat módosításuk nélkül is párosíthatjuk korszerű titkosítási megoldásokkal. Mick Bauer elmondja, hogy az SSL megjelenése előtti programokat hogyan tudjuk naprakésszé varázsolni.

A világon rengeteg a munkáját jól végző, az idők próbáját kiállt, ám biztonsági szempontból rémálomnak számító hálózati alkalmazás létezik. ***Telnet***? Lenyűgöző egyszerűség és sokoldalúság, ám a bejelentkezési adatokat nyílt szövegben továbbítja. ***rcp***? Ismert, kiváló parancsfájlok írhatók hozzá, ám az ***rhosts*** alapú hitelesítés ideje, köszönhetően az IP-címek hamisításának, lejárt. Természetesen megtehetjük, hogy megszokott igavonóinkat titkosított utódaikra cseréljük le – ekkor az ***SSH*** felváltja a ***telnetet***, az ***scp*** vagy az ***SSH*** feletti ***rsync*** pedig az ***rcp***-t. Van azonban más lehetőségünk is, mégpedig az, hogy általános célú ***IPSec*** alagutat építünk ki minden olyan távoli állomás felé, amellyel adatcserét szeretnénk folytatni. Az utóbbi megoldás nyilván túlzás, az előbbit viszont sokszor könnyebb elképzelni, mint megvalósítani, különösen, ha az adott környezetben használt alkalmazások köre nem a mi ellenőrzésünk alá esik. Szerencsére arra is van lehetőség, hogy a hagyományos hálózati alkalmazásokat erőteljes titkosítással bővítsük.

Ez a lehetőség a nagy tudású ***SSL***-burkoló, az ***Stunnel*** használata. Ez alkalommal szeretném elmondani, hogy az ***Stunnel 4.0*** és az ***OpenSSL*** segítségével régi alkalmazásaink hogyan felelhetnek meg a korszerű követelményeknek, vagyis hogyan tudnak kellő biztonságot nyújtani. Hogy mi a helyzet a vezeték nélküli hálózatokkal? Aki kénytelen nyílt szövegekkel dolgozó hálózati alkalmazásokat gyengén védett, vezeték nélküli, például 802.11b hálózatokon keresztül használni, az nyugodtabban fog aludni, miután használatba vette az Stunnelt? Hamarosan ez is kiderül.

A jelszó nélküli tanúsítványok használatának veszélyei

A félig vagy teljesen önműködően, parancsfájl által létrehozott kiszolgáló tanúsítványok használata gyors és kényelmes megoldás, de van vele egy kis baj: az ilyen tanúsítványokhoz szinte soha nem tartozik jelszó. Az engedélyek módosításával gondoskodnunk kell arról is, hogy a tanúsítványt csak a root tudja olvasni, így legalább rendszerünk jogok nélküli használói nem tudnak hozzáférni. Azt is érdemes viszont figyelembe venni, hogyha rendszerünk root hozzáférését valaki megszerzi, akkor a jelszó nélküli tanúsítványt akár rossz célra is fel tudja használni.

Persze lehet, hogy ennek kockázata nem különösebben érdekel bennünket, és valóban, a jelszó nélküli kiszolgáló tanúsítványokat széles körben használják. Jobban belegondolva, elég bosszantó, ha az *Stunnel* összes indításakor kézzel kell beírni a jelszót. Természetesen a biztonsági szakemberek meglehetősen merész húzásnak tartják a jelszó nélküli tanúsítványok alkalmazását. Ha egy folyamat van annyira fontos, hogy a titkosítás kiemelt szerepet kapjon, akkor legyen annyira fontos, hogy minden alkalommal emberi közreműködéssel gondoskodunk az indításáról.

Háttérismeretek

Az ***Stunnel*** használatához két dologgal kell tisztában lenni. Először is, tudni kell, hogy a hálózati alkalmazások hogyan használják a hálózatot. Ha egy egyszerű, csupán egyetlen TCP-kaput használó alkalmazásról van szó, ilyen például a kizárólag a 23-as TCP-kapun át forgalmazó ***telnet***, akkor az ***Stunnel*** minden további nélkül működik. Ha UDP-t, kapumegfeleltetőt (***portmapper***) vagy egyéb dinamikus kapumegfeleltető módszert használ, akkor az ***Stunnellel*** nem jutunk előrébb. Az RPC alkalmazások például nem fognak működni, mert kapumegfeleltetőt használnak. Az FTP a 21-es TCP-kapun keresztül vezérli a forgalmat, de az adatkapcsolatokat magasabb sorszámú, véletlenszerűen kiválasztott kapukon át bonyolítja, így szintén kiesik a körből.

Másodszor, tisztában kell lenni a nyilvános kulcsú titkosítás alapjaival, bár az ***X.509***-es nyilvános kulcsú infrastruktúrát (***PKI***) nem feltétlenül kell ismerni. Ezekről több korábbi írásomban is volt már szó. (Például: „Az ***OpenSSH*** száz meg egy előnye”, Linuxvilág, 2001. február-márciusi szám). Most legyen elég annyi, hogy a nyilvános kulcsú titkosításnál

1. kódrészlet Kiszolgáló tanúsítvány létrehozása az OpenSSL használatával

```
helyiugyfel:/etc/stunnel# openssl req -x509
➥-newkey rsa:1024 -days 365
➥-keyout stunnel.pem -out stunnel.pem
Using configuration from
/usr/lib/ssl/openssl.cnf (A
/usr/lib/ssl/openssl.cnf fájlban megadott beál-
lítások használata)
Generating a 1024 bit RSA private key (1024 bi-
tes RSA titkos kulcs létrehozása)
...++++++
.........................................+++
+++
writing new private key to ´key2.pem´
(Az új kulcs kiírása a „key2.pem” fájlba)
Enter PEM pass phrase: (Adja meg a PEM jelszót:)
************
Verifying password - Enter PEM pass phrase:
(Jelszó ellenőrzése - Adja meg a PEM jelszót)
******
---
You are about to be asked to enter information
that will be incorporated into your certificate
request. (Az alábbiakban megadott adatok a tanú-
sítványkérelemben is szerepelni fognak.)
What you are about to enter is what is called
a Distinguished Name or a DN. (Következőként
a megkülönböztetett név, röviden DN megadására
lesz szükség.)
There are quite a few fields but you can leave
some blank. (Csak néhány értéket kell megadni,
és bizonyos mezők üresen is hagyhatók.)
For some fields there will be a default value,
(Egyes mezőknél alapértelmezett értéket lát
majd,)
If you enter ´.´, the field will be left blank.
(ilyenkor a „.” karaktert beírva hagyhatja üre-
sen a mezőt.)
---
Country Name (2 letter code) [AU]:US
(Országnév, kétbetűs kóddal)
State or Province Name (full name) []:Minnesota
(Állam, tartomány teljes neve)
Locality Name (eg, city) []:St. Paul
(Helység, pl. város neve)
Organization Name (eg, company) []:pelda
(Szervezet, pl. cég neve)
Organizational Unit Name (eg, section) []:
(Szervezeti egység, pl. osztály neve)
Common Name (eg, YOUR name)
[]:helyiugyfel.pelda.org
(Közös név, pl. az ön saját neve)
Email Address []:X.509kozpont@pelda.org

nearclient:/etc/stunnel# chmod 600 stunnel.pem
```

minden résztvevőnek két kulcsa van: egy nyilvános, amelyet megoszt a többi résztvevővel és egy titkos, amit megőriz magának. A többi fél a mi nyilvános kulcsunkat használja a nekünk szóló üzenetek rejtjelezésére, mi pedig titkos kulcsunkkal fejtjük azokat vissza.
A digitális aláírások pontosan fordítva működnek. Ha aláírunk valamit a titkos kulcsunkkal, akkor nyilvános kulcsunk birtokában bárki ellenőrizheti, hogy az aláírás létrehozása a mi titkos kulcsunk felhasználásával történt-e, vagyis lényegében mi voltunk-e az aláírók. Ismétlem, a rendszer működésének alapja, hogy a titkos kulcsot csak mi ismerjük, függetlenül attól, hogy nyilvános kulcsunkból hány másolat létezik a világon.
Az ***X.509*** világában a nyilvános kulcsot tanúsítványnak hívjuk, amely lényegében a titkos kulcsból és a tulajdonos személyes adataiból, például nevéből és e-mail címéből álló adathalmaz, digitális aláírással ellátva. A titkos kulcsot egyszerűen csak kulcsnak nevezzük. Hogy növeljük a keveredést, előfordul, hogy a kettőt együttesen ugyancsak tanúsítványnak nevezzük. A szövegkörnyezet szerencsére mindig segítségünkre van: ha jelszótól mentes tanúsítványt említünk, akkor tudjuk, hogy kombinált kulcsról/tanúsítványról van szó, mivel maga a tanúsítvány, mint a nyilvános kulcs, nem rendelkezhet jelszóval.
Azt hiszem, ennél rövidebben még sosem sikerült összefoglalnom a nyilvános kulcsú titkosítás és az ***X.509*** lényegét. Ha mindez nem elég a cikk további részeinek – stílusosan szólva – dekódolásához, akkor tanulmányozzuk át az ***Stunnel GYK***-t vagy a mindenre kiterjedő ***RSA Crypto GYK***-t. (Lásd az internetes források részt.) Nos, itt az ideje, hogy végre az ***Stunnellel*** is elkezdjünk foglalkozni.

Az Stunnel telepítése

Jó esélyünk van rá, hogy Linux-terjesztésünk tartalmaz bináris ***Stunnel*** csomagokat. Az újabb ***SuSE***, ***Fedora*** és ***Red Hat Enterprise*** terjesztésekben az ***Stunnel 4***-es, a ***Debian 3.0***-ban (***Woody***) pedig 3.22-es változata szerepel.
A 3.22-es biztos és áttekinthető működésű változat, kiváló leírások tartoznak hozzá, míg a négyesben számos részt újraírtak, amelynek köszönhetően több alagutat is könnyebben lehet kezelni vele. A továbbiakban az újabb változattal foglalkozok. Aki ***Debiant*** használ, erősen fontolja meg az újabb ***Stunnel*** forrásának letöltését és lefordítását.
Az ***Stunnelt*** bármelyik terjesztés alatt könnyen és gyorsan le lehet fordítani. Először ellenőrizzük, hogy telepítettük-e terjesztésünk ***OpenSSL*** csomagját (ennek neve általában ***openssl***), az ***OpenSSL*** fejlesztői könyvtárakat (***openssl-devel*** vagy ***libssl096-dev***) és a TCP-burkoló fejlesztői könyvtárakat (***Debianon libwrap0-dev***, a ***SuSE*** és ***Fedora*** alaptelepítéseknek pedig része). Ezután bontsuk ki az ***Stunnel*** forrás .tar állományát, és adjuk ki a megszokott parancsokat:

```
./configure && make && make install
```

Ha valami nem működik megfelelően, próbálkozzunk a `./configure –help` paranccsal, amely megjeleníti a különleges, a beállító parancsfájlnak átadható fordítási kapcsolókat. Ha végeztünk az ***Stunnel*** telepítésével, létre kell hoznunk néhány tanúsítványt, majd meg is kezdhetjük az alagutak használatát.

2. kódrészlet Átfogó beállítások az stunnel.conf fájlban

```
cert = /etc/stunnel/stunnel.pem
chroot = /var/run/stunnel/
pid = /stunnel.pid
setuid = nobody
setgid = nogroup
debug = 7
output = /var/log/stunnel.log
client = yes
```

A továbbiak nagy része csak az ***Stunnel 4.0.0*** és újabb változataira érvényes. Aki úgy dönt, hogy inkább a ***Debianban*** lévő 3.22-es ***Stunnel*** csomagot használja, az tanulmányozza át a csomaghoz mellékelt leírásokat vagy az ***Stunnel*** webhelyen szereplő példákat (lásd a forrásokat). A webhely anyagai túlnyomó részt a régebbi kiadást tárgyalják.

Tanúsítványok létrehozása az OpenSSL-lel

Minden ***Stunnel*** kiszolgálónak – vagyis olyan állomásnak, amely fogadja a valamelyik helyi nyílt szöveges szolgáltatásnak szóló titkosított csomagokat – szüksége van egy kiszolgáló tanúsítványra. Az ***Stunnel*** ügyfelek oldalán ilyesmire nincs szükség, hacsak nem akarjuk ügyféltanúsítvány alapú hitelesítésre állítani az ***Stunnel*** kiszolgálót. Az ügyféltanúsítvány alapú hitelesítés sajnos túlmutat jelenlegi témánkon, ezért egyik példában sem fog az ügyfél saját tanúsítvánnyal rendelkezni, ilyen csak a kiszolgáló oldalán lesz.
Ha az ***Stunnelt*** forrásból való fordítás után telepítettük, és ennek során kiadtuk a `make install` parancsot, akkor már van is kiszolgáló tanúsítványunk a (***/usr/local/etc/stunnel/stunnel.pem***) helyen. Ha a telepítést bináris csomagból végeztük, akkor lehet, hogy a csomag telepítés utáni teendőket ellátó parancsfájlja létrehozta számunkra a kiszolgáló tanúsítványt – de lehet, hogy nem.
A ***Fedora Core 1 Stunnel RPM***-je például – ki tudja, miért – üres tanúsítványt állít elő. Nyilván nem kell mondani, hogy helyes tanúsítványra van szükségünk. ***Fedora*** alatt ilyet úgy tudunk előállítani, hogy belépünk a ***/usr/share/ssl/certs*** könyvtárba, kiadjuk a `make stunnel.pem` parancsot, majd az ***stunnel.pem*** fájlt átmásoljuk a ***/etc/stunnel*** könyvtárba. Ehhez a tanúsítványhoz nem fog jelszó tartozni, ahogy az ***Stunnel*** forrás csomagjának `Make` parancsfájlja által létrehozotthoz sem tartozik.
Ha ***Stunnel*** kiszolgálónkhoz valamiért szükségünk van egy tanúsítványra, esetleg az önműködően létrehozott nem felel meg az igényeinknek – például, mert nem tartozik hozzá jelszó –, akkor az ***OpenSSL*** paranccsal sajátot is elő tudunk állítani. Az egyébként nagy tudású program használatát részletesebben – helyhiány okán – nem ismertetem. Szerencsére az ***Stunnel*** GYK (lásd a forrásokat) az ***OpenSSL*** és az ***Stunnel*** párbeszédével kapcsolatosan leggyakrabban felmerülő kérdésekre megadja a válaszokat, illetve további, az ***OpenSSL***-lel kapcsolatos információforrásokra is tartalmaz hivatkozásokat. Csak annyit mondanék, hogy lehetnek olyan helyzetek, amikor érdemes saját hitelesítő szervezetet (CA) létrehozni, telepíteni a megfelelő CA tanúsítványt (nem a kulcsot) az összes ***Stunnelt*** futtató rendszerre, majd ezeket a CA tanúsítványokat használni az összes kiszolgáló tanúsítvány aláírására. Például akkor van erre szükség, ha azt akarjuk, hogy az ***Stunnel*** kiszolgálók csak a megadott tanúsítványok birtokában lévő ***Stunnel*** ügyfélgépektől fogadják a csatlakozási kéréseket. Sok, sőt a legtöbb ***Stunnel*** felhasználó számára a saját aláírással ellátott tanúsítványok is tökéletesen megfelelnek, ilyenkor nincs gond a CA üzemeltetésével. Nézzük tehát, hogyan lehet saját aláírású kiszolgáló tanúsítványt készíteni az ***OpenSSL***-lel.

3. kódrészlet Szolgáltatások megadása a helyiugyfelen

```
[telnets]
accept = 23
connect = tavolikiszolgalo.pelda.org:992
```

A lényeg az *1. kódrészlet* első sorában található. A paranccsal, ha a kapcsolóin balról jobbra haladunk végig, arra utasítjuk az ***OpenSSL***-t, hogy egy ***X.509*** digitális tanúsítvány formátumú tanúsítványkérelmet állítson elő, 1024 bites kulccsal dolgozó ***RSA*** titkosítási eljárást használjon, a tanúsítvány 365 napig legyen érvényes, a kulcsot és a – nyilvános – tanúsítványt ugyanabba a fájlba (***stunnel.pem***) írja ki. Ha inkább jelszómentes tanúsítványt szeretnénk létrehozni, akkor a sor végére illesszük oda a `-nodes` kapcsolót is. Előbb mindenképpen érdemes átolvasni az „A jelszómentes tanúsítványok használatának veszélyei” című széljegyzetet.
Az 1. kódrészlet fennmaradó része az ***OpenSSL***-lel folytatott párbeszédet szemlélteti. A program bekéri a tanúsítványba beillesztendő, az ***X.509*** szabvány által megadott egyéb adatokat, mint például az ország és a régió nevét. Ezt elrontani nem nagyon lehet, viszont a ***Common Name*** (közös név) megadása nagyon fontos. Ha kiszolgáló tanúsítványt hozunk létre, akkor a benne szereplő közös névnek annak az állomásnak a teljesen minősített tartománynevével, vagyis teljes állomásnevével kell egyeznie, amely a tanúsítványt használni fogja.
Az ***SSL/TLS***-képes alkalmazások nagy része a közös név alapján indít ***DNS***-kérést, és szerzi be a kiszolgáló IP-címét. Ugyan az ***Stunnel*** ezt nem teszi meg, amíg a beállítások módosításával rá nem vesszük, ám a közös név azonossá tétele a teljesen minősített tartománynévvel mindenképpen jó szokás.
Az 1. kódrészlet utolsó sorában `0600`-ra (`-rw-------`) módosítottam az új kiszolgáló tanúsítványra vonatkozó engedélyeket. Mivel az ***OpenSSL***-t rootként futtattam, tulajdonosa már eleve ő volt. Minden kiszolgálói tanúsítványnál fontos, legyen az akár jelszóval ellátott, vagy jelszómentes, hogy csak a root tudjon hozzáférni, és ő is csak olvasni tudja. Jómagam az ***OpenSSL*** parancsot a ***/etc/stunnel*** könyvtárból futtattam, a tanúsítványom tehát egyből a helyére került, és nem volt szükség arra, hogy kézzel helyezzem át.

Az Stunnel átfogó beállításainak megadása

Ha előállítottuk a megfelelő kiszolgáló tanúsítványt, állítsunk be magunknak egy alagutat. Ez a lépés tagadhatatlanul könnyebb lesz, mint az előző, viszont a programváltozatoktól is erősebben függ. Az ***Stunnel 4.0*** előtti változatai minden beállítást parancssori kapcsolóként fogadtak. A je-

4. kódrészlet Szolgáltatások megadása a tavolikiszolgalon

```
[telnets]
accept = 992
connect = 23
```

lenlegi változatban az egyetlen létező parancssori kapcsoló az, amely szükség esetén a beállító fájl alapértelmezettől eltérő helyét adja meg.

Ha az ***Stunnelt*** forrásból, az alapértelmezett fordítási beállításokkal telepítettük, akkor a ***/usr/local/etc/stunnel*** könyvtárban keresi a beállító fájlt. Ha a telepítést bináris csomagból végeztük, akkor nagy valószínűséggel a ***/etc/stunnel*** könyvtárban kell keresgélnünk. A 2. *kódrészlet* egy rövidített minta, mely az ***stunnel.conf*** fájl átfogó beállításait tartalmazza (a rövidítés leginkább a megjegyzések eltávolítására terjedt ki).

A `cert` átadott érték azt közli az ***Stunnellel***, hogy hol keresse kiszolgáló tanúsítványát; logikusan ilyesmire csak az ***Stunnel*** kiszolgálón van szükség, az ügyfeleken nincs.

A `chroot` érték azt a könyvtárat adja meg, amelybe indítás után az ***Stunnel*** `chroot` műveletet hajt végre – persze csak azután, hogy beolvasta a beállító fájlt és a kiszolgáló tanúsítvány fájlokat. Lehet, hogy a `chroot` helyszínéül szolgáló börtönt (***jail***) érdemes kézzel létrehozni, majd feltölteni néhány hasznos dologgal, például külön ***/etc/hosts.allow*** és ***/etc/hosts.deny*** fájlokkal, már amennyiben TCP-burkoló stílusú hozzáférés-vezérlést akarunk használni.

A `pid` azt határozza meg, hogy az ***Stunnel*** hova írja ki folyamatazonosítóját. Az itt megadott útvonal a `chroot` révén megadotthoz viszonyított, vagyis az ***Stunnel*** csak a `chroot` művelet végrehajtása után írja ki folyamatazonosítóját.

A `setuid` és a `setgid` adja meg, hogy az ***Stunnel*** melyik felhasználó és csoport nevében fusson. Ha az ***Stunnelnek*** 1025 alatti számú TCP-kapun is hallgatóznia kell, akkor rootként kell indítani. Ilyenkor beolvassa a beállító fájlt és a kiszolgáló tanúsítványt, köt a védett kapuhoz, majd lefokozza önmagát.

Alapesetben az ***Stunnel*** biztonsági értesítéseit és ennél nagyobb fontosságú naplóüzeneteit a helyi démon ***syslog*** eszköznek továbbítja. A ***Fedorában*** található változat üzeneteit az `authpriv` eszköznek küldi, innen ezek a ***/var/log/secure*** fájlba kerülnek. A `debug` kapcsolóval eltérő naplózási szintet is választhatunk. A hetes a legmagasabb szint, ezt akkor érdemes használni, ha nem sikerül munkára bírnunk az ***Stunnelt***. Az output beállítással arra vehetjük rá az ***Stunnelt***, hogy üzeneteit a megadott fájlba írja, és ne adja át a ***syslognak***.

A 2. kódrészlet utolsó sorában a `client` (ügyfél) kapcsolót „`yes`”-re, azaz igenre állítjuk, amivel jelezzük, hogy a megadott rendszer kezdeményezni fogja az ***SSL***-tranzakciókat, és nem fogadni. Értelemszerűen a kapcsolatokat fogadó kiszolgálón ez az érték az alapértelmezett „`no`”, vagyis „nem” marad.

Alagút megadása

Lassan célba érünk, végre az alagutakkal is foglalkozhatunk. Példaképpen ***telnet*** kapcsolat létesítése céljából hozzunk létre egy alagutat a `helyiugyfel` és a `tavolikiszolgalo` állomások között. A `tavolikiszolgalo` ***stunnel.conf*** fájljának

5. kódrészlet Példa telnet kapcsolat

```
helyiugyfel:/usr/local/etc/stunnel#
➥telnet 127.0.0.1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is ´^]´.
Fedora Core release 1 (Yarrow)
Kernel 2.4.22-1.2115.nptl on an i686
login: tavoli_felhasznalonev
Password: **********
Last login: Sun Jun 13 21:39:17 from
localhost.localdomain
[tavoli_felhasznalonev@tavolikiszolgalo
➥tavoli_felhasznalonev]$
```

átfogó beállításai gyakorlatilag meg is egyezhetnek a 2. kódrészletben szereplőkkel, azzal a kivétellel, hogy a `client` beállításnak „`no`” értéket kell adni. A két állomás beállításai között a legfontosabb különbség a szolgáltatások megadásában lelhető fel.

Mielőtt mélyebbre ásnánk, ismerkedjünk meg részletesebben is a példakörnyezettel. Tegyük fel, hogy a `tavolikiszolgalo` már ***telnet*** kiszolgálóként üzemel, vagyis már képes a ***telnet*** kapcsolatok fogadására a 23-as TCP-kapun keresztül. Ha nem akarjuk, hogy a `helyiugyfel` a nyílt szövegű forgalmat bonyolító kapura csatlakozzon, akkor az ***SSL***-kapcsolatok számára másik kaput kell választanunk. Micsoda szerencse, hogy az ***IANA*** már kiválasztotta az ***SSL*** alapú ***telnet***, vagyis ***telnets*** kapcsolatok kapuját, ez a 992-es TCP-kapu.

Az alagútnak tehát a `helyiugyfel` és a `tavolikiszolgalo` 992-es TCP-kapuja között kell létrejönnie. De vajon honnan tudja az SSL kezelésére képtelen ***telnet*** parancsunk és hasonló hiányosságokkal küzdő ***telnet*** kiszolgálónk, hogy hogyan használja az alagutat? A kérdés csapda volt, az alagút ugyanis a küldő és a fogadó ***telnet*** folyamat számára egyaránt tökéletesen átlátszó.

A `helyiugyfel` ***Stunnel*** folyamata a szokásos kapun, a 23-as TCP-n fogadja a kapcsolatot (a kapu számát meg lehet változtatni), SSL alapon titkosítja a csomagokat, majd továbbítja őket a `tavolikiszolgalo` 992-es TCP-kapujára.

A `tavolikiszolgalo` visszafejti a csomagokat, majd továbbítja őket a 23-as TCP-kaput figyelő helyi ***telnet*** folyamatnak. Persze a csomagok előbb a ***xinetd***-hez vagy az ***inetd***-hez kerülnek, az ***in.telnetd*** csak utána kapja meg őket, de a lényeg – remélem – érthető.

Ennél a megoldásnál, ha a `helyiugyfel` felhasználói csatlakozni szeretnének a `tavolikiszolgalohoz`, akkor kiadják a `telnet 127.0.0.1` parancsot, aminek hatására létrejön a titkosított kapcsolat. A `tavolikiszolgalo` válaszcsomagjai ugyanezen az útvonalon haladnak, csak értelemszerűen ellentétes irányba. Mindkét ***telnet*** folyamat, a ***telnet*** ügyfél és az ***in.telnetd*** is azt hiszi, hogy helyi felhasználóval áll kapcsolatban, ám a csomagok a valóságban egy SSL titkosítású ***Stunnel*** kapcsolaton keresztül utaznak. Nagyjából ennyi a magyarázata annak az összesen hat sornak, amely a *3.* és a *4. kódrészletben* látható.

Mint kitűnik, a szolgáltatások megadásához mindössze két-két sorra van szükség, az `accept` sor a figyelt kaput adja meg, a `connect` sor pedig a célkaput. A két sor pontos jelentése attól függ, hogy ***Stunnel*** ügyfélről vagy ***Stunnel*** kiszolgálóról van-e szó. Az ügyfélrendszereken (`client = yes`) az ***Stunnel*** nyílt szöveges csomagokat vár a figyelt kapun, és titkosított csomagokat továbbít a célkapura. A kiszolgálókon (`client = no`) az ***Stunnel SSL***-kapcsolatokat, vagyis titkosított csomagokat fogad a figyelt kapun, és ezeket visszafejtett formában továbbítja a célkapura.

Érdemes megemlíteni, hogy az ügyfeleken a `connect` sorban nemcsak kaput, de egy távoli állomás nevét vagy IP-címét is meg lehet adni, ahogy a 3. kódrészletben is `connect = tavolikiszolgalo.pelda.org:992` szerepel. A 3. és 4. kódrészlet beállításai alapján tetszőleges állomás csatlakozhat a `helyiugyfel` 23-as TCP-kapujára, majd az alagúton keresztül kapcsolatba léphet a `tavolikiszolgaloval`. Ezt többféle módszerrel is meg tudjuk akadályozni, többek közt az ***iptables*** használatával. Ha a `helyiugyfelen` az `accept = 127.0.0.1:23` sort adjuk meg, akkor az ***Stunnel*** ügyfélfolyamatát rá tudjuk venni arra, hogy csak helyi kapcsolatokat fogadjon.

A megoldás a `tavolikiszolgalon` is működik. Ha a `tavolikiszolgalonak` több hálózati csatolója is van, például ***eth0***, ***wlan0***, ***ppp0*** stb., akkor hasonló `accept` utasítással választhatjuk ki azt a csatoló IP-címet, amelyen fogadni akarjuk az alagúton keresztül érkező csomagokat. Az ***Stunnel*** ügyfeleken és kiszolgálókon az `accept` utasítások IP-címe alapesetben az `any` (vagyis az összes helyi csatoló), a `connect` utasítás alapértelmezett IP-címe pedig a `127.0.0.1` (`localhost`).

Mit kell tudni a `tavolikiszolgalon` futó ***telnet*** kiszolgálóról? Végül is a nyílt szöveges ***telnet*** kapcsolatok használata általában elég rossz ötlet. Éppen ezért ne feledjük el a `bind = 127.0.0.1` sort hozzáadni a */etc/xinetd/telnet* parancsfájlhoz, ennek hatására csak helyi folyamatok kapcsolódhatnak a 23-as TCP-kapuhoz.

Ha az ügyfélen és a kiszolgálón egyaránt megtettük a szükséges beállításokat, akkor egyszerűen az `stunnel` parancs kiadásával indítsuk el az ***Stunnelt***. Különösebben nem kell foglalkoznunk azzal, hogy a kiszolgálón vagy az ügyfélen indítjuk el előbb, az ügyfél úgysem próbál alagutat létrehozni, amíg arra tényleges igény nincs, vagyis amíg – például – el nem indítunk egy ***telnet*** kapcsolatot. Ha valamelyik, esetleg mindkét fél jelszóval védett kiszolgáló tanúsítványt használ, akkor itt az ideje, hogy beírjuk a jelszót. Erről akkor se feledkezzünk el, ha az Stunnel indítására parancsfájlt készítünk. Ha a tanúsítvány beolvasása megtörtént, elvileg minden készen áll.

Adjuk ki a `ps auxw` parancsot, és ellenőrizzük, hogy a kimenetben szerepel-e az ***Stunnel*** folyamat. Maga az ***Stunnel*** a konzolra nem ír ki semmit, amivel visszaigazolná gond nélküli elindulásának tényét. Az erre utaló és egyéb jelzéseket, köztük az indítási üzeneteket a ***syslognak*** küldi el.

Ha az ***Stunnel*** az ügyfélen és a kiszolgálón is fut, akkor a `helyiugyfel` felhasználói a `helyiugyfel` 23-as TCP-kapujára – például a `telnet 127.0.0.1` paranccsal – csatlakozva válthatják ki az alagút létrehozását. Az *5. kódrészlet* egy titkosított ***telnet*** kapcsolat felépülését szemlélteti.

Az Stunnel és az inetd/xinetd kapcsolata

Gyónnom kell. Az itt szereplő példák nem mutatják be a *telnet* kapcsolatok *Stunnel* segítségével végzett titkosításának leghatékonyabb módját, bár azt legalább megígérhetem, hogy kipróbáltam őket, és működnek.

Azzal szeretném menteni magam, hogy szerettem volna egyszerűen ismertetni az általános TCP alapú szolgáltatások alagútba terelését, és olyan módszert akartam bemutatni, ami bármely egyetlen TCP-kaput használó szolgáltatással működik. A telnet, a pop3 és sok más szolgáltatás indítását viszont egy szuperkiszolgáló – *inetd* vagy *xinetd* – végzi, és az *Stunnel* számos különböző, itt nem tárgyalt szolgáltatással támogatja ezeket.

Megoldható például hogy az *Stunnel* kiszolgáló ne továbbítsa egy connect (csatlakozás) utasítással a csomagokat, hanem adja át őket egy másik folyamatnak, amelyet egy exec hívással az *stunnel.conf* alapján maga indít el. Az *Stunnel* maga is meghívható dinamikusan *inetd* vagy *xinetd* alól.

Az *Stunnel* dinamikusan indított démonokkal való együttes, illetve *inetd* vagy *xinetd* alatti használatáról az *stunnel(8) man* oldal és az *Stunnel GYK* tartalmaz további tájékoztatást.

Látható, hogy a felhasználó szempontjából a `tavolikiszolgalohoz` való kapcsolódás pontosan úgy zajlik, mint bármely más ***telnet*** kapcsolat létrehozása. A kódrészlet végén megjelenő ***bash*** parancssor jelzi, hogy a `tavolikiszolgaloval` valóban sikeresen létrejött az összeköttetés.

Összefoglalás

Remélem, az eddigiek alapján mindenki el tudja kezdeni az ***Stunnel*** használatát, illetve a vele való ismerkedést. Mint máskor, most is csak a felszínt érintettük. Mindenkinek meghagyom azt az örömöt, hogy önállóan fedezhesse fel az ***Stunnel*** ügyféltanúsítványok alapján történő alagúthitelesítési képességét, a TCP-burkoló stílusú hozzáférésvezérlés támogatását, valamint az ***stunnel.conf*** fájlba beilleszthető átfogó és a szolgáltatásokra egyedileg jellemző beállítások sokaságát.

Ennek során bátran forduljunk segítségért az ***stunnel(8) man*** oldalhoz – mire a végére érünk, megszokott, titkosításra képtelen, TCP alapú alkalmazásaink forgalmát többé nem lehet majd lehallgatni.

Linux Journal 2004. szeptember, 125. szám

Mick Bauer (mick@visi.com)
Biztonsági szakember, a Linux Journal biztonsági témákkal foglalkozó szerkesztője, biztonsági tanácsadó a Minnesota állambeli Minneapolisban található Upstream Solutions LLC Inc.-nél.