Az OpenSSH száz meg egy előnye (1. rész)

Mick az ssh felszínét kapargatja.

Eljött a Paranoid Pingvin mai mesedélutánjának ideje. De ne aggódjanak – ez a mese pont ugyanolyan mazsoláznivalókról szól, mint amilyeneket már megszokhattak a Linuxvilágtól. Sőt, az az igazság, hogy az OpenSSH-ban olyan sok mazsoláznivaló található, hogy ez a cikk a következő számokba is átnyúlik majd!

Ebben a hónapban az ssh hátterével és szerkezetével foglalkozunk. Megnézzük, miképpen kell lefordítani, illetve telepíteni az OpenSSH-t, hogyan használhatunk ssh-t a telnet titkosított kiváltására, hogyan állítsunk be néhány alapvető változót, illetve miképpen használjuk az scp-t titkosított fájlátvitelhez. A következő hónapban az RSA/DSA bejelentkezésről fogok írni, a helyi kapu-átirányításról, a távoli parancsvégrehajtásról, illetve néhány más fejlett és hatékony ssh/OpenSSH képességről.

A rend kedvéért először arról szeretnék beszélni, miképpen került hozzánk ez a program, illetve azokról az emberekről ejtenék pár szót, akik elérhetővé tették számunkra.

Az SSH története

Amióta csak megszületett, a Unixban az volt az egyik legnagyszerűbb dolog, hogy a rendszer-karbantartási feladatokat többféleképpen is el lehetett végezni távoli konzolokról. Sajnos, a legtöbb ilyen módszer (telnet, rsh és az X, csak hogy néhányat említsünk) mindent egyszerű szövegként küld át a hálón, beleértve a jelszavakat is. Ahogy az Internetet egyre többen használták, és ahogy egyre több lelkes önjelölt kalózocska és unatkozó csomagszaglászó kisiparos jelent meg, a szövegalapú hálózati felügyelet túlhaladottá vált.

Néhány évvel ezelőtt azonban egy finn betyárszaki, *Tatu Ylonen* létrehozott egy lélegzetelállítóan jó dolgot, amit Secure Shellnek, azaz ssh-nak neveznek. Az ssh olyan eszközök csoportja, ami nagyjából a Sun rsh, rcp és rlogin parancsainak felel meg, egyetlen igen fontos eltéréssel: az üldözési kényszerrel. Az ssh-val mindent meg lehet valósítani, ami az rsh, rcp és a rlogin segítségével elérhető, kihasználva a választott szabad kódkönyvtárakat és azonosító módszereket. Az ssh egy változata erősen függ az RSA-tól, a kitűnő, de szabadalmak által védett módszertől, ami megköveteli, hogy minden őt használó program engedélyek birtokosa legyen (azaz fizessenek érte), kivéve azt az esetet, ha „nem kereskedelmi" módon használják. Gyakran azonban a nem kereskedelmi felhasználás is nehézkes lehet. Várjunk csak, az RSA US szabadalom 2000 szeptemberében lejárt – gond megoldva, nem? Majdnem: Tatunak is meg kell élnie valamiből, így aztán mire az RSA végre kötetlenné válna, az ssh vált kötötté, mivel Tatu cége, a F-Secure megszigorította az engedélykövetelményeket. Lényegében az ssh 2.0 változatától kezdve az ingyenesség és a szabad felhasználás többé már nem engedélyezett (az RSA-fejleményektől függetlenül). Akárhogy is nézzük, az ssh csak akkor válhat internetszabvánnyá (ahogy azt Tatu is szeretné), ha legalább egy ingyenes megvalósítása létezik.

Ezzel be is léptünk *Theo de Raadt* és az OpenBSD birodalmába. Az OpenBSD, a BSD ingyenes változatának, a NetBSD-nek egy biztonságra sokat adó hajtása. Theo és a nyílt forráskód közösségbeli hittestvéreink az OpenBSD fejlesztőgárdájában szerették volna beil-

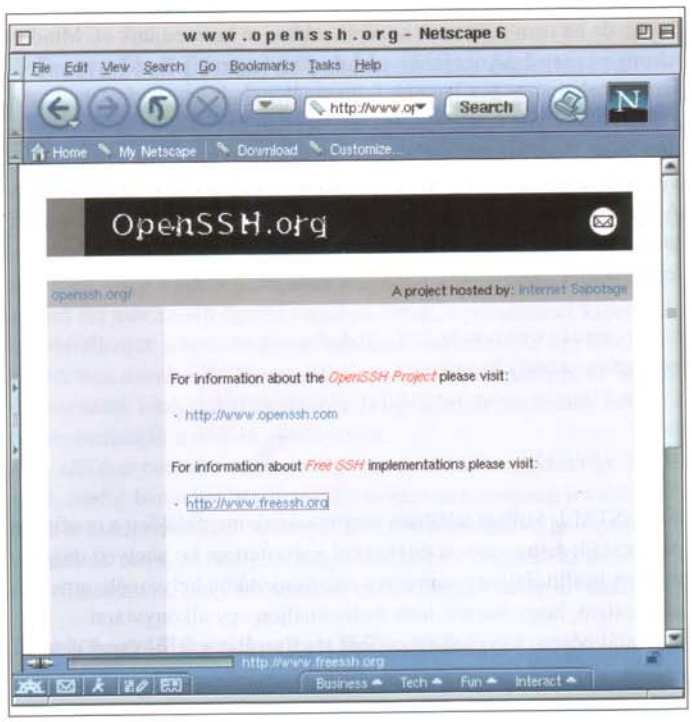
1. táblázat Az ssh_config néhány alapvető értéke

Érték	*Lehetséges érték*	*Leírás*
CheckHostIP	*Yes No*	(Alapértelmezett=yes) Figyeljen-e az ügyfélgép IP-címének változására ismert számítógépkulcsok esetén, vagy sem. Figyelmezteti a felhasználót, ha ellentmondást talál.
Chiper	*3des blowfish*	(Alapértelmezett=3des) Melyik blokk-kódolást használja a ssh v.1 kapcsolatokhoz.
Chipers	*3des-cbc, blowfish-cbc, arcfour, cast128-cbc*	Milyen sorrendben próbálja végig blokk-kódolásokat a ssh v.2 kapcsolatokhoz.
Compression	*Yes No*	Használja-e a gzip programot a titkosított adatok tömörítésére vagy sem. Korlátozott sávszélesség esetén hasznos lehet, de valamelyest lassít.
ForwardX11	*Yes No*	(Alapértelmezett=yes) Átirányítja az X kapcsolatokat a titkosított csatornára, és emellett megfelelően beállítja a DISPLAY változót. Nagyon hasznos lehetőség!
Password-Authentication	*Yes No*	(Alapértelmezett=yes) Megpróbáljon-e (titkosított) Unix jelszóazonosítást végezni, az RSA/DSA azonosításon felül, vagy helyett.

leszteni az ssh-t az OpenBSD 2.6-os kiadásába. Csakhogy beleütköztek az ssh különböző megkötéseibe. Amint tudomásukra jutott, hogy egy svéd programozó, *Bjoern Groenvall* az ssh 1.2.12 (Ylonen legutolsó ingyenes – kivéve az RSA részt – változata) egy fejlettebb változatát mutatta be, az OpenBSD-s fickók egy percet sem vártak tovább, azonnal a nyilvánosság elé vitték. Az OpenSSH azóta az OpenBSD része, és mára már jóformán minden Unix-változatra átültették.
A Groenvall munkájára épülő OpenSSH (az ő OSSH nevű változata szintén elérhető) az ssh protokoll korábbi változatait egészíti ki modulrendszerrel és titkosítási eljárásokkal oly módon, hogy az OpenSSH-t egyetlen szabadalmaztatott eljárás vagy hasonló dolog használata nélkül is le lehet fordítani (például az ssh v.1 protokollok nélkül, ezek az RSA-tól függenek). Az OpenBSD csapat másik újítása, hogy az OpenSSH kódot szétválasztotta egy tiszta (clean) változatra, ami olyan egyszerű és felületfüggetlen amennyire csak lehetséges, illetve egy hordozható (portable) változatra, amit többféle Unix-változat alá is le lehet fordítani az OpenBSD mellett.
Ez az utolsó újítás a legérdekesebb a linuxosok számára: a tiszta változat megtartása teszi lehetővé a kód ellenőrizhetőségét, ami így alapvetően állandó és biztonságos. Csak miután Theo (aki egy igazi paranoid) rábólintott erre a kódra, készülhetnek el a hordozható fejlesztések. Így aztán mi egy olyan programcsomagból húzhatunk hasznot, amely nagyon biztonságos, emellett százszázalékosan Linux-megfelelő.
Mellesleg az OSSH felfedezésétől számítva, kevesebb mint két hónap kellett ahhoz, hogy az OpenBSD csapat kiadja az OpenSSH 1.2.2-t, és mindössze hat és fél hónap, hogy a teljes mértékben hordozható, és ssh v.2-megfelelő OpenSSH 2.0 megjelenjen. Ez még akkor is figyelemre méltó teljesítmény, ha tekintetbe vesszük, hogy Ylonen és Groenvall munkájára építettek, különösen, ha beszámítjuk a végeredmény kitűnő minőségét, illetve azt, hogy senki nem fizetett nekik ezért egy petákot sem!
Nos, ennyi lenne az ssh és az OpenSSH története. Remélem, egyetértünk abban, hogy elég lenyűgöző, akárcsak az OpenSSH maga, amely minden valószínűség szerint hamarosan a nyílt forráskódú Unixok elsődleges ssh változatává válik.
Egyébként az „ssh v.1.x" és a „ssh protocoll v.1" az ssh program kiadásnak, illetve a protokollnak felel meg, és nem igazán jelentik ugyanazt. De mivel a csomag és a protokoll változatszámai nagyjából megfeleltethetők egymásnak, itt most az „ssh v.1.x" kifejezést fogom használni, ha az RSA-alapú ssh/OpenSSH változatra gondolok, illetve az „ssh v.2.x" kifejezést, amennyiben a RSA/DSA-t támogató változatokra szeretnék utalni. Ha esetleg nem tudnánk, mi a különbség az RSA és a DSA között, legyen elég annyi, hogy mindkettő ugyanazt teszi, de a DSA-t nem köti semmiféle szabadalom vagy engedély.

Hogyan működik az SSH?

A Secure Shell működése erősen hasonlít a webes SSL-kapcsolatokra (nem véletlen, hogy az OpenSSH által használt titkosító függvényeket az OpenSSL nyújtja, amely a Netscape Secure Socket Layer forráskód könyvtárainak szabadon felhasználható változata). Mindkettő titkosított csatornákat épít fel, amelyek általános gépkulcsokra (host key) épülnek, vagy nyilvánossá tett bizonyítványokat (digitális aláírás – digital certificate) ellenőriznek egy megbízható szolgáltatónál (certificate authority, például a VeriSign). Lássuk, tulajdonképpen hogyan is épül fel ez a kapcsolat.
Először is, az ügyfél és a kiszolgáló (nyilvános) gépkulcsokat cserélnek. Ha az ügyfélgép még soha nem találkozott az adott nyilvános kulccsal, az ssh és a legtöbb böngésző rákérdez, hogy elfogadjuk-e a bizonytalan eredetű kulcsot. Ezután a gépek a kulcsokat használják arra, hogy létrehozzanak egy kapcsolatkulcsot (session key). Minden további adattitkosítás a kapcsolatkulcs alapján történik valamelyik blokk-kódolási módszer szerint, mint amilyen például a Triple-DES (3DES), a blowfish vagy az idea. Ezután a kiszolgáló megpróbálja azonosítani az ügyfélgépet RSA vagy DSA bizonyítványok segítségével. Amennyiben ez nem lehetséges, az ügyféltől hagyományos felhasználónév/jelszó párt vár (lehetőségünk van rhosts-típusú, tehát az ügyfél IP-címe alapján működő azonosításra RSA-kulcsokkal vagy anélkül, de az OpenSSH támogatja a Kerberos IV és az skey rendszereket is). Végül a sikeres azonosítás után kezdetét veheti a tulajdonképpeni kapcsolat: egy távoli héjprogram, biztonságos fájlátvitel, távoli parancsfuttatás stb.



Amint azt korábban már említettem, az ssh tulajdonképpen egy eszközgyűjtemény:

- sshd – kiszolgáló démonként működik minden más programhoz.
- ssh – az elsődleges eszköz: távoli héjprogram, távoli parancsvégrehajtás, illetve kaputovábbítás.
- scp – fájlátvitelhez használható eszköz.
- sftp – interaktív fájlátvitelre használható eszköz. Általában csak kereskedelmi ssh-csomagokban található meg.
- ssh-keygen – titkos, illetve nyílt kulcspárokat készít RSA vagy DSA azonosításhoz (ide értve a gépkulcsokat is).
- ssh-agent – az RSA/DSA azonosítás automatizálására szolgáló démon.
- ssh-add – titkos kulcsot tölt be az ssh-agent folyamatba.
- ssh-askpass – az ssh-add X felülete.

Ezen eszközök közül a legtöbb felhasználót csak az ssh érdekli, hiszen a „titkosított telnet" az ssh legegyszerűbb felhasználása.
Az scp, az ssh-agent, és az ssh-add, valamint az ssh a maga erős azonosítási és TCP-kapuátirányítási képességével együtt rendkívül hatékony csomagot jelent. Mivel paranoidok vagyunk, és minél több hálózaton átküldött anyagot szeretnénk titkosítani, ezzel a rugalmassággal tényleg sokat nyerhetünk.

Az OpenSSH letöltése és telepítése

Ha az OpenSSH legutolsó változatát szeretnénk letölteni akár forráskód, akár RPM formátumban, először is látogassuk meg az OpenSSH honlapját (lásd a Kapcsolódó címeket). Ugyanitt található az OpenSSL, ez elengedhetetlen az OpenSSH-hoz. Szükségünk lehet még a zlib-re, ez a freesoftware.com honlapján (lásd a Kapcsolódó címeket) érhető el.

Megjegyzem, nem biztos, hogy az RPM-ekkel könnyen boldogulunk. Amikor RPM-eket akartam telepíteni a OpenSSH.com-ról a SuSE Linuxot futtató laptopomra, minden kiválóan működött kivéve az sshd-t, ami nem települt fel a SuSE „chkconfig" csomag hiánya miatt. Az egyes Linux-változatok vagy szenvednek ebben a hibában vagy nem (már ha van rajtuk egyáltalán chkconfig). Nos, az is lehet, hogy a SuSE-nek saját RPM csomagja van az OpenSSH-hoz. Amennyiben az RPM nem működik, az OpenSSH-t (és valószínűleg az OpenSSL-t, illetve a zlibet is,) forrásból kell lefordítani. A Linux öreg motorosainak a „készítsd el saját telepítésed" stílus megszokott dolog, de ha nem tartozunk közéjük, akkor se keseredjünk el. Mindhárom csomag egy .configure parancsfájlt használ, ez pedig szükségtelenné teszi a legtöbb felhasználónak, hogy szerkesztenie kelljen bármilyen makefájlt. Ha rendszerünkben megvan a gcc fordító és az általában használatos programkönyvtárak, illetve ezek viszonylag naprakészek, a fordítás általában gyors és zökkenőmentes. Az én esetemben az OpenSSL 0.9.5a és a lib 1.1.3 változat telepítése után a következő lépéseket kellett megtenni az OpenSSH 2.1.1p4 telepítéséhez:

```
tar -xzvf openssh-2.1.1p4.tar.gz
cd openssh-2.1.1p4
./configure --sysconfdir=/etc/sshd
make
make install
```

Az INSTALL fájlban található utasításoknak megfelelően a configure parancsfájlt testre szabott értékekkel egészítettem ki: ahelyett, hogy minden beállítófájlt egyszerűen a /etc könyvtárba helyeznék, arra utasítottam, hogy hozzon létre és használjon egy alkönyvtárat /etc/sshd néven. Mivel az OpenSSH e változata az RSH és a DSA kulcsokat is támogatja, nem rossz gondolat mérsékelni azt a zűrzavart, amit az ssh a /etc könyvtárban kelt. Egy csak ügyfélalapú telepítés esetén mindössze ennyit kell tenni. Megjegyezzük, hogy a fent említett változatszámok már valószínűleg túlhaladottá váltak, mire ezt olvassuk: ne felejtsük el megnézni az OpenSSH honlapját a legfrissebb változatokért.

Ha a sshd Secure Shell démont is futtatni szeretnénk (azaz ssh kapcsolatokat kívánunk fogadni távoli gépekről), létre kell hoznunk indítófájlokat, valamint a SuSE esetében át kell szerkeszteni a /etc/rc.config fájlt.

A Red Hat könyvtár tartalmaz egy sshd.init fájlt, amit a /etc/rc.d könyvtárba lehet másolni, és a megfelelő futásszintű könyvtárból hivatkozni rá (/etc/rc.d/rc2.d stb.). Ugyanitt található az „sshd.pam", ezt a /etc/pam könyvtárba kell másoljuk, amennyiben Pluggable Authentication Modules (PAM) rendszert és „openssh.spec"-et használunk (ezzel saját OpenSSH RPM csomagokat lehet létrehozni). Ezeket a fájlokat nyilvánvalóan a RedHat alatti használatra tervezték, de elképzelhető, hogy működnek más RedHat-alapú rendszeren is (Mandrake, Yellow Dog stb.).

A SuSE könyvtár szintén tartalmaz egy „openssh.spec" állományt a SuSE-hoz használható prpm OpenSSH csomagok létrehozásához, és egy „rc.sshd" fájlt, amit az /etc/rc.d könyvtárba lehet másolni (jelenleg /sbin/init.d a SuSE-ban). Továbbá tartalmaz egy „rc.config.ssd"-t is, ennek tartalmát a /etc/rc.config fájlhoz kell adni, hogy az rc.sshd parancsfájl helyesen működjön. Ezt egyszerűen meg lehet tenni a következő paranccsal:

```
cat ./rc.config.ssd >>/etc/rc.config
```

Készítsünk egy közvetett hivatkozást (`ln -s`) az rc2.d illetve az rc3.d könyvtárakba, és a SuSE már készen is áll a titkosított héjprogramok

2. táblázat Az sshd_config értékbeállításai

Érték	*Lehetséges értékek*	*Leírás*
Port	*number*	(Alapértelmezett=22) A démon által figyelt TCP-kapuszám. A megváltoztathatóság lehetősége különösen akkor válik hasznossá, amikor kapucímátírást (Port Adress Translation) használunk, hogy több gépet rejthessünk el egyetlen IP-cím mögé.
PermitRootLogin	*yes no*	Elfogadjon-e rendszergazdai bejelentkezést, vagy sem. A leghelyesebb ezt az értéket „No"-ra állítani. A rendszergazdák előnyben nem részesített azonosítóval jelentkezhetnek be, majd su-val léphetnek be rendszergazdaként.
PasswordAuthentication	*yes no*	(Alapértelmezett=yes) engedélyezzen (titkosított) felhasználónév/jelszó belépést, vagy tartson ki az RSA/DSA kulcsalapú azonosítás mellett.
PermitEmptyPasswords	*yes no*	(Alapértelmezett=no) engedélyezzen-e üres jelszóval történő bejelentkezéseket a rendszerbe. Figyelmen kívül hagyható, ha a PasswordAuthentication=no. Ugyanígy nem vonatkozik az RSA/DSA kulcsok jelszavaira sem. (Az üres jelszó a kulcsok esetén elfogadható).
X11Forwarding	*yes no*	(Alapértelmezett=no) engedélyezze-e, hogy az ügyfelek X-windows alkalmazásokat futtassanak ssh csatornákon keresztül. Valójában semmit sem nyerhetünk azzal, hogy no-ra állítjuk, mivel az sshd_config nem képes kikapcsolni az általános TCP továbbítást is egyúttal (ami éppúgy használható X11-továbbításra is).

fogadására! A démon életre kel minden rendszerindításkor, vagy kézzel is indíthatjuk a `/etc/rc.d/rc.sshd start` paranccsal.

SSH mindenkinek, avagy mi ez a „titkosított telnet" dolog

Mi a helyzet, ha csupán interaktív héjprogramokat szeretnénk futtatni távoli rendszereken à la Telnet? Nagy valószínűséggel még a beállítófájlokba sem kell belenézni, elég, ha ennyit begépelünk:

```
ssh távoli.gép.neve
```

Ezután megadjuk a jelszót (az ssh feltételezi, hogy ugyanazt a felhasználónevet szeretnénk használni a távoli gépen is), és ha nem gépeltük el, már használhatjuk is távoli héjprogramunkat! Ez vitathatatlanul egyszerű, és összehasonlíthatatlanul biztonságosabb, mint a telnet. Ha a távoli rendszeren más névvel szeretnénk dolgozni, mint helyi gépen, a -l kapcsolóval adhatjuk meg a kívánt felhasználónevet. Például ha a laptopomon „mick"-ként vagyok bejelentkezve, és szeretnék bejelentkezni ssh-val a kong-fu.mutantmonkeys.org-ra mint „mbauer", a következő parancsot használom:

```
ssh -l mbauer kong-fu.mutantmonkeys.org
```

Mit csinál ez a parancs? Egyáltalán nem látszik másnak, mint a telnet. Rákérdez, használhatja-e a kiszolgáló nyilvános kulcsát vagy sem, esetleg valamivel tovább tart a kapcsolat létrehozása, illetve a rendszer foglaltságától, a hálózattól stb. függően a kapcsolat némileg lassabb lehet, mint a telnet, de a legtöbb esetben nem sok különbséget lehet észrevenni. Viszont úgy jelentkeztem be a rendszerbe, hogy nem küldtem keresztül a hálózaton egyszerű szövegként a jelszavamat és a felhasználónevemet, és az összes adat szintén titkosítva van! Bármit megtehetek, amit csak akarok, akár egy `su` parancsot is kiadhatok, anélkül, hogy leselkedőktől kellene tartanom. Ráadásul mindez csupán egy kevéske lassulásba került!

Turkáljunk a beállítófájlokban!

Az OpenSSH beállítása szintén nem túl bonyolult dolog. A ssh-ügyfél és kiszolgáló viselkedésének irányításához mindössze két állományt kell módosítani: az ssh_config és sshd_config fájlokat. A csomag telepítésétől függően ezek a fájlok a vagy a /etc alatt, vagy a ./configure --sysconfdir értékben megadott könyvtárban találhatók.
Az ssh_config a helyi gépről kezdeményezett ssh kapcsolatok általános beállítóállománya. Ezeket a beállításokat felülbírálhatjuk a parancssori kapcsolók és a felhasználók saját beállítóállományai segítségével (ezek helye a $HOME/.ssh/config). Például ha a /etc/ssh/ssh_config a következő sort tartalmazza:

```
Compression no
```

de a /home/bobo/.ssh/config-ban

```
Compression yes
```

szerepel, akkor valahányszor Bobó ssh-kapcsolatot kezdeményez, a tömörítés alapértelmezés szerint engedélyezett lesz, annak ellenére, hogy minden más felhasználó számára, akinek a $HOME/.ssh/config állománya nem tartalmazza ezt a beállítást, a tömörítés ki lesz kapcsolva. Másrészről viszont, ha Bobó a következő paranccsal indítja az ssh-t:

```
ssh -o Compression=no távoli.gép.neve
```

akkor ebben a kapcsolatban nem alkalmaz tömörítést.

Az ssh_config egy értéklistát tartalmaz, soronként egy értékkel, a következő formátumban: `kapcsoló érték(ek)`
Néhány kapcsoló kétértékű, azaz értéke vagy „yes" vagy „no". Mások viszont értékek vesszővel elválasztott listáját is tartalmazhatják. A legtöbb érték magától értetődő, de mindegyiket részletesen ismerteti az ssh(1) leírása. Az *1. táblázat* bemutat néhányat a legérdekesebbek és legfontosabbak közül (a dőlt betűk lehetséges értékeket jelölnek).
Jó néhány érték használható még ezeken kívül. Néhányat ezek közül a cikk második részében ismertetünk. A teljes lista pedig megtalálható a leírásban.

A Secure Shell Démon, az sshd beállítása és futtatása

Mindez kitűnően működik mindaddig, amíg mások által karbantartott rendszerekhez kapcsolódunk. De még nem beszéltünk arról, miképpen kell beállítanunk saját gépünket, hogy ssh-kapcsolatokat is fogadni tudjunk. Ahogy az már lenni szokott, ez is nagyon egyszerű. Akárcsak az ssh-ügyfél esetében, az sshd alapértelmezett viselkedését is egyetlen fájl, az sshd_config szabályozza, amely szintén vagy a /etc alatt, vagy a ssh beállításakor megadott könyvtárban található. Amint azt már az ssh-ügyfél esetében láttuk, a parancssori kapcsolók felülbírálhatják a beállítófájlban található értékeket. Az ügyféllel ellentétben azonban, a démonnak nincs külön beállítófájlja az egyes felhasználók könyvtáraiban, hiszen az egyszerű felhasználók nem befolyásolhatják a démon viselkedését.
A *2. táblázat* bemutat néhány dolgot azok közül, amelyeket az sshd_config-ban lehet beállítani. Természetesen rengeteg további érték is módosítható itt. Néhányat közülük a következő hónapban bemutatok majd, az eddigiek megértése bőven elegendő az induláshoz (feltételezve, hogy a közvetlen cél a telnet és az ftp kiváltása).

Az scp használata titkosított fájlátvitelhez

Még egy témát tárgyalunk az e havi részben. Az scp parancs, legtöbb működésében azonos a jó öreg rcp eszközzel, amely könyvtárak és fájlok gépek közötti másolására szolgált. (Az igazság az, hogy az scp a rcp forráskódján alapul.) Ha esetleg nem ismerjük egyiket sem, ezek nem interaktív programok: mindegyiket parancssorból kell indítani, amelyben meg kell adni a másolandó anyag és a cél pontos elérési útját.
Emiatt a nem interaktív jelleg miatt az scp kevésbé felhasználóbarát, mint az ftp: akár tetszik, akár nem, az scp használatához bele kell nézzünk a leírásba (vagy egy olyan cikkbe, mint ez), és meg kell jegyezni néhány kapcsolót. Ezzel szemben, mint általában a többi parancssoros eszköz, az scp is sokkal inkább használható parancs-

Kapcsolódó címek

OpenSSH: ➲ http://www.openssh.com

zlib: ➲ ftp://ftp.freesoftware.com/pub/infozip/zlib

SSH információk
➲ http://www.inf.bme.hu/~mulzs/sshfaq/ssh-faq.html
➲ http://www.boran.com/security/ssh_stuff.html

SSH hivatkozások (linkek)
➲ http://www.db.toronto.edu/~djast/ssh.html
➲ http://linuxmafia.com/pub/linux/security/ssh-clients
➲ http://ssh.gatordog.com

SSH FTP tükrök
➲ ftp://ftp.wiretapped.net/pub/security/cryptography/ssh/
➲ ftp://ftp.zedz.net/pub/crypto/ssh/

fájlokban, mint amilyen egy interaktív eszköz lehetne. Az scp „gyorstalpaló" használatát valójában könnyű elsajátítani. Az alapvető formátum a következő:

```
scp [ kapcsolók ] forrásmeghatározás \
        célmeghatározás
```

ahol a forrás és a cél meghatározásai lehetnek egyszerű Unix-típusú fájl- és útvonalnevek (pl.: `./docs/hello.txt`, `/home/me/mydoc.txt` stb.), vagy egy gép és felhasználó azonosítására is alkalmas karaktersorozat:

```
felhasználó@távoli.gép.név:útvonal/fájlnév
```

Például: tegyük fel, hogy a „crueller" gépen dolgozunk, és a „recipe" fájlt szeretnénk átküldeni a „kolach" nevű gépen található saját könyvtárunkba. Továbbá feltételezzük, hogy mindkét gépen azonos felhasználónevet használunk. A folyamat valahogy a következőképpen néz ki:

```
crueller: > scp ./recipe kolach:~
mick@kolach's password: *******
recipe        100%  |****************>|   13226
00:01
crueller: >
```

A parancs begépelése után a jelszót kellett megadnunk, a rendszer feltételezte, hogy azonos nevet használunk a távoli gépen is. Az scp ezután átmásolta a fájlt, miközben egy hasznos kis folyamatjelző sáv mutatja az események előrehaladtát. Ennyi az egész!
Ha a crueller en „mick"-ként dolgozom, a kolachra azonban mbauer-ként szeretném felmásolni a fájlt, a data/pastries könyvtárba, akkor parancssor a következőképpen alakul:

```
crueller: > scp ./recipe \
       mbauer@kolach:~/data/pastries/
```

Most változtassunk egy kicsit. Tételezzük fel, hogy a /etc/oven.conf állományt szeretnénk letölteni a kolachról:

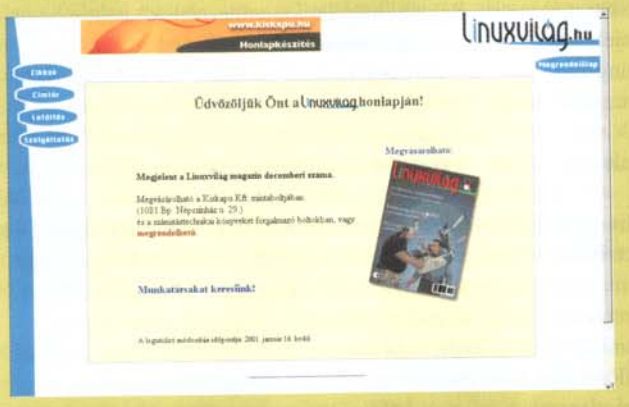
```
crueller: > scp mbauer@kolach:/etc/oven.conf
```

Mindenki érti a lényeget? Az egyetlen fontos dolog, amire emlékezni kell, hogy a forrás megelőzi a célt.
Habár a legtöbb felhasználó az ssh-t és az scp-t mindössze egyszerű bejelentkezésre és fájlátvitelre használja, ez csak a felszíne annak, amire az ssh képes. A következő hónapban azzal foglalkozunk, hogyan lehet az ssh-átvitelt még biztonságosabbá tenni RSA és DSA kulcsok segítségével, miképpen teszik lehetővé a „null-passphrase" kulcsok azt, hogy az ssh parancsok parancsfájlokban is használhatók legyenek, hogyan lehet a bizonyítványokat a memóriában tárolni, hogy elkerüljük a nemkívánatos azonosító kéréseket, és azzal, mi módon lehet keresztülvezetni egyéb TCP-szolgáltatásokat egy titkosított ssh kapcsolaton.

Mick Bauer (mick@visi.com)
alkalmazott biztonsági vezető az ENRGI hálózatmérnöki és tanácsadó cég minneapolisi részlegénél. 1995 óta Linux-rajongó, és 1997 óta vakbuzgó OpenBSD-s. Különös élvezetét leli abban, hogy ezeket az élvonalbeli operációs rendszereket rávegye arra, hogy elavult roncsokon fussanak. Mick szívesen vesz minden kérdést és hozzászólást.

Szakmai tanácsok

Az ssh-keygen programmal hozz létre egy kulcspárt, ezzel megoldhatod, hogy távoli rendszerre bejelentkezéskor ne kelljen jelszót megadni. A program futtatása után másold a .ssh/identity.pub fájt a távoli gépre .ssh/authorized_keys néven.
Nem tudsz bejelentkezni ssh-val? Az ssh jelszót kér, amikor szerinted nem kéne ezt tennie? Használd a `-v` kapcsolót, így böngészheted a bőbeszédű tájékoztató- és hibaüzeneteket.
Állítsd be az RSYNC_RSH változót „ssh"-ra, így minden rsync forgalom titkosítva lesz.
Titkosíthatsz minden POP-forgalmat a géped és a levelező-kiszolgáló között, ha a fetchmailt ssh-n keresztül futtatod. Lásd a Linuxvilág honlapján (➲ www.linuxvilag.hu).



Ha a bash parancssorában meg akarsz találni egy korábban begépelt utasítást, nyomd le a Ctrl-R billentyűkombinációt, majd írj be a keresett parancsból néhány betűt.
Ha le akarod másolni a webhelyed teljes tartalmát, de nincs parancssori hozzáférésed a kiszolgálóhoz, használd a `wget` programot a `--mirror` kapcsolóval.
Debian rendszereken futtathatod a következő cron feladatot éjszakánként: `apt-get update && apt-get -y --download-only dist-upgrade && apt-get autoclean`.
Ez tárolja a frissített csomagokat, így a telepítésük gyorsabb lesz.
A /usr fájlrendszert csak programok telepítése során kell írni.
Mivel az fsck indításkor csak az írható fájlrendszereket ellenőrzi (befűzések száma stb.), ha a /usr-t csak olvashatóra állítod, meggyorsíthatod a rendszerindítást. Emellett, ha a rendszer összeomlik, a /usr-t nem kell majd fsck-zni.
A levelező-kiszolgálód nyílt levéltovábbító (open relay)? Használd a rlytest programot és győződj meg róla, hogy nem az.
Az inetd.conf fájlban tegyél megjegyzésbe mindent, amit nem használsz.
A .Xdefaults fájlban kikapcsolhatod a Netscape-ben a <blink> formázás megjelenítését a következő hozzáadásával:

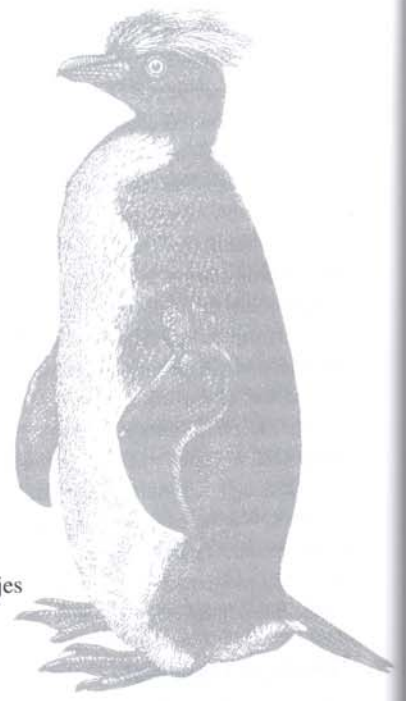
```
blinkingEnabled      False
```

A .netscape/preferences.js fájlból kikapcsolhatod a Netscape „Shop" gombját:

```
user_pref("browser.chrome.disableMyShopp
ing", true)
```

Az OpenSSH száz meg egy előnye (2. rész)

Nézzük meg, hogy még mi mindenre jó az SSH!



A legtöbb SSH-felhasználó nem kerül kapcsolatba a két legegyszerűbb szolgáltatással: a titkosított távoli héjprogrammal és a titkosított fájlátvitellel. Ezt vesztük ki az előző hónapban. Ez eddig rendben is volna. Végül is minek megtanulni olyan dolgokat, amelyeket soha nem használunk. Önképző olvasóink közt azonban minden bizonnyal akad olyan is, aki az SSH megmaradt 99 előnyéből is értékelni tudna valamit. Úgyhogy nézzünk is körül az SSH, és különösképp az OpenSSH igazán hasznos tulajdonságai közt.

Engedjenek meg egy megjegyzést: ebben a cikkben az általános értelemben vett Secure Shellre, azaz „Secure Shell Systemre" az „SSH" kifejezéssel hivatkozom.

Nyilvános kulcsú titkosítás

A nyilvános kulcsú titkosítás (Public Key Criptography) teljes leírása egyszerűen nem férne el egy olyan cikkben, melynek egyébként az OpenSSH-ról kellene szólnia. Ha teljesen ismeretlen ez a terület, ajánlom az RSA Crypto FAQ-t, vagy talán még ennél is jobb forrásként *Bruce Schneier Applied Cryptography* című kitűnő könyvét. Számunkra most elegendő lesz annyit tudni, hogy a nyilvános kulcssémában minden felhasználónak szüksége van egy kulcspárra. A titkos kulcsot (private key) használhatjuk digitális aláírás készítéséhez és a kapott titkosított anyagok visszafejtéséhez. Levelező partnereink a nyilvános kulcsunk (public key) segítségével ellenőrizhetik az állítólag általunk aláírt üzeneteket, illetve ez alapján kódolhatják az üzeneteket, ha azt szeretnék, hogy csak mi olvashassuk azokat. Az *1. ábra* alján láthatjuk, miképpen lehet a két felhasználói kulcspárt használni az egyik felhasználótól a másikhoz érkezett levél hitelesítésére, titkosítására, visszakódolására és ellenőrzésére. Figyeljük meg, hogy Bob és Alice nyilvános kulcsa is jelen van a levelezőtárs gépén, titkos kulcsaikat viszont mind a ketten biztonságos helyen tartják.

Ahogy láthatjuk, az üzenet útja során négy fontos állomást különböztethetünk meg: 1. Bob hitelesíti az üzenetet saját titkos kulcsa segítségével; 2. ezután Bob titkosítja az üzenetet Alice nyilvános kulcsával; (Figyelem: eltekintve attól, hogy Bob megtart leveléből egy másolatot, még ő sem tudja visszakódolni az üzenetet – ezt csakis Alice teheti meg.) 3. Alice megkapja az üzenetet, és visszakódolja a saját titkos kulcsa segítségével; 4. végül Alice Bob nyilvános kulcsát használva ellenőrzi, hogy az üzenetet valóban Bob titkos kódjával hitelesítették-e.

Az olyan tömbkódolásokhoz viszonyítva, mint amilyen az IDEA vagy a blowfish, ahol ugyanazt a kulcsot használják titkosításra és visszakódolásra egyaránt, ez talán kissé csavarosnak tűnhet. A tömbkódolásokkal ellentétben, amelyek esetében a kémkedéstől vagy más támadástól mentes, biztonságos kulcscseréknek kell lehetővé tenni mindkét fél számára a kulcs megszerezését, a nyilvános kulcsokat használó titkosítási rendszert könnyebb biztonságosan használni. Ez azért lehetséges, mert a PK (Public Key) séma szerint úgy küldhetnek a felek üzeneteket egymásnak, hogy előtte semmiféle titkos kódot vagy hasonlót nem kell cserélniük. Egyetlen hátulütője van a dolognak: a nyilvános kulcsokon alapuló eljárások lassabbak és sokkal számításigényesebbek, mint más titkosítási eljárásosztályok, például a tömbkódolások és a folyamkódolások (stream chiper), például a 3DES és az RC4. Történetesen azonban a PK kódolás kitűnően használható olyan biztonságos kulcsok előállítására, amit aztán más eljárásokban lehet felhasználni.

A gyakorlatban a PK titkosítást sűrűn használják azonosításra („Tényleg te vagy?") és kulcsegyeztetésre („Melyik 3DES kulccsal kódoljuk a további adatokat?"), de annál ritkábban a teljes adatfolyam vagy fájlok tömeges kódolásához. Ez a helyzet mind az SSL, mind az SSH esetében.

Magasabb szintű SSH-elmélet: Hogyan használja az SSH a PK titkosítást

A kulcsegyeztetés az egyik legelső dolog, mely akármelyik SSH-ügyletben végbemegy, és ennek köszönhetően lehetségessé válik egy viszonylag gyenge azonosítási módszer (felhasználónév és jelszó) használata. A Diffie–Hellman Kulcscsere Protokoll működése igen bonyolult, és messze meghaladja e cikk kereteit (További részletekért látogassunk el a ➲ http://www.ietf.org/ címre, ahol elolvashatjuk a „draft-ietf-secsh-transport-07.txt" című vázlatot.) Elég annyit tudni, hogy ennek az egész nagy prímszámkavalkádnak a kapcsolatkulcs (session key) lesz az eredménye. Ezt mindkét fél ismeri, de ez ténylegesen nem kerül át az eddig titkosítatlan kapcsolaton keresztül. Minden további csomag adatmezője ezzel a kapcsolatkulccsal lesz titkosítva, a két gép által közösen választott tömbkódolás szerint, átlátszóan, de az adott ssh-folyamat fordítása és beállítása alapján. Többnyire a következő kódolások valamelyike használatos: Triple-DES (3DES), blowfish vagy IDEA. Maga az azonosítás csak a kapcsolat titkosítása után kezdődhet el. Mielőtt még belemerülnénk az RSA/DSA azonosítás rejtelmeibe, álljunk meg egy pillanatra és nézzük meg, miképp lehet a kulcsegyeztetés átlátszó, feltételezve, hogy PK-titkosítást használ, és mint általában, a titkos kulcsok jelszóvédettek. Az SSH két különböző kulcspárt használ: a gépkulcsokat és a felhasználói kulcsokat. A gépkulcs egy különleges kulcspár, amihez nincsen jelszó rendelve. Mivel ezeket jelszó begépelése nélkül is bárki használhatja, az SSH egyeztetni tudja a kulcsokat és a felhasználó számára teljesen átlátszó, titkosított kapcsolatokat állíthat fel. Az SSH-telepítés része a gépkulcs (-pár) készítése. A beállításkor készített gépkulcs az adott gépen azonosítás nélkül használható, biztonsági okokból. Mivel a gépkulcs az adott gépet azonosítja és nem az adott felhasználót, minden gépnek csak egy gépkulcsra van szüksége. Megjegyezzük, hogy gépkulcsot minden SSH-t futtató gép használ, függetlenül attól, hogy futtat-e SSH-ügyfélprogramot, például `ssh` héjat, SSH-démont, vagy mindkettőt.

A felhasználói kulcs természetesen az egyes felhasználókhoz van rendelve, és a felhasználó azonosítására szolgál azon a gépen, amivel kapcsolatot kezdeményezett. A legtöbb felhasználói kulcsot az alkalmazás előtt a megfelelő jelszóval ki kell nyitni.

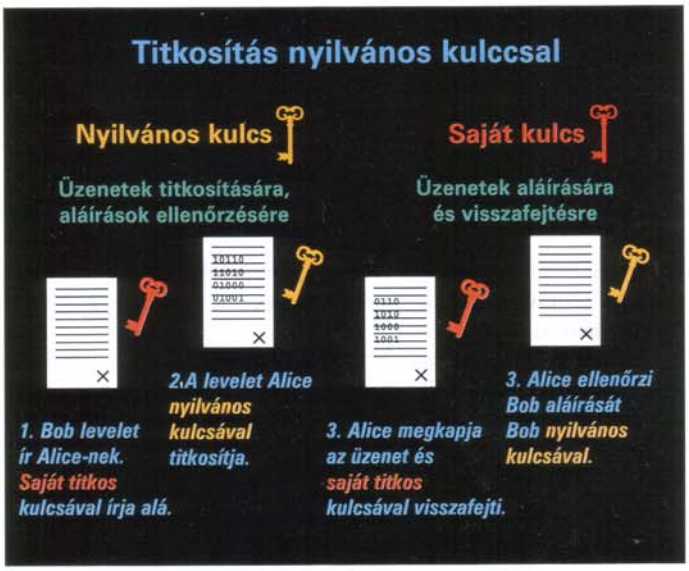
A felhasználói kulcsok biztonságosabb azonosítási eljárást tesznek lehetővé, mint a felhasználónév/jelszó azonosítás (még akkor is, ha minden azonosítás titkosított csatornákon keresztül folyik). Emiatt

alapértelmezés szerint az SSH mindig PK-azonosítást próbál először, és csak ezután tér vissza a felhasználó/jelszó azonosításra.
Amikor elindítjuk az `ssh`-t, az először megnézi a $HOME/.ssh könyvtárunkat, hogy lássa, van-e titkos kulcsunk (`id_dsa` néven). Amennyiben van, a program kérni fogja a kulcs jelszavát, majd a titkos kulcs segítségével készített aláírást a nyilvános kulcsunkkal egyetemben elküldi a távoli kiszolgálónak.
A kiszolgáló ellenőrzi, hogy a nyilvános kulcs engedélyezett-e, azaz jogszerű felhasználóhoz tartozik-e, és mint ilyen, jelen van-e a megfelelő `$HOME/.ssh/authorized_keys2` fájlban. Ha a kulcs engedélyezett, és azonos a kiszolgáló által előzőleg eltárolt másolattal, a kiszolgáló ennek segítségével fogja ellenőrizni, hogy az aláírás ehhez a kulcshoz tartozó nyilvános kulccsal lett-e elkészítve. Ha sikerrel jár, a kiszolgáló engedélyezi a kapcsolatot, esetleg az után, hogy egy felhasználónév/jelszó azonosítást is végrehajt.
(Megjegyzés: a fenti két bekezdés az SSH Protocol v.2 által használt DSA-azonosításra vonatkozik; az RSA-azonosítás valamivel bonyolultabb, de azon kívül, hogy más fájlneveket használ, a felhasználó szemszögéből feladatát azonos az előzővel.)
A PK azonosítás biztonságosabb a felhasználónév/jelszó módszernél, mivel a digitális aláírásokat nem lehet visszafejteni, vagy az előállításukhoz felhasznált titkos kulcsot más módon kikövetkeztetni, még a nyilvános kulcs ismeretében sem. Azáltal, hogy a hálózaton csak digitális aláírásokat és nyilvános kulcsokat küldünk el, biztosíthatjuk, hogy a kémkedő még akkor sem tud elegendő adatot gyűjteni a jogosulatlan bejelentkezéshez, ha a kapcsolatkulcsot valahogy fel is tudná törni.

Beállítások és az RSA-és DSA-azonosítás használata

Rendben. Most már készen állunk arra, hogy saját kulcspár készítésével az ssh-bubusok iskolájában következő osztályába lépjünk. Nézzük, mit kell tennünk.
Elsőként az ügyfélgépen, vagyis azon a gépen, ahol a távoli konzolt szeretnénk majd működtetni, le kell futtatnunk az `ssh-keygen` programot. *(1. lista)* Itt meghatározhatunk pár dolgot: többek közt azt, hogy RSA- vagy DSA-kulcsokat szeretnénk használni, a kulcs hosszát, egy tetszőleges megjegyzésmezőt, a kulcsfájlok nevét és a jelszót – ha akarunk ilyet –, amivel a titkos kulcsok rejtjelezzük. Most, hogy az RSA-szabadalom lejárt, az eljárás kiválasztása tulajdonképpen tetszőleges. Másrészt viszont, az IETF-hez internetszabvány-ajánlásként elküldött SSH Protocol v.2 DSA rendszerű kulcsokat használ. Ha mindez nem igazán fontos kérdés olvasóink számára, én a DSA-t javaslom. De ha valaki mégis RSA-t kedveli, nyugodtan használhatjuk azt is. A `-d` kapcsoló állítja be a DSA-eljárást, az alapértelmezés ugyanis az RSA.
A kulcshossz már fontosabb jellemző. *Adi Shamir* Twinkle című írásában ismertet egy csak elméletben létező, de megvalósíthatónak tűnő számítógépet, melyet az 512 bitnél rövidebb RSA/DSA-kulcsok nyers erővel (brute force) történő feltörésére lehetne használni, ennek fényében én az 1024 bites kulcsok használatát ajánlom. A 768 is rendben lenne, de nem észrevehetően gyorsabb, mint az 1024. A 2048 viszont egyértelműen kész halál: nem sokkal biztonságosabb, viszont mindent érezhetően lelassít. Az alapértelmezett kulcshossz az 1024, de a `-b` kapcsolót követő számmal más értéket is megadhatunk.
A comment (megjegyzés) mezőt egyik ssh-folyamat sem használja: szigorúan csak a mi kedvünkért létezik. Én általában a helyi levélcímemet írom oda. Ezáltal, ha megtalálom a kulcsot valamely másik rendszerem `authorized_keys` (jogosult kulcsok) fájljai között, legalább tudom, honnan érkezett. A megjegyzéseket a `-C` kapcsolóval adhatjuk meg.
A jelszó és a fájlnév megadható a `-N8` és a `-f` kapcsolókkal, de nem kötelezőek a parancssorban. Ha nincsenek megadva, a program úgyis rákérdez.

Az első listában egy 1024 bit hosszú DSA-kulcspárt készítek, megjegyzésként pedig az mbauer@sprecher.enrgi.com címet adom meg. Az `ssh-keygen`-re bízom, hogy megkérdezze, milyen fájlba mentse a kulcsokat. Ez lesz a titkos kulcs neve. A nyilvános kulcs neve ugyanez lesz, de a végére a .pub kiterjesztés kerül. Ebben a példában elfogadtam az alapértelmezett id_dsa, és így az id_dsa.pub fájlnevet. Szintén az `ssh-keygen`-re hagytam, hogy a jelszót megkérdezze. A csillagsorozat (**********) valójában nem fog megjelenni a képernyőn, csak azért szúrtam be a példába, hogy jelöljem, egy hosszú jelszót gépeltem be, ami nem jelenik meg a képernyőn.



1. ábra Nyilvános kulcsú titkosítás

Egyébként a jelszavak mindent vagy semmit alapon adhatók meg. A jelszavunk lehet üres is, ha az új kulcsot gépkulcsként vagy SSH-t tartalmazó parancsfájlokban akarjuk használni, vagy pedig egy hosszú, nagy- és kisbetűket, számokat és központozást tartalmazó karaktersorozatnak kell lennie. Nem olyan bonyolult, mint amilyennek hangzik. Például egy dalszöveg sora szándékos, de kiszámíthatatlan félregépelésekkel könnyen megjegyezhető, de nehezen kitalálható. Minél inkább véletlenszerű egy jelszó, annál erősebb.
Ez minden, amit az ügyféloldalon tenni kell. Az egyetlen dolog, amire szükségünk van a távoli gépen – ahová erről az ügyfélről szeretnénk belépni –, hogy egy új nyilvános kulcsot helyezzünk a `$HOME/.ssh/authorized_keys2` fájlba, itt a $HOME a saját könyvtárunk elérési útja. Az `authorized_keys2` egy nyilvános kulcsokat tartalmazó lista, minden igen hosszú sorában egy bejegyzéssel, amit a könyvtárat birtokló felhasználó bejelentkezéséhez lehet felhasználni.
Ha szeretnénk eljuttatni a nyilvános kulcsunkat ahhoz a távoli géphez, amin jogosultságunk van, egyszerűen küldjük át a nyilvános kulcsunkat tartalmazó fájlt (a fenti példában `id_dsa.pub`) a távoli gépre, és toldjuk az `authorized_keys2` fájl végéhez. Hogy miképpen szerezzük meg az állományt, az nem igazán számít. Emlékezzünk, ez a nyilvános kulcsunk, így ha egy kukucskáló le is másolná is útközben, akkor se történne semmi. Ha azonban van némi üldözési kényszerünk, egyszerűen írjuk be a `scp ./id_dsa.pub` *`távoligépnév:/saját-könyvtár`* parancsot – miként azt a múlt havi írásunkból már megismerhették. Azután, hogy hozzáadhassuk az `authorized_keys2` fájlhoz, jelentkezzünk be a távoli gépre és gépeljük be:

```
cat id_dsa.pub >> .ssh/authorized_keys2
(feltételezve, hogy a saját könyvtárunkban
vagyunk.)
```

1. lista DSA-kulcspárok készítése

```
mbauer@homebox:~/.ssh > ssh-keygen -d -b 1024
    ➥-C mbauer@homebox.pinheads.com
Generating DSA parameter and key.
Enter file in which to save the key
(/home/mbauer/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again: **********
Your identification has been saved in
/home/mbauer/.ssh/id_dsa.
Your public key has been saved in
/home/mbauer/.ssh/id_dsa.pub.
The key fingerprint is:
95:a9:6f:20:f0:e8:43:36:f2:86:d0:1b:47:e4:00:6e
mbauer@homebox.pinheads.com
```

Ennyi! Mostantól ahányszor SSH-val jelentkezünk be a távoli gépre, a kapcsolat valahogy olyasféleképpen fog kinézni, mint amit a *2. listában* láthatunk.
Figyeljék meg, hogy amikor az `ssh`-t meghívom, a `-2` kapcsolót használom, ez ugyanis arra utasítja az SSH-t, hogy csak az SSH Protocol v.2 módszert használja. Alapértelmezés szerint a Protocol v.1 módszert alkalmazza, de az csak az RSA-kulcsokat támogatja, holott mi DSA-kulcsokat másoltunk át. Azt is érdemes megfigyelni, hogy a kulcsra csak helyi elérési úttal/fájlnévvel hivatkozom: ez jó példa arra, hogy amikor RSA- vagy DSA-azonosítást használunk, az általunk begépelt jelszó csak a helyileg tárolt titkos kulcsunk kinyitásához szükséges, és semmilyen formában nem továbbítódik a hálózaton.
A fenti egyszerű példához csak egyetlen megjegyzést szeretnék még hozzáfűzni. Két dolgot tételeztünk fel a távoli gépről: 1. ugyanaz az ottani felhasználói nevünk, mint az itteni és 2. a távoli kiszolgáló ismeri az SSH Protocol v.2-t. Ha az első feltétel nem teljesül, a `-l` kapcsolóval megadhatjuk a távoli géphez tartozó felhasználónevet (avagy a `-l` nélkül, az scp-stílusú felhasználónév@gépnév írásmódot használva, például mick@zippy.pinheads.com).
Ha a távoli gép sshd-démonja nem támogatja a Protocol v.2-t, akkor újra kell próbálkozni a `-2` kapcsoló nélkül, és hagyni, hogy az SSH visszatérjen a név/jelszó azonosításhoz, hacsak nincs egy RSA-kulcspárunk, melynek nyilvános része be van jegyezve a távoli gépen.
A fentieket RSA-kulcsokkal végezve, ugyanezeket a lépéseket kell megtennünk, csak más fájlnevekkel:

1. RSA felhasználói kulcspár készítése az ssh-keygen segítségével, például

```
ssh-keygen -b 1024 -C mbauer@homebox.pinheads.com
```

2. Minden távoli gépre, ahova csak csatlakozni szeretnénk, fel kell másolni a nyilvános kulcsunkat a `$HOME/.ssh` könyvtárban található `authorized_keys` fájl egy külön sorába. Az RSA-kulcsok alapértelmezett fájlnevei az identity és az identity.pub.

Még egyszer, ha az `ssh`-t a `-2` kapcsoló nélkül futtatjuk, akkor alapértelmezés szerint RSA-azonosítást használ.
Mi történik, ha elfelejtjük az RSA- vagy DSA-kulcshoz használt jelszavunkat? Hogyan fogunk visszakerülni a távoli gépre, és mi módon tudjuk megváltoztatni az immár használhatatlanná vált kulcsot az authorized_keys fájlban? Nem kell aggódni: amennyiben nem sikerül belépni RSA/DSA azonosítással, az SSH automatikusan visszavált név/jelszó azonosításra, és rákérdez a távoli gépen megadott jelszavunkra. Ha ezt a „visszaváltás" lehetőséget szeretnénk kikapcsolni, és kizárólag az RSA/DSA bejelentkezéseket akarjuk megengedni, akkor az `sshd_config`-ban található `PasswordAuthentication` értéket változtassuk „no"-ra, minden távoli gépen, ahol `sshd` fut. Emlékezzünk rá, hogy amikor a dolgok kiszolgálóoldaláról beszélünk, az sshd az RSA és a DSA azonosítást egyaránt engedélyezi, ha egy ssh-ügyfél ilyen kérelemmel fordul hozzá. Két értékkel közvetlenül engedélyezhetjük, illetve tilthatjuk a protokollokat, ezek az `RSAAuthentication` és a `DSAAthentication`.

2. lista Bejelentkezés DSA-kulcsok segítségével

```
bauer@homebox:~/ > ssh -2 zippy.pinheads.com
Enter passphrase for DSA key
'/home/mbauer/.ssh/id_dsa':
Last login: Wed Oct  4 10:14:34 2000 from
homebox.pinheads.com
Have a lot of fun...
mbauer@zippy:~ > _
```

Egy vagy több felhasználói kulcs erősebbé teszi az azonosítási biztonságot, és jobban kihasználja az SSH által nyújtott lehetőségeket, mint a név/jelszó azonosítás. Emellett ez az első lépés afelé, hogy az SSH-t parancsfájlokban használjuk. Egyetlen gond jelentkezhet a PK-titkosítás automatizálásával kapcsolatban: bár a PK-alapú azonosítás átlátszó, a megelőző kulcsazonosítás nem az.
Hogyan tudnánk biztonságosan átlépni, vagy egyszerűsíteni ezt a folyamatot?

Erős azonosítás egyszerűsítése az ssh-agent segítségével

Több lehetőség is van: az egyik, hogy jelszó nélküli kulcsokat készítünk, így senkinek sem kell jelszót begépelnie, amikor a kulcsot használja, a másik pedig az `ssh-agent` használata.
Ez utóbbi tulajdonképpen egy saját titkos kulcsgyorstár a memóriában, mely lehetővé teszi, hogy ismételten használhassuk a kulcsot, ha egyszer már begépeltük a jelszavunkat. El kell indítani az `ssh-agent`-et, majd be kell tölteni a kulcsot az `ssh-add` paranccsal, végül meg kell adni a kulcshoz tartozó jelszavunkat. Az ily módon „kinyitott" titkos kulcs a memóriában marad, így minden további `scp`- vagy `ssh`-hívás képes lesz használni a gyorstárazott, kinyitott kulcsot, a jelszó újrakérdezése nélkül is.
Ez nem tűnik túl biztonságosnak, pedig az. Először is, csak az ssh-agent folyamat tulajdonosa használhatja a betöltött kulcsokat. Például ha „root" és „bubba" bejelentkeznek, és mindketten elindították a saját `ssh-agent` folyamatukat, valamint betöltötték a megfelelő titkos kulcsaikat, akkor sem tudnak egymás gyorstárazott kulcsaihoz hozzáférni, nem áll fenn annak a veszélye, hogy „bubba" a rendszergazda azonosítóit használja fel `scp` vagy `ssh` folyamatokhoz.
Másodszor, az `ssh-agent` csak helyi `ssh`- és `scp`-hívásokra válaszol, a hálózatról közvetlenül nem érhető el. Más szavakkal, ez egy helyi szolgáltatás, ezért nem fordulhat elő az sem, hogy egy külső, behatolni próbáló személy ellopjon, vagy valamely módon megváltoztasson egy távoli `ssh-agent` folyamatot.
Az `ssh-agent` használata meglehetősen magától értetődő: egyszerűen gépeljük be az `ssh-agent` parancsot, majd hajtsuk végre a képernyőn megjelenő utasításokat. Ez utóbbi talán kicsit zavarosnak tűnhet, és valószínűleg nem túl egyértelmű: az `ssh-agent`, mielőtt a háttérbe vonulna, rövid, az általunk futtatott parancsértelmezőnek megfelelő környezeti változóra vonatkozó meghatározást ír a képernyőre, amit végre kell hajtanunk, mielőtt bármilyen kulcsot betölthetnénk. A végrehajtáshoz egyszerűen jelöljük ki ezeket a parancsokat az egérrel, majd jobb gombbal másoljuk őket a parancssorba (lásd a *3. listát*).

3. lista Az ssh-agent indítása

```
mbauer@pinheads:~ > # Megjegyzés: ezek itt
fordított aposztrófok.
mbauer@pinheads:~ > eval `ssh-agent`
Agent pid 11518
mbauer@pinheads:~ > _
```

4. lista Cat futtatása a távoli gépen

```
mbauer@homebox > ssh mbauer@zippy.pinheads.com \
    cat /var/log/messages | more
Oct  5 16:00:01 zippy newsyslog[64]: logfile
turned over
Oct  5 16:00:02 zippy syslogd: restart
Oct  5 16:00:21 zippy ipmon[29322]:
16:00:20.496063
ep0 @10:1 p 192.168.1.103,33247 -> 10.1.1.77,53
PR
udp len 20 61  K-S K-F

satöbbi...
```

5. lista Xterm átirányítása a távoli gépről

```
mick@homebox:~/ > ssh -2
mbauer@zippy.pinheads.com
Enter passphrase for DSA key
'/home/mick/.ssh/id_dsa':
Last login: Wed Oct  4 10:14:34 2000 from
homebox.pinheads.com
Have a lot of fun...
mbauer@zippy:~ > xterm &
```

6. lista FTP átirányítás ssh segítségével

```
mick@homebox:~/ > ssh -2 -f mbauer@zippy -L \
7777:zippy:21 sleep 20
Enter passphrase for DSA key
'/home/mick/.ssh/id_dsa':
mick@homebox:~/ > ncftp -p 7777 localhost
```

A *3. listában* az út harmadánál járok: elindítottam az `ssh-agent` folyamatot, és az `ssh-agent` kinyomtatta a BASH írásmód szerinti változómeghatározásokat.
Megjegyzem, hogy a kivágás és másolás minden xterm alatt is működik, a tty (szöveges) konzol alatti működéshez futnia kell a `gpm`-nek. Ha minden más csődöt mond, még mindig begépelhetjük a parancsokat. Ha az `ssh-agent` már fut, az SSH_AUTH_SOCK és az SSH_AGENT_PID pedig meghatározva és exportálva van, akkor eljött az idő, hogy betöltsük a titkos kulcsunkat. Egyszerűen csak gépeljük be az `ssh-add` parancsot, egy szóközt, majd pedig a betölteni kívánt titkos kulcs nevét a teljes elérési úttal. Ha nem adunk meg fájlt, akkor a program automatikusan megpróbálja betölteni a $HOME/.ssh/identity-t. Ez azonban az RSA-azonosításhoz rendelt alapértelmezett felhasználói titkos kulcs neve, amennyiben a kulcsunkat másképpen hívják, vagy DSA-kulcsot használunk, meg kell adnunk a nevet, mégpedig teljes elérési úttal, azaz például:

```
ssh-add /home/mbauer/.ssh/id_dsa.
```

Az ssh-add programot annyiszor futtathatjuk (annyi kulcsot tölthetünk be), ahányszor csak akarjuk. Ez hasznos lehet, ha RSA- és DSA-kulcspárokat is használunk, és különböző SSH-változatokat futtató távoli gazdagépeket érünk el akár csak RSA-kulcsokat, akár DSA-kulcsokat egyaránt támogatnak.

Jelszó nélküli kulcsok a héjprogramok számára

Az `ssh-agent` hasznos lehet, ha bejelentkezéskor indítunk parancsfájlokat, vagy többször kell az `ssh`-t, illetve az `scp`-t futtatnunk. De mi a helyzet a `cron` munkákkal? Nyilvánvaló, hogy a `cron` nem tud név/jelszó választ adni, vagy begépelni a jelszót a PK-azonosításhoz.
Ez az a hely, ahol jelszó nélküli kulcspárokat kell használni. Egyszerűen futtassuk le az `ssh-keygen`-t a fentiek szerint, de jelszó begépelése helyett üssünk ENTER billentyűt. Valamilyen fájlnevet is begépelhetünk az alapértelmezett „identitiy" vagy „id_dsa" helyett, ha ezt a kulcspárt nem alapértelmezett felhasználói kulcspárnak szánjuk valamilyen automatizált feladatokat futtató különleges azonosítóhoz. A `-i` kapcsoló használatával megadhatunk egy bizonyos kulcsot az `ssh` és `scp` folyamatokhoz. Például ha `scp`-t használok egy `cron` munkában, mely naplófájlokat másol, az `scp` sor valahogy így nézne ki:

```
scp -i /etc/script_dsa_id /var/log/messages.*
      ➥scriptboy@archive.g33kz.org
```

Mikor a parancsfájl fut, ez a sor jelszókérés nélkül hajtódik végre: ha a jelszó értéke ENTER, az SSH elég okos ahhoz, hogy ne háborgassa feleslegesen a felhasználót.
Emlékezzünk azonban arra, hogy a távoli gép oldalán a `/etc/script_dsa_id.pub`-ban rejtőző kulcsot a megfelelő `authorized_keys2` fájlba, jelen esetben a `/home/scriptboy/.ssh/authorized_keys2` állományba kell másolni.
Figyelem! Mindig védjük az összes titkos kulcsunkat. Ha kétségeink vannak, hajtsunk végre egy `chmod go-rwx titkos_kulcs_fájl` parancsot.

Távoli parancsvégrehajtás SSH használatával

Itt az ideje, hogy visszatérjünk ebből a PK-varázslásból, és megnézzünk egy egyszerű, de a parancsfájl-készítéshez elengedhetetlen SSH-képességet: a távoli parancsokat. Mindeddig az `ssh` parancsot kizárólag héjprogramok indítására használtuk. Azonban ez csak az alapértelmezett viselkedése, ha ugyanis az `ssh`-t úgy indítjuk, hogy utolsó értékként egy parancsot adunk meg, akkor az SSH a megadott parancsot fogja végrehajtani a távoli gépen a héjprogram helyett.
Tegyük fel, hogy egy gyors pillantást szeretnék vetni a távoli rendszer naplójára, ahogy azt a *4. listában* megfigyelhetjük. Amint látható, a „zippy" nevű gép visszaküldi a `/var/log/messages` állományának tartalmát a konzolomra. Figyeljük meg, hogy a kimenetet átirányította a helyi gépre, további feldolgozás végett.
Két dologra hívnám fel a figyelmet: 1. A további felhasználói közbeavatkozást igénylő programok futtatása trükkös megoldást igényel, ezért kerülendő. A héjprogram kivételével, az `ssh` ugyanis akkor működik a legjobban, ha felhasználói bemenetet nem igénylő programokat futtat. 2. Minden azonosítási szabály továbbra is érvényes:

ha egyébként jelszót kellene megadni, továbbra is meg kell tenni. Ezért, ha az SSH-t `cron` munkából vagy más, nem interaktív környezetből indítjuk, bizonyosodjunk meg róla, hogy jelszó nélküli kulcsokat használunk, illetve betöltöttük a kulcsokat az `ssh-agent`-tel.
Mielőtt befejeznénk az SSH a parancsfájlokban témakör tárgyalását, hanyag lennék, ha nem ejtenék pár szót az „rhosts" és az „shosts" azonosításról. Ezek azok a módszerek, amelyek révén a belépés automatikusan engedélyezett minden olyan felhasználó számára, aki a következő fájlokban meghatározott gépekről lép be: `$HOME/.rhosts`, `$HOME/.shosts`, `/etc/hosts.equiv`, és `/etc/shosts.equiv`.
Gondolom, mindenki kitalálta, hogy az rhosts-elérés elképesztő módon nem biztonságos, mivel teljes egészében a forrás IP-címében és a gépnevekben bízik, ezeket pedig igen sokféleképpen lehet hamisítani. Ezért aztán, az rhosts-azonosítás alaphelyzetben le van tiltva. Az shosts már más, annak ellenére, hogy hasonlóképpen viselkedik, mint az rhosts a kapcsolódó gép azonosságának ellenőrzése itt gépkulcsellenőrzéssel történik. Továbbá az shosts eljáráson keresztül csak a fellépni szándékozó gép rendszergazdája képes átlátszóan kapcsolódni.
Egyébként, az rhosts eléréseket RSA- vagy DSA-azonosítással egyesíteni nagyon hasznos móka, különösen, ha jelszó nélküli kulcsokat használunk. Az shosts és rhost PK azonosítással vagy anélkül történő, SSH alatti használatával foglalkozó további adatokért tekintsük meg a `sshd(8)` leírását.

TCP-kaputovábbítás SSH-val: VPN a tömegeknek!

Íme, megérkeztünk a végkifejlethez (és ahhoz a részhez, amihez az rhosts/shosts teljesebb megvitatásának feláldozása árán mentettem helyet): a kapuátirányítás témájához. Az `ssh` lehetőséget ad a távoli bejelentkezésre és parancsvégrehajtásra, `scp`-vel pedig megoldható a fájlmásolás. De mi a helyzet az X-szel, a POP3-mal és az FTP-vel? Nem kell aggódnunk, az SSH ezeket is, sőt, a legtöbb TCP-alapú szolgáltatást biztonságossá tudja tenni!
Az X-alkalmazások távoli konzolunkra továbbítása kivételesen egyszerű. Először a távoli gépen szerkesszük át a `/etc/ssh/sshd_config` fájlt és állítsuk az X11Forwarding értéket „yes"-re (az OpenSSH 2x változatban az alapértelmezett érték a „no"), második lépésként létesítsünk ssh-kapcsolatot a helyi konzolról a távoli gépre, a megfelelő azonosítási módszerrel, és végül futtassunk olyan X alkalmazást, amilyet csak akarunk. Ennyi az egész! Mondanom sem kell (remélem), a helyi rendszeren X-nek kell futnia. Ha fut, a távoli alkalmazás minden X kimenetet a helyi munkafelületre irányít át.
Miután az `xterm &` parancsot kiadjuk, egy új xterm ablak jelenik meg a helyi gépen. Ugyanilyen könnyedén futtathatunk Netscape-et, GIMP-et vagy bármi mást, amit csak a helyi X-kiszolgálónk kezelni képes, és telepítve van a távoli gépen.
Az X11 az egyetlen szolgáltatáscsoport, melynek automatikus továbbítására az SSH-t közvetlenül felkészítették. A többi szolgáltatás is könnyen továbbítható a `-L` kapcsolóval. Figyeljük meg a *6. listát*! Az `ssh`-sor első része már ismerős: SSH Protocol v.2-t használok és más felhasználónévvel (mbauer) jelentkezek be a távoli gazdagépre (zippy). A `-f` kapcsoló azt jelzi az `ssh`-nak, hogy a háttérben fusson (fork background), miután elindította az utolsó értékként kapott parancsot, mely jelen esetben `sleep 20`. Ez azt jelenti, hogy az `ssh` 20 másodpercig fog aludni ahelyett, hogy egy héjprogramot indítana el.
Húsz másodperc bőven elegendő, hogy elindítsuk a csatornában futó (tunneled) FTP-kapcsolatot, ez a `-L` kapcsolót követő varázslás következtében jön létre. A `-L` helyi továbbítást (local forward) jelöl: egy átirányítást az ügyfélrendszerünkön található helyi TCP-kapuról a kiszolgálórendszer távoli kapujára. A helyi továbbítás a következő írásmódot követi: helyi_kapu_szám:távoli_gépnév:távoli_kapu_szám ahol a helyi_kapu_szám tetszőleges kapu a helyi gépen, a távoli_gépnév a távoli kiszolgáló neve vagy IP-címe, a távoli_kapu_szám pedig a távoli gép azon kapuja, ahová a kapcsolatot továbbítani akarjuk. Megjegyzem, `ssh`-val bármely felhasználó létrehozhat helyi átirányítást magas (1024-nél nagyobb) kapuszámokon, de csak a rendszergazda engedheti meg nekik, hogy kivételes (1024-nél kisebb) kapukat használjanak. A fenti példánál maradva, miután az `ssh` „elalszik" visszatértünk a helyi héjprogramunkba, és 20 másodpercünk van arra, hogy felépítsük az FTP-kapcsolatot. Megfigyelhető, hogy a 6. listában `ncftp`-t használok: ennek az az oka, hogy az `ncftp` támogatja a `-p` kapcsolót, amivel rávehetem a 7777-es helyi átirányított kapuhoz való csatlakozásra. Az is látható, hogy az `ncftp`-nek a saját rendszerem nevét adtam meg a távoli gép neve helyett, ugyanis a kapcsolatot az `ssh` fogja átirányítani a távoli helyre.
Húsz másodperc múltán, az ssh-folyamat megpróbál leállni, de mivel a helyi átirányítást használó FTP-kapcsolat még mindig működik, az `ssh` egy üzenetet küld erről, és élő marad mindaddig, míg az átirányított kapcsolat le nem zárul. Semmi sem gátolhat meg bennünket abban, hogy egy bejelentkező héjprogramot indítsunk távoli parancs helyett (egyszerűen csak el kell hagynunk a `-f` kapcsolót, majd egy másik virtuális konzolon keresztül elindíthatjuk az FTP-t stb.). Ha a `-f` kapcsolót és a `sleep` parancsot használjuk, akkor sem vagyunk pontosan 20 másodperces várakozásra kárhoztatva – az alvási időtartam lényegtelen (mindaddig, amíg elég időnk van elindítani az átirányított kapcsolatot).
Így aztán, bármilyen távoli parancsot futtathatunk, ami előidézi a megfelelő szünetet, de logikus dolog a `sleep`-et használni, hiszen pontosan ilyesmire való. Még egy ötlet: ha egy adott helyi átirányítást minden ssh-kapcsolatnál használni akarunk, akkor megadhatjuk a munkakönyvtárunkban található beállítófájlban (`$HOME/.ssh/config`). Az írásmód a `-L` kapcsolóéhoz hasonló:

```
LocalForward 7777 zippy.pinheads.com:21
```

Kicsit bővebben: a „LocalForward" értéknév után egy szóköz vagy egy TAB következik, majd a helyi kapuszám, újabb szóköz, a távoli gép neve vagy IP-címe, kettőspont szóköz nélkül, végül a távoli kapu száma követi. Ugyanezt az értéket a `/etc/ssh/ssh_config` fájlban is használhatjuk, ha azt szeretnénk, hogy valamennyi ssh-folyamatra érvényes legyen.
Nos, ezek lettek volna az SSH és az OpenSSH magasabb szintű alkalmazásai. Most már el kell köszönnöm, a további részleteket és újabb képességeket olvasóinknak kell kikeresniük a leírásokból. Ne feledkezzenek meg a titkosításról!

Mick Bauer (mick@visi.com)
az ENRGI hálózatmérnöki és tanácsadó cég minneapoloisi részlegének biztonsági vezetője. 1995 óta Linux-rajongó, és 1997 óta az OpenBSD megszállottja. Különös élvezetét leli abban, hogy ezeket az élvonalbeli operációs rendszereket rávegye arra, hogy elavult roncsokon fussanak. Mick szívesen fogad minden kérdést és hozzászólást.

Kapcsolódó cím

RSA Crypto FAQ
➲ http://www.rsasecurity/rsalabs/faq/