# Szerverek távoli karbantartása szabad szoftveres eszközökkel – Távoli management

*Szabad szoftver keretrendszer*

Készítette a Közigazgatási és Igazságügyi minisztérium E-közigazgatási Szabad Szoftver Kompetencia Központja, Budapest, 2013

# Tartalomjegyzék



# Történeti áttekintés

A nagygépes UNIX idők kezdetéből (kb. 70-es évek eleje) származik az első olyan távoli karbantartásra szánt eszköz, amely segítségével a rendszergazda úgy

volt képes dolgozni helyszíntől függetlenül, mint ha ott a gép előtt tevékenykedett volna. Ennek a protokollnak a neve Telnet[1], amely az RFC 854-es szabvány leírásban van pontosan részletezve. A kornak megfelelő szemléletmód alapján lett kialakítva. Nevezetesen abból indultak ki, hogy aki képes kezelni egy ilyen eszközt és rendelkezik a megfelelő azonosítóval, az a tudását pozitív irányban fogja kifejteni. Talán ezért (vagy mivel akkoriban nem volt tömeges igény a biztonságos hálózati kommunikációra, hiszen akik nagygépes rendszereken dolgoztak, azok jóformán ismerték egymást személyesen), nem építettek titkosítási réteget a telnet protokollba. Igaz, abban az időben elterjedt titkosítási réteg nem is igazán volt. Összefoglalva, a telnet egy flexibilis eszköz, amely a terminál típusoktól független módon TCP-n keresztül kommunikálva kötötte össze kétirányú kommunikációval a gépeket. A mai napig majd minden Linux terjesztés alap részét képezi, mivel például hálózati protokoll tesztelésre (http, pop3 és sok más) kiválóan alkalmas. Ma azonban már egy magára valamit adó üzemeltető tiltja a gépei telnettel való távoli elérését,

[1] http://simple.wikipedia.org/wiki/Telnet

azaz nem futtat telnet-kiszolgálót (telnetd, azaz telnet „démon”).

Manapság inkább olyan kontextusban kerül a szerver oldali telnet szóba, hogy például Nmap[2] segítségével megvizsgáljuk, nem maradt-e az alap telepítés része a telnetd. Bár igaz, hogy gyakorlatilag nincs olyan Linux-terjesztés[3], amely a szerver részt alapból telepített állapotban tartalmazná.

A telnet múltjából és jelenéből jól látható, hogy ma már távoli hozzáférésnél alapkövetelmény a titkosított közegben való kommunikáció és a jogosultságkezelés, hiszen gyakorlatilag akinek távolról helyi üzemeltetői (root) joga van a gépre, az majdnem olyan jogosultságokkal rendelkezik, mintha a gép előtt ülne közvetlenül (eltérés például, hogy egyéb eszköz híján, a rendszerindítási folyamatba nem tudunk beavatkozni, illetve bizonyos hardverhez kapcsolódó műveleteket nem hajthatunk végre – CD/DVD-csere például).

[2] http://nmap.org/

[3] A BSD-ken alapból telepítve van, persze nincs engedélyezve a futása.

# SSH

A megoldás egészen 1995-ig váratott magára, amikor is az SSH Communications Security-t a finn Tatu Ylönen megalapította, és létrehozta az első SSH (secure shell) szerver-kliens párost. Lerakták az alapjait az SSH szabványnak. Természetesen ahogy lenni szokott, a piac gyorsan reagált az SSH megjelenésére, és hamarosan számtalan fejlesztőcég alkotta meg a maga verzióját. Ma már teljesen általános, hogy az OpenSSH[4] implementációját használja minden disztribúció, mivel tudásában kiemelkedő. Az OpenSSH projectet Theo de Raadt[5] vezeti és alapította, aki számos más, főleg BSD operációs rendszerrel kapcsolatos fejlesztésben is vezető.

Az SSH alap eszközévé vált az üzemeltetéssel foglalkozó emberek számára, ugyanis rengeteg adminisztrációs feladat vagy eleve csak karakteresen végezhető el, vagy pedig sokkal gyorsabban megoldható szinte bárhonnan ezen a módon. Köszönhetően a tömeges felhasználásnak, a legtöbb forgalomban lévő okostelefon,

[4] http://en.wikipedia.org/wiki/OpenSSH
[5] http://en.wikipedia.org/wiki/Theo_de_Raadt

vagy egyéb olyan eszköz, amely alkalmas lehet admin funkciók végrehajtására, rendelkezik futtatható SSH kliens programmal. Azaz például Android, iOS vagy akár Windows Mobile operációs rendszert futtató telefonokról is beléphetünk vele távolról gépünkre (ugyanígy régebben a Symbian operációs rendszerekre is számos kliens létezett és még létezik is.). Az egyik legnépszerűbb windowsos SSH kliens a Putty-csomag[6], amely MIT licenc alatt kerül kiadásra (gyakorlatilag a BSD licenccel egyező feltételek mellett). Ez látható a letöltési oldalon is a számtalan platformra fordított bináris letölthetőségéből[7]. Tudása olyan komplex, hogy egy átlagos képességű rendszergazda is csak 5-10%-nyi funkcióját használja ki. Legtöbb esetben a rendszergazda nyit egy titkosított shellt vagy az SSH csomag részét képező SCP kliens segítségével titkosított fájlátvitelt valósít meg. Az OpenSSH weboldalán[8] szereplő előnyök közül

[6] http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[7] Kevéssé ismert, hogy UNIX, Linux alá is lefordítható; sok Linux disztribúció (például Ubuntu), és a különböző BSD-rendszerek alá is elérhető bináris csomagként

[8] http://www.openssh.com/features.html

kiemelhető, hogy nem csak erős titkosítás valósítható meg például RSA vagy DSA kulcspárok létrehozásával, hanem OTP azonosítás[9], port továbbítása titkosított alagútban, adattömörítés az átvitt adatok esetében automatikusan (egy mobil telefonos kapcsolat esetében – például GPRS kapcsolat – nem mindegy, hogy egy egész képernyő képét kell állandóan le-fel töltögetnünk, vagy csak egy tömörített pár soros szöveget viszünk-e át) és még számtalan olyan előny sorolható fel mellette, amely alapeszközzé teszi a rendszer-üzemeltetésben.

## Fontosabb ajánlható használati módozatok

Alapértelmezés szerint az SSH-szerver konfigurációját az /etc/ssh/sshd_config alatt találjuk a legtöbb disztribúcióban. Ha forrásból telepítettük, akkor pedig ahova beállítottuk (az ssh telepítése forrásból általánosságban nem javasolható, támaszkodjunk a csomag kezelőre). Az alapkonfiguráció a PAM-alapú azonosítást és a legtöbb esetben a root alapú ssh bejelentkezés lehetőségét is meghagyja nekünk. Mint jellemzően minden szerver konfig alap kiindulási konfigurációja, ez is a

[9] http://en.wikipedia.org/wiki/One-time_password

működésre van specifikálva, nem pedig a teljes biztonságra. Érdemes ezért alaposan átnézni a lehetőségeket, és a számunkra elfogadható maximális szintre növeli a biztonságot. Pár példa a szerver beállítási lehetőségeire:

**AllowUsers, AllowGroups, DenyUsers direktíva:** mivel az sshd a PAM-modullal kommunikálva fogja eldönteni, hogy ki az aki beléphet és ki az aki nem, első közelítésben itt felsorolhatjuk explicit módon, hogy mi kit engedünk. Ez felül fogja bírálni az egyéb felhasználók próbálkozásait. Például megadhatjuk így is

```
AllowUsers felhasznalo1 felhasznalo2
  felhasznalo3
```

**PasswordAuthentication, RSAAuthentication, PubkeyAuthentication:** összetartozó direktívák, amelyek segítségével letilthatjuk, hogy sima jelszó megadásával beléphessen valaki az SSH alrendszer segítségével. A kulcs alapú azonosítás lényege, hogy az **ssh-keygen**[10] program segítségével létrehozunk egy publikus és egy titkos kulcsból álló kulcspárt. A publikus részt feljuttatjuk a szerverünk meghatározott könyvtárába, majd a megfelelő opciók kiválasztásával már csak a titkos

[10] http://en.wikipedia.org/wiki/Ssh-keygen

kulcs birtokosa fog tudni belépni a rendszerbe, a sima /etc/shadow-ban tárolt jelszóval már nem. A privát kulcsot jelszóval védett formában pedig praktikusan egy erre elkülönített könyvtárban az identity- előtag használatával felcímkézve tároljuk, lásd a példát később. A publikus kulcsú azonosítás nagyban növeli a valós biztonságot és segítségével megnehezíthető például egy hátunk mögül meglesett jelszó, vagy egy keylogger program segítségével rögzített jelszóval való illetéktelen behatolás. A konfig állományba tehát helyezzük el a

```
PasswordAuthentication NO
RSAAuthentication yes
PubkeyAuthentication yes
```

sorokat.

**PermitRootLogin:** meghatározza, hogy a root felhasználó távoli elérés segítségével beléphet-e a rendszerbe. Érdemes tiltani, és a root felhasználót semmilyen Linux/Unix rendszeren nem használni. Helyette a sudo[11] (vagy su) megoldásait preferálni. Használata:

```
PermitRootLogin NO
```

**Port:** itt adhatjuk meg, hogy melyik TCP-porton hallgasson az SSH, ide fogunk tudni csatlakozni. Az

[11] http://hu.wikipedia.org/wiki/Sudo

alap beállítás szerint ez a 22-es lesz. Külső (internet) irányból publikusan elérhető szervereknél ezt érdemes tűzfallal védeni, esetleg port kopogtatásos[12] technológiával ezt nem nyilvánossá tenni mindenki számára. Az alapértelmezettől eltérő port használata néha magának a rendszergazdának nehezíti meg az életét (sok hálózaton belülről a 22-s portra szabad csatlakozni, ellenben máshova nem.)

Például:

```
Port 2022
```

**ListenAddress:** Több hálózati csatlakozás esetén, megadhatjuk az SSH-nak, hogy melyik „lábon” fogadjon el (tipikusan a menedzsment hálóból, illetve azon a lábon) kapcsolatot.

Pár példa a kliens konfigban javasolható beállításokra:

**~ssh/.config:** az ssh kliens esetében számtalan olyan megoldást alkalmazhatunk amely egyszerűsíti az életünket, és jó néhány olyat, amelyet más szoftver esetében csak plusz szolgáltatások beüzemelésével érhetnénk el. A leggyakoribb kliens oldali beállítás, amikor

[12] http://en.wikipedia.org/wiki/Port_knocking

az általunk adminisztrált hostokat felvesszük a .config fájlba, és utána alias formájában hivatkozunk rá, ez nem csak azért praktikus, mivel a kulcsot és port számot is meg tudjuk adni, hanem többek között a felhasználót is beállíthatjuk: például:

```
Host szerverem1
IdentityFile /home/user/kulcsok/identity-
  szerverem1
Port 2222
Protocol 2
User felhasznalo1
HostName 192.168.1.1
PasswordAuthentication no
```

Hasonlóan hasznos paraméter, ha tűzfalon kell keresztül SSH-zni, ahol a kapcsolat tétlenségi időkorlátja limitálva van, azaz például 5 perc inaktivitás elteltével bontja a kapcsolatot, akkor érdemes ezt a 3 kapcsolót is használni :

```
ServerAliveInterval 60
ServerAliveCountMax 10
ForwardAgent yes (ez az agentet állítja be,
  amelyről még lesz szó bővebben)
```

**Transzparens multi-hop:** Ha rendelkezem egy SSHGW géppel, amire bejutva tovább tudok menni a

WEB szerver gép felé, és onnan tovább a kvm1-es host gépre (a kvm1 egy virtuális gép, amelyre csak a host közbeiktatásával tudunk jelen esetben belépni), akkor alapesetben a Forward Agent és ssh-key opciókkal operálva mindenhova be tudok ssh-zni, majd onnan tovább a következőre. Ha azt szeretnénk elérni, hogy 1 db ssh parancs segítségével azonnal a 3. azaz a végcél kvm1-es gépre juthassunk el (SSHGW -> WEBSZERVER -> KVM1), akkor a következőt kell kiadni:

```
ssh -A -t SSHGW ssh -A -t WEBSZERVER ssh -A
  kvm1
```

Az -A opció mondja meg az SSH-nak, hogy a ForwardAgent kapcsoló mindenhol aktív (ON) legyen, attól függetlenül, hogy a klienseken engedve van-e vagy sem. A -t opció pedig erőlteti a Pseudo-TTY allokációt (ez az interaktív parancsfuttatásokhoz kell). A fenti SSH képességet a ~ssh/.config fájlban a következőképpen rögzíthetjük és hivatkozhatunk rá:

```
Host SSHGW
     IdentityFile
  /home/felhasznalo/kulcsok/identity-MAIN
     Port 22
     Protocol 2
```

```
    User update
    HostName 192.168.1.1
    PasswordAuthentication no
Host WEBSZERVER
    IdentityFile
/home/felhasznalo/kulcsok/identity-Test
 Port  22
    Protocol 2
    User update
 HostName 192.168.1.2
    PasswordAuthentication no
Host kvm1
  ProxyCommand ssh -q MAIN nc -q0 kvm1 22
```

Ha ezt így rögzítettük a konfigban, akkor utána egy egyszerű ssh kvm1 parancs kiadása után (látványosan tovább tart a bejutás, akár 5-10 másodperccel is) egyenesen a kvm1 gépre jutunk. Ugyanígy akár fájlokat is küldhetünk közvetlenül a kvm1 gépre/gépről. Ebben a példában a távoli gépről másolunk a saját gépünkre:

```
scp kvm1:/tmp/proba.txt /tmp/proba.txt
```

Vagy szükség esetén indíthatunk egy távoli (szintén beágyazott, azaz tűzfal és szerver mögötti alhálón) desktop gépen egy Nautilus ablakot, és hozzáférhetünk az állományokhoz a következőképpen (maradva a fel-

sorolt konfig sorainál képzeljük azt, hogy a kvm1 gép egy Ubuntu desktop, tehát fut rajta X szerver):

```
ssh -X kvm1 nautilus
```

kisvártatva egy a kvm1 gépen indított és az ottani állományokat megjelenítő fájlkezelő felületet kapunk. A fenti konfigban természetesen a kulcsokat és a felhasználókat tetszőlegesen variálhatjuk. Azaz, ha az első SSHGW gépen admin felhasználó amivel bejutunk, a 2.-on admin2, a kvm1 hoston pedig staff, akkor csak át kell írni a konfigot és persze az scp parancs is működni fog utána.

**Porttovábbítás**: elő szokott fordulni, hogy szükség van egy titkosított adatcsatornára két gép között. Mind az OpenSSH, mint a Putty háromféle porttovábbítási funkciót támogat. Talán leggyakrabban a -L (más néven lokális) formáját használjuk:

```
elso.gep.hu> ssh -f -N -L
  localhost:1111:harmadik.gep.hu:80
  masodik.gep.hu
```

Ezzel a paranccsal a saját gépünk (elso.gep.hu) 1111-es portján keresztül elérjük a harmadik.gep.hu 80-as portját, de a harmadik gép úgy érzékeli, hogy a má-

sodikról jöttünk (tipikusan ilyen, amikor egy olyan eszköz felügyeleti oldalát kell elérni távolról, amelyik csak a (neki) lokális hálózatról fogad el kapcsolódási kéréseket. Miután a fenti parancsot elindítottuk, a böngészőt a http://localhost:1111 címre irányítva máris a harmadik.gep.hu weboldalán találjuk magunkat. Ritkábban használt a -D az un. dinamikus applikációs porttovábbítás, ebben az esetben az ssh SOCKS-proxyként viselkedik. Használatához SOCKS-proxy támogatással rendelkező alkalmazás, vagy a későbbiekben tárgyalt **tsocks** szükséges.

```
elso.gep.hu> ssh -f -N -D localhost:1080
  masodik.gep.hu
```

A böngészőnkben beállítva a proxy támogatást, és proxy szerverként megadva a localhost:1080-as adatot, a forgalom az SSH-alagúton keresztül, a masodik.gep.hu -n keresztül jut el a címre. A harmadik, un. távoli (remote) porttovábbítás használata (-R opció) meglehetősen ritkaságszámba megy.

```
elso.gep.hu> ssh -f -N -R
  localhost:1111:harmadik.gep.hu:80
  masodik.gep.hu
```

eredménye: ha valaki a második gép 1111-es portjára csatlakozik, az az SSH kapcsolaton keresztül átkerül a kiinduló (első) gépre (ahol az ssh parancsot kiadták), majd onnan csatlakozik a harmadik gép 80-as portjára. Ez a példa az első példa (a lokális porttovábbítás) megfordítása, azaz amikor mi szeretnénk elérni azt, hogy valaki kívülről elérhesse a mi belső hálózatunkról elérhető eszközt. (Mi belülről megnyithatjuk az ssh-kapcsolatot, de a távoli ember kívülről nem.)

**Korlátozás az SSH-kulcs segítségével:** ha az a cél, hogy explicit megmondjuk az ssh-kulcs segítségével, hogy az adott felhasználó azonosítás után milyen parancsokat futtathat, akkor az ssh erre is kínál egy beépített megoldást (ez azonban nem keverendő a chroot direktívával). A kulcs generálása után a publikus oldali kulcsnál a ~/.ssh/authorized_keys fájlban a következő megoldást kell ebben az esetben alkalmazni:

```
from="192.168.1.1",command="/home/update/val
  idate-parancs" ssh-dsa
  AAANas2s4s5za82C1yc2EAAs97mAIPABIwAABAEA8x
  RLEVyrscvIoJmcWd9/qH......
```

Ahogy látszik, a kulcs mellett egy IP-címet határozunk meg, majd egy parancsot adhatunk meg. Ez akár

egy shell parancsállomány is lehet, amiben aztán az SSH_ORIGINAL_COMMAND nevű környezeti változót felhasználva akár többféle parancs engedélyezésére is lehetőségünk van. Itt[13] egy igen szép példa található. Ennek a gyakorlati jelentősége például az olyan jellegű mentéseknél van, amikor egy távoli root hozzáférést kell engedélyezni, amely aztán egy pillanatképet fog készíteni egy távoli mentő szerverre. Persze célszerű ezt a megoldást kerülni, és fordított logikával megoldani, azaz a helyi root felhasználó futtatja a mentést, és küldi át egy ssh-alagút, vagy VPN segítségével a távoli szerverre a pillanatképet, nem pedig fordítva. A példához visszatérve, jelen esetben az /etc/ssh/sshd_config fájlz érdemes a

```
PermitRootLogin=forced-commands-only
```

opcióval kiegészíteni a NO helyett, így a root kulcsa mellett engedélyezett fenti parancsok futtatására lesz csak lehetőség.

Egy további lehetőség az azonosításra a **google authenticator** és az SSH összepárosítása. Ha rendelkezünk Androidos vagy IOS-es mobil telefonnal és feltelepítjük

[13] http://troy.jdmz.net/rsync/#validate-rsync

a google hitelesítő alkalmazását[14], akkor lehetőségünk nyílik arra, hogy Ubuntu LTS alatt az SSH hoz kapcsoljuk. Fontos tudni, hogy az publikus kulcs alapú azonosítás és a ChallengeResponseAuthentication[15] egyszerre csak a 6.2 es verziójú OpenSSH-tól felfele támogatott. Azonban az Ubuntu LTS nem tartalmaz ilyen verzió számú OpenSSH-t, így a ennek a hitelesítésnek a használata csak akkor javasolt, ha a publikus kulcs azonosítását kizártuk, vagy valamilyen okból kifolyólag ideiglenesen szükséges, hogy a kulcs nélkül, de mégis egy sima jelszavas azonosításnál biztonságosabb módszerrel férjünk hozzá a gépünkhöz. Ilyen indokolt eset lehet, ha pl nyaralás közben egy internet kávézóból Live Ubuntu lemez segítségével, de kulcs nélkül akarunk hozzáférni az SSH hoz és a telefon nálunk van. A google authenticator telepítésének egyszerű lépései:

```
sudo apt-get install libpam-google-
  authenticator
```

[14] https://support.google.com/accounts/answer/1066447?hl=hu

[15] http://en.wikipedia.org/wiki/Challenge%E2%80%93response_authentication

a tárolóból feltelepítjük az authenticator alkalmazást, majd annak a felhasználónak a nevében elindítjuk, amelyiknek a nevében szeretnénk az azonosítást végezni:

```
$ google-authenticator
```

A terminálban megjelenő QR-kódot a telefonunk google authenticator alkalmazásával felolvastatjuk, majd a terminálban közben futó shell script néhány egyértelmű konfigurációs kérdésére válaszolunk. Ezekután az /etc/pam.d/sshd állományhoz hozzáadjuk a következő sort:

```
auth required pam_google_authenticator.so
```

Majd az /etc/ssh/sshd_config állományba pedig a következőt:

```
ChallengeResponseAuthentication yes
```

Figyeljünk arra, hogy a PAM auth be legyen kapcsolva az SSHD beállításokban (alapesetben be van). Ezek után a service ssh restart paranccsal újraindítjuk az SSHD-t. Innentől ha mindent jól csináltunk, akkor a slogin -l user@localhost parancs után kérni fogja a jelszavunkat, majd ha azt jól adtuk meg, akkor Verification code-ot is kérni fog, amelyet a telefonunk generál számunkra. Ha 6.2-es vagy e feletti OpenSSH val ren-

delkezünk, akkor bekapcsolhatjuk a kulcsos és a google authenticatort egy időben az AuthenticationMethods publickey,keyboard-interactive sshd opció segítségével, de vigyázat, ezt csak a 6.2-es verzió feletti SSH fogja értelmezni!

**SFTP alrendszer:** az SSH részét képezi az FTP leváltására kiválóan alkalmas és erősen ajánlott SFTP alrendszer. Nagyon sok rendszer-adminisztrátor a mai napig kénytelen az elavult és ezer sebből vérző FTP protokoll fölé ültetni egy TLS/SSL réteget és azt használni, mivel sok esetben a követelmények kimondják az FTP protokoll használatát (pl. elavult weblaptervező szoftver stb.). Pedig az SFTP remek megoldást kínál az SSH összes előnyével együtt. Támogatja a chroot-olt környezet létrehozását, felhasználókra és csoportokra is, vagy akár sftp-only felhasználókat is létrehozhatunk. Ezeket kell hozzá a szerver beállításaiba (/etc/ssh/sshd_config) írni:

```
Subsystem sftp internal-sftp
Match Group sftp-only
AllowTCPForwarding no
X11Forwarding no
ForceCommand internal-sftp
```

```
ChrootDirectory %h
```

Főként azért nagyon jó választás az FTP-vel szemben, mivel az SSH titkosítását és opcióit kínálja az FTP-vel szemben, így akár az AllowUsers, DenyUsers stb direktívákat és természetesen az RSA és DSA kulcsokat is használhatjuk. Szerencsére ma már szinte az összes elterjedt operációs rendszerre rendelkezésre áll valamilyen grafikus felület[16], amely segítségével a kezdők is tökéletesen elboldogulnak. Továbbá az olyan mentésekhez is fel lehet használni, ahol parancs állomány írása egészíti a mentést. Tipikusan ilyen terület, amikor az SQL-szerver tömörített dump állományait kell időről időre átvinni valahova (kiegészítő heti, havi mentés stb.):

```
sftp -b /root/sftp.sh -i /root/keyfile
  192.168.1.1:/backup/
```

Az SSH számtalan izgalmas lehetőséget tartalmaz még számunkra. Jellemzően egy átlag SSH felhasználó legfeljebb az 1-2%-át használja a lehetőségeknek. Sajnálatos módon azonban ez a fejezet nem az SSH szinte

[16] Windows környezetben javasolható a WinSCP (http://winscp.net/) nevű alkalmazás, de parancssoros SFTP kliens a Putty-hoz is tartozik.

végtelen lehetőségeiről szól, csupán a leggyakoribb lehetőségeket taglalja.

## Üzemeltetést segítő alkalmazások

Az SSH kliens-szerver megoldás segítségével most már hozzáférhetünk helytől független módon a távoli szerverünkhöz. A kérdés az, mire is lesz ez jó nekünk? A legtöbb disztribúciónak alapesetben része, vagy csomagkezelővel elérhető számtalan olyan eszköz, amely az alapvető adminisztrációt kezeli. Olyan sok adminisztrációs eszköz létezik, hogy azok felsorolása túlmutat ezen fejezet céljain. Néhány fontosabb eszközt azonban részletezünk:

### Nmap[17]

Ezen eszköz segítségével a szerverünk hálózati (lokális és távoli) felderítését tudjuk elvégezni. Segítségével a nyitott/zárt/rejtett portokat tudjuk letapogatni,

[17] http://nmap.org/

vagy közelítő információt kaphatunk az operációs rendszer verziójáról. Számos egyéb funkciója van[18].

## Sudo[19]

Unix/Linux környezetben lehetővé teszi azt, hogy egy másik felhasználó – általában a root – jogosultságával (nevében) futtassunk programokat. Igen részletesen beállítható, hogy akár csak egyetlen parancs milyen módon és paraméterekkel engedélyezett. A mai modern szemlélet szerint a root felhasználó csupán technikai jelleggel szerepel a rendszerben, azt élesben (belépni a nevében, vagy su parancson keresztül) használni nem tanácsos, éppen ezért érdemes a sudo használatára áttérni. A Sudo konfigurációs állománya az /etc/sudoers fájlban található, jellemzően a mai disztribúciók a visudo parancs segítségével kínálják szerkesztésre a sudoers állományt. Manapság már nem szokás közvetlenül a felhasználókat felvenni a sudoers fájlba, hanem a tipikusan sudo-nak nevezett csoportba érdemes berakni az adott felhasználót ahhoz, hogy rendelkezzen az ALL jo-

[18] http://en.wikipedia.org/wiki/Nmap
[19] http://hu.wikipedia.org/wiki/Sudo

gosultsági szinttel. A sudo számtalan lehetőséget kínál[20], az egyik leggyakrabban használt opció, amikor egy adott felhasználónak nem engedünk meg mindent, csupán egy adott parancs, vagy parancsfájl futtatását engedélyezzük, az alábbi módon:

```
update ALL=NOPASSWD: /usr/bin/apt-get,
  /usr/bin/aptitude
```

A fenti példában a később tárgyalandó apt-dater program használatához létrehoztunk egy update nevű felhasználót, és ezen felhasználó részére jelszó nélkül engedélyeztük, hogy az apt-get és aptitude parancsokat root jogosultsággal futtassa. Régebben, jellemzően a betárcsázós korszakban, a pppd futhatott hasonló jogosultságokkal, hogy az összes felhasználó vezérelhesse.

## GPG[21]

Biztonsági szempontból az egyik legnagyszerűbb szabad szoftver, természetesen GPL licenc alatt elérhető. Legfontosabb funkciója a titkosítás. Segítségével ál-

[20] Egy üres délutánon vegyük magunk elé a dokumentációt, jól el fogunk csodálkozni a lehetőségein

[21] http://www.gnupg.org/

lományokat, leveleket lehet titkosítani oly módon, hogy az harmadik fél számára nem fejthető vissza. Nyílt kulcsos titkosítást használ, amely eljárásnál két kulcsot használunk. Az egyik egy publikus (nyilvános) kulcs, amit titkosítás nélkül kell közzétennünk a másik fél számára – ezt bárki megszerezheti. A másik kulcs a privát (titkos) kulcs, amelyet jelszóval védve biztonságban kell tárolnunk. A rendszer alapelve, hogy a publikus kulcsból nem határozható meg a privát kulcs (és fordítva sem). A másik nagy előnye ennek a titkosítási módnak, hogy már akkor tudok egy általam ismeretlen embernek titkosítottan levelet küldeni, ha még nem találkoztunk és akár nem is ismer. Egyetlen feltétele, hogy az interneten található kulcsszerverek egyikén, vagy a másik ember publikusan elérhető weboldalán szerepeljen a publikus kulcsa, amelyet felfűzve a saját kulcskarikámra máris tudok neki titkos levelet írni, vagy állományt kódolni neki úgy, hogy azt csak és kizárólag ő fogja tudni kibontani/olvasni. Ideális tehát rendszergazdák közötti kommunikációra, hiszen sokszor érzékeny információt kell utaztatni nyílt hálózaton (jelszavak, hozzáférések stb.). Továbbá ideális lokális jelszótárolásra is. Alkalmas továbbá bizalmi körök kialakítására a

saját aláírási rendszerével. Fontos funkciója, hogy a kulcsaink rendelkeznek úgynevezett egyedi ujjlenyomattal (*fingerprint*). Azaz ha kulcsot akarok cserélni egy általam még ismeretlen emberrel, akkor első teendő a kulcs felfűzése a karikára, majd az ujjelenyomat egyeztetése a másik féllel, immár egy azonosításra alkalmas csatornán, pl. telefon, videobeszélgetés, aláíró parti, stb. A Debian alapú rendszerek[22] csomagjai a biztonsági csapat GPG-aláírásával vannak ellátva, így azok telepítésnél ellenőrzésre kerülnek. Akár az MD5 szignó helyett is alkalmazható remek és annál megbízhatóbb eszköz. A GPG bővebb ismertetését és végfelhasználói konfigurációját, valamint a hozzá kapcsolódó segédprogramokat a keretrendszer GPG melléklete tartalmazza.

## tsocks[23]

Gyakori probléma, hogy olyan nyílt hálózaton keresztül kell távoli adminisztrációt végezni, amely nem

[22] a legtöbb terjesztés erős késéssel ugyan, de csatlakozott ehhez a hitelesítési megoldáshoz

[23] http://tsocks.sourceforge.net/

titkosított (nincs VPN, nincs SSL, stb), mint például egy nyílt Wi-Fi. Ilyen esetben a legegyszerűbb megoldás, ha egy SSH-alagutat[24] húzunk ki egy megbízható szerverünk és a kliens között. Az alagútba aztán beterelhetjük olyan alkalmazások forgalmát, amelyek ezt támogatják. Például a Firefox, a Chrome (és a Chromium), de még az Opera böngésző is támogatja a SOCKS5 (hivatalos nevén: socket-proxy version 5) üzemmódot, amikor a forgalom ebben az alagútban tud közlekedni, így rejtve el titkosított módon az amúgy titkosítás mentesen menő – például web – forgalmat. Sajnos azonban nem minden program tud alagút technológiával dolgozni, éppen ezért érdemes a tsocks programot használni, amely a legtöbb TCP-alapú program kommunikációját bele tudja terelni egy előtte SSH-val kiépített csatornába.[25] Ha a tsocks csomag már telepítve van, akkor használata a következő. Az SSH fejezetben

[24] http://www.linuxjournal.com/content/ssh-tunneling-poor-techies-vpn

[25] A tsocks-hoz hasonló elven működő alkalmazás nagyon sok létezik, van aki a dante-client csomagban levő socksify, mások a proxychains nevű eszközre esküsznek, de a sort lehetne még folytatni

már tárgyalt porttovábbítás használatával kiépítünk egy lokális dinamikus porttovábbítást, például így:

```
ssh -i /home/user/kulcsok/identity-szerver
   -p 22 -v -N -D localhost:1080
   user@szerveremneve
```

Fenti[26] eredményeként a localhost 1080-as portján elérhetővé válik egy SOCKS5 proxy, amely a hozzá érkező forgalmat az SSH-alagúton keresztül átküldi „szerveremneve” gépre. Onnan pedig már mintha arról a gépről indult volna, megy az adat a célhoz. Ezt követi a tsocks. Először állítsuk be a nekünk szükséges opciókat az /etc/tsocks.conf fájlban. Amelyek az előző ssh-parancshoz igazodva a következők legyenek:

```
server = 127.0.0.1
server_port = 1080
```

Ez a 2 egyszerű sor azt mondja meg, hogy a localhost-on a 1080-as porton hallgat a SOCKS5 proxy, amin keresztül kell majd a tsocks-nak a forgalmat továbbítania. Vigyázat, nincs több azonosítás ez után.

[26] Ha a fenti parancsban a -v opció helyett a -f -et használjuk, akkor az ssh ugyan jóval kevesebb infót küld, cserébe háttérbe megy, így nem szükséges a terminált nyitvatartani a kapcsolat életbentartásához - a tesztek utáni éles használathoz talán alkalmasabb így.

Azaz aki elérheti a localhost 1080 portját, az az alagútban fog forgalmazni! Mindezek után már csak a tsocks előtaggal kell indítani az alkalmazást ahhoz, hogy az adott alkalmazás az ssh alagútban forgalmazzon, például így:

```
tsocks pidgin
```

Természetesen böngészőt is indíthatunk ugyanilyen módon, bár a böngészők – mint az fentebb egyszer már szerepelt – általában beállíthatóak SOCKS5 proxy használatára. A legtöbb esetben ez jól működik. Akadnak azonban olyan esetek, például pont böngészők esetében, amikor a forgalmazás például a böngészőből hívott Java alkalmazás miatt csak részben megy az alagútban. Ilyenkor érdemes magának a Java alkalmazásnak is a SOCKET-proxy használatát beállítani.

## sshuttle[27]

A sshuttle is egy remek eszköz VPN építéshez, ha a megvalósítandó feladat egyezik a tsocks-szal megoldható feladatkörökkel. A fő különbség, hogy amíg a tsocks esetében az alkalmazásoknak vagy támogatni kell a socket proxy üzemmódot, vagy pedig a tsocks parancs

[27] https://github.com/apenwarr/sshuttle

segítségével a tunnelbe kell terelni a forgalmat, addig az sshuttle esetében erre nincs szükség, mivel az IP-TABLES-t állítja be a lokális gépen, a forgalmat a távoli szerver fele terelve. A távoli szerveren viszont szükséges a Python interpreter és az ssh-n keresztüli parancsfuttatási lehetőség megléte (lévén az sshuttle szerver komponensét feltöltés után ott futtatja az eszköz). UDP és ICMP adatot nem fogunk tudni küldeni/fogadni, de a legtöbb esetben erre nem is feltétlen van szükség, legalábbis nem feltétlen a tunnelen keresztül. (Kivételként a névfeloldáshoz szükséges DNS-forgalmat lehetne említeni, de ehhez van egy --dns opció, ha szükséges.) Telepítése és használata is egyszerű:

```
apt-get install sshuttle
```

A telepítés után nincs szükség konfigurációk állítására, egész egyszerűen egy létező távoli SSH hozzáférésre van csak szükségünk:

```
sshuttle -r
  felhasznalonev@tavoli.szerver.kft 0/0
```

Ebben az esetben az összes TCP forgalmat a távoli szerver felé tereltük, de megadhatunk neki különböző alhálózatokat is. Természetesen a korábban már leírt

transzparens MultiHop konfigurációk is használhatóak, illetve az ssh .config állományába rögzíthetjük a neveket, portokat, felhasználókat előre. A csatlakozás után az összes forgalom a megadott SSH szerver felé fog menni, így például a böngészők (Firefox, Opera, Chrome), csevegőprogramok (Pidgin, Xchat, Skype), levelező programok (Thunderbird) automatikusan az SSH tunnelben fognak végződni. Lehetőség van a -D opció segítségével arra, hogy démon üzemmódban a háttérben fusson, így nem kell azt a konzolt/terminált nyitva tartani, amelyben a parancsot kiadtuk (bár ez utóbbi funkcionalitás akár a screen, akár az fg parancs segítségével is kiváltható).

## Hálózati monitorozó szoftverek

**I**ptraf[28], arpwatch[29], wireshark[30], tcpdump [31], lsof[32] és még számtalan hasonló program létezik lokális és tá-

[28] http://iptraf.seul.org/
[29] http://en.wikipedia.org/wiki/Arpwatch
[30] http://www.wireshark.org/about.html
[31] http://www.tcpdump.org/
[32] http://hu.wikipedia.org/wiki/Lsof

voli hálózatok forgalmának felügyeletére, hiba keresésére.

## Jelszókezelés[33]

Minden rendszer használatának igen sarkalatos pontja a megfelelő jelszó megválasztása. A jó jelszó számok, betűk, írásjelek megfelelő kombinációjából áll, legalább 8-12 karakter hosszúságban. Jelszavak tömeges létrehozása esetén fontos, hogy ne a fantáziánkra bízzuk a jó jelszó kitalálását, hanem egy programra, mint amilyen például a pwgen[34].

## Editor

Egy igen sarkalatos és megosztó pontja ez a Unix/Linux közösségeknek. A karakteres felületen végzett munka egyik sajátossága, hogy a konfigurációs állományokat tudni kell szerkeszteni, terminál- és billentyűzetkiosztás-független módon. A disztribúciók általá-

[33] Részletesebb tárgyalása a Központi autentikációról szóló fejezetben

[34] http://pwgen-win.sourceforge.net/

ban alap telepítésben tartalmazzák a következők valamelyikét: vi[35], vim[36], joe[37], nano[38], pico[39], emacs[40]

## Shell program

Amint az SSH program segítségével azonosítottuk magunkat, a bejelentkezés következő lépése, hogy a /etc/passwd állományban meghatározott programot indítja a rendszer számunkra. Ugyanúgy, mint az editor kiválasztásában itt is az számít, hogy a saját komfort zónánkat megtaláljuk és úgy válasszuk ki a számtalan lehetőség közül a megfelelőt. A legnépszerűbb lehetőségek[41]: bash, tcsh, zsh, ksh, jsh stb.

[35] http://hu.wikipedia.org/wiki/Vi
[36] http://www.vim.org/
[37] http://joe-editor.sourceforge.net/
[38] http://www.nano-editor.org/
[39] http://en.wikipedia.org/wiki/Pico_(text_editor)
[40] http://www.gnu.org/software/emacs/
[41] http://penguin.dcs.bbk.ac.uk/academic/unix/linux/shells/index.php

## Apt-Dater[42]

Egy terminál alapú, szervereken csomagok karbantartására és frissítésre használható eszköz. Egy igazi hiánypótló alkalmazás, amely alapjait a nagyvállalati környezetből merítették. Az ötlet lényege, hogy nagyvállalati környezetben több tíz vagy száz (vagy akár ezer) ugyanolyan szervert kell karbantartani. Azaz ha érkezik egy javítás, amelyet azonnal telepíteni kell, akkor az adminnak ne több száz gépre kelljen ssh-val bemennie és ott kézzel frissítenie, vagy a cron ütemezőből megoldani a dolgot (hiszen sokszor ezek a javítások interaktív módon beavatkozást is igényelnek), hanem egy üzemeltetői shellből biztonságos módon tudja figyelemmel kísérni és természetesen be is tudjon avatkozni. Az apt-dater erre kiválóan alkalmas, kezeli a kulcsos ssh azonosítást, használja a screen-t, és ha szükséges az ssh-agent-et. Ha több mint 2-3 Debian vagy Ubuntu alapú szervert kell üzemeltetni, akkor gyakorlatilag nélkülözhetetlen alkalmazás. Igazi előnye, hogy a beállítása szerverenként csak pár percet vesz igénybe. Az apt-

[42] http://www.ibh.de/apt-dater/

dater szóhasználata szerint kliens az a gép, ahonnan a többit menedzselem, és host-nak hívja a menedzselt szervereket. A beállításához először fel kell telepítenünk arra a kliens gépre az **apt-dater**-t amelyikről az összes szervert elérjük. Ez lehet a saját PC/notebook gépünk, vagy akár egy erre fenntartott, biztonságosan elszeparált (például: csak VPN-nel elérhető, titkosított merevlemezű VM) virtuális gép is:

```
apt-get install apt-dater
```

Az apt-dater telepíteni fogja a függőségeit is, ezek után készítsünk egy megfelelően nagy ssh kulcsot a külön felhasználóhoz, amely futtatni fogja a dater-host scripteket. A -C opció egy megjegyzést fűz a kulcshoz, a -f segítségével pedig megadhatjuk, hogy az alapértelmezett id_rsa (és id_rsa.pub) helyett hol tárolja a kulcsokat

```
ssh-keygen -t rsa -b 8192 -C apt-dater-kulcs
  -f ~/kulcsok/identity-update-server[43]
```

[43] A példában RSA kulcsot generálunk. A RSA mellett szól a szinte tetszőleges kulcshosszúság megadási lehetősége, ezért is szerepel a példában 8K. A DSA előnye hogy véletlenül sem tudnánk SSH protokol v1-et használni vele (viszont sajnos nem állítható a kulcshossz.)

Az elmentett titkos kulcsot tároljuk biztos helyen, a publikus párját pedig juttassuk fel a karbantartani kívánt szerverekre az ssh-copy-id parancs segítségével, miután a felhasználót már létrehoztuk. A host gépeken létrehozunk tehát egy update felhasználót, amelynek beállítunk egy kellően véletlen (pwgen -s 16) jelszót (ez a továbbiakban nem fog kelleni).

```
adduser update
```

Az ~/update/.ssh/authorized_keys állományba másoljuk a készített publikus kulcsot. Ha ezzel megvagyunk, akkor az /etc/ssh/sshd_config-ban engedélyezzük az update felhasználó belépését az AllowUsers sor bővítésével:

```
AllowUsers admin update
```

Majd leteszteljük, hogy távolról az update felhasználó bejelentkezése működik-e a kulcs segítségével:

```
ssh-add ~/kulcsok/identity-update-server
  #(ez az állomány tartalmazza a titkos
  kulcsot)
ssh update@szerverunk.cime
```

Ha minden rendben volt akkor a host gépre felrakjuk az apt-date host script gyűjteményt:

```
apt-get install apt-dater-host
```

Majd beállítjuk a hoston szintén a sudo paramétereit, hogy az update felhasználónak legyen joga frissíteni:

```
visudo
update ALL=NOPASSWD: /usr/bin/apt-get,
  /usr/bin/aptitude
```

Ezzel lényegében a host gépeken végeztünk a beállításokkal. A továbbiakban a saját gépünkön, vagyis azon a gépen fogunk dolgozni, ahonnan a frissítéseket vezérelni akarjuk. Az első lépésben már felraktuk az apt-dater-t, amely így alap beállításokkal került telepítésre. A konfig állományok a ~/.config/apt-dater/ könyvtárban találhatóak. Amivel első körben érdemes dolgozni, az a hosts.conf állomány. Szintaxisa viszonylag egyszerű és érdemes az ssh .config állományával variálni:

```
[Main]
  Hosts=update@192.168.1.1:22
[Test]
  Hosts=update@192.168.1.2:22
[sql]
  Hosts=update@sql
[apache]
  Hosts=update@apache
```

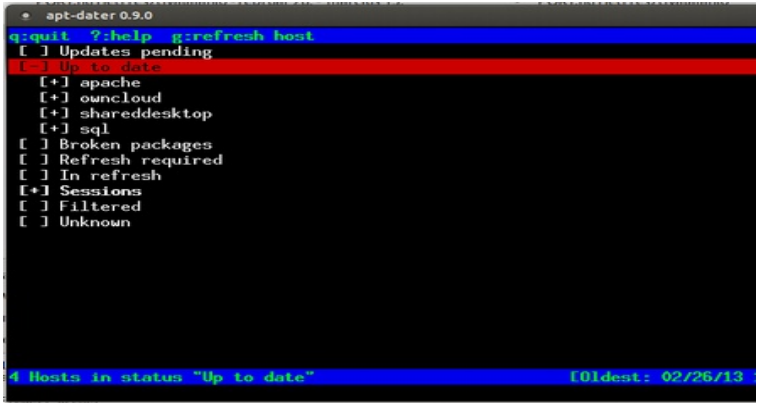
az első két gép, Main és Test egyszerűbb, közvetlenül ssh-val elérhető gépek. A másik kettő (sql, apache) pedig a Main vagy test gépeken keresztül, mivel azok az itteni példákban virtuális gépei azoknak. Jelen konfighoz az SSH Transzparens Multihop képességét fogjuk használni, amelyet az ssh fejezetben részletesen ismertettünk:

```
Host SSHGW
   IdentityFile /home/user/kulcsok/identity-
   MAIN
   Port 22
   Protocol 2
   User update
   HostName 192.168.1.1
   PasswordAuthentication no
Host WEBSZERVER
   IdentityFile /home/user/kulcsok/identity-
   Test
   Port  22
   Protocol 2
   User update
   HostName 192.168.1.2
   PasswordAuthentication no
Host apache
   ProxyCommand ssh -q MAIN nc -q0 apache 22
```

```
Host sql
ProxyCommand ssh -q MAIN nc -q0 sql 22
```

Az Apt-dater és az ~/.ssh/config paraméterei összhangban kell hogy legyenek a fenti leírás szerint. Jelen esetben, ha a kliensről akarunk a 2. szintű apache virtuális géphez eljutni, akkor az ssh-agent indítása után (amelyet a standard Ubuntu desktop alapesetben elindít) az ssh-add /home/user/kulcsok/identity-MAIN paranccsal a kulcsot hozzáadjuk az ssh-agent tárolójához, és ezek után már csak az ssh apache parancsot kell majd kiadni, és azonnal a 2. szintű gépre lépünk be.

Mindezek után az apt-dater parancs kiadása után már ez látszik előttünk:




Az apt-dater teljesen alapszintű használata viszonylag egyszerű, a súgót a ? lenyomásával tudjuk aktiválni.

## Screen[44]

Szintén igen hasznos program, amely segítségével egy belépett shell ablakot sokszorozhatunk meg. Gyakorlatilag egy karakteres ablakkezelő, amelynek segít-

[44] http://www.gnu.org/software/screen/

ségével akkor is folytathatjuk a program futását, ha a terminál kapcsolat megszakadt (azaz kiléptünk az ablakból). Ideális több gépen való munka koordinálására, akkor ha például egy ablakban akarunk mindent csinálni. Valamint ideális batch alapú program futtatására, megfigyelésre, adatgyűjtésre. Tipikus felhasználási terület, amikor egy nagy méretű adatbázist állítunk helyre (dump) távolról. Ha ilyenkor megszakad az internet-kapcsolat akár csak 1-2 másodpercre is, az SSH nem biztos, hogy megtartja nekünk a kapcsolatot. Így az adatbázis dump is megszakadhat, ha azt kézzel végeztük. Ha ugyanezt a folyamatot screen alatt futtatva hajtjuk végre, akkor újra belépve a screen -x parancs kiadásával visszakapjuk az éppen aktuális állapotot. Segítségével akár élőben is megfigyelhetünk egy kezdő rendszergazdát (a betanítás során), így elkerülhető, hogy egy esetleges hiányosság nagyobb problémát okozzon.[45]

[45] A screen mellett érdemes megemlíteni, hogy több hasonló funkciókra alkalmas másik program létezik, például tmux (elérhető a http://tmux.sourceforge.net/ oldalról) vagy byobu (http://byobu.co)

# Távoli asztal alkalmazások

Míg e dokumentum első (nagyobbik) részében a távoli rendszerek elérésének elsősorban karakteres felületű részével foglalkoztunk, ez az utolsó rész a grafikus felületű elérések különböző megvalósításairól szól. A klasszikus unixos rendszergazda szemléletbe ez nem fér bele: az alap hozzáállás az, hogy a rendszer adminisztrációjához csak a következő dolgokat kell tudni: hogy hívják a menedzselendő szoftver konfigurációs állományát/állományait, hol van(nak) ez(ek) a fájl(ok) és mi a szintaxisuk. Ezen kívül már csak egy egyszerű szövegszerkesztőre van szükség, és minden megoldható.

Sajnos(?) nem lehet megkerülni a tényt: a Linux disztribúciókban egyre több kényelmi funkció jelenik meg, egyre több szolgáltatáshoz készülnek grafikus adminisztrációs eszközök, amelyek néha meglehetősen komplex szoftverrendszerek viszonylag egyszerű kezelését oldják meg. Ráadásul egyre kevesebb rendszergazda kell felügyeljen egyre több (és többféle) rendszert. Ezért aztán ennek a résznek a végére egy rövid áttekintés keretében megemlítjük azokat a lehetősége-

ket is, amelyek arra szolgálnak, hogy egy távoli gép grafikus felületét elérjük.

Elsőként egy kakukktojás. A Unix/Linux-világ hagyományos grafikus felülete az X Window System névre hallgató, kliens-szerver alapon működő felület. Az X-szerver egy speciális program, amely a felhasználóval kommunikál, kezeli a beviteli (billentyűzet, egér) és megjelenítő (monitor) eszközöket. Tipikusan azon a gépen fut, amelyik elé a felhasználó leül dolgozni. Az X-kliens pedig az a program, amelyik (esetleg egy másik gépen futva) csinál valamit, aminek az eredményét egy ablakban meg lehet nézni, aminek a futását befolyásolhatjuk a billentyűzetről vagy az egérrel bevitt adatok segítségével. Mivel az X már eredetileg is kliens-szerver felépítésű, természetesen része, hogy a két fő komponens (az X-szerver és az X-kliens) nem ugyanott fut. Azaz a legelső megoldás, hogy ha a saját gépemen fut már az X-szerver, akkor egy távoli gépen futtathatok X-kliens alkalmazást, és megmondhatom neki, hogy a megjelenítést, és adatbeolvasást az én gépemen futó X-szerveren keresztül oldja meg. Ehhez először be kell jelentkezni a távoli gépre – akár parancssoros felületen keresztül. A futtatásnak két egyszerű módja: vagy van

az eszköznek erre a célra parancssori kapcsolója – ez tipikusan a -display (ritkábban -d) –, ebben az esetben:

```
xterm -d SZERVEREM:0.0
```

formában kell futtatni, vagy pedig a DISPLAY nevű környezeti változó beállításával jelezhető, hogy hol az X-szerver, amihez a kliens csatlakozni fog. Azaz:

```
DISPLAY=SZERVEREM:0.0
export DISPLAY
xterm
```

formában.

A korábban már említett ssh ehhez is nyújt segítséget, az ssh parancssorában megadható -X (vagy -Y) opció egy ún. X11-forwarding funkciót valósít meg, ez leegyszerűsítve azt jelenti, hogy a távoli gépre ssh-n keresztüli bejelentkezés során az ssh létrehoz egy alagutat, beállítja az előbb említett DISPLAY változót, így tehát tetszőleges grafikus alkalmazás automatikusan az ssh által biztosított titkosított alagúton keresztül, a lokális gépünkre fogja továbbítani a grafikus alkalmazás adatait. Gyakorlatilag a fenti eredményre vezet a következő parancs:

```
ssh -Y user@host /usr/bin/xterm
```

A grafikus eszközök távoli elérésének másik módja, amikor nem csak egy-egy alkalmazást, hanem akár a teljes futási környezetet átvesszük. Ennek oka lehet az, hogy segítenünk kell egy felhasználónak, és könnyebb hálózaton keresztül elérni a gépét, mint odamenni. Vagy az, hogy szeretnénk valamit a csoda grafikus eszközzel elérni, de sajnálatos módon csak az ikonját ismerjük fel az asztalon/tálcán, de a nevét azt nem tudjuk.

Rengeteg népszerű megoldás létezik a távoli asztal megjelenítésére, függően a technológiától amely segítségével kapcsolatot lehet teremteni két kliens között. Lehetséges akár SSH-alagút, akár VPN megoldások közbeiktatásával, amennyiben 2 publikus IP között, vagy egy belső hálózaton a gépek közvetlen tudnak egymáshoz csatlakozni. Praktikus megoldást jelent sok esetben az IT osztály munkatársai számára, ha az érdekeltségi körükbe tartozó klienseket úgy tudják támogatni, hogy közben pontosan látják a másik gép képernyőjét. Ez azonban felvet természetesen adatbiztonsági és személyiségi jogi kérdéseket is, azonban ha ezek tisztázottak, akkor valóban nagy könnyebbség úgy segíteni egy program telepítésében (vagy akár feltelepíteni neki a programot), hogy a lokális gépünk segítségével „kö-

zel hozzuk" a távol gép képernyőjét, billentyűzetét és egerét. Az ilyen szoftverek hasznosak továbbá a keresztplatformok esetében is. Azaz egy linuxos VNC klienssel el tudunk érni egy Windows asztalt, ha azon fut egy VNC-szerver és természetesen visszafelé is működőképes a módszer[46].

**UltraVNC**[47]: egy GPL licenc alatt kiadott teljes funkcionalitású távoli asztal elérésére szolgáló kliens-szerver szoftver. Ideális Windowst futtató gépeken, mivel támogatja a titkosítást, azonban a kliens változat Linux alá csak böngészőből, Java használatával lehetséges.

**RealVNC**[48]: privát használatra ingyenes, de számos fizetős megoldás létezik belőle. A kliens és a szerver is elérhető a legtöbb Linux terjesztésben csomagként. Titkosítást az ingyenes verzió nem támogat.

**Rdesktop**[49]: a linuxos kliens szoftver GPL licenc alatt elérhető, és mivel natívan támogatja az RDP proto-

[46] Ehhez Linux alá sok felhasználóbarát eszköz létezik, mint pl. a Vinagre vagy a Remmina.

[47] http://www.uvnc.com/

[48] http://www.realvnc.com/

[49] http://www.rdesktop.org/

kollt, ezért jól használható a Windows kliensek elérésére Linux alól is.

**Spice:** a Spice protokollt a Red Hat tette nyílt forrásúvá 2009-ben. Számtalan előnye mutatkozik a VNC-vel szemben, tekintve, hogy sokkal hatékonyabb tömörítést és eleve titkosítást is használ. Mára már integrálható a KVM alá is, illetve használható sima asztal-asztal összekötésként is.

Szinte közös jellemzője az összesnek, hogy az ingyenesen használható felületen a titkosítás vagy nem érhető el, vagy nem elégséges, ezért ezek használata jellemzően belső hálózatokon a védett zónán belül javasolt, vagy VPN vagy SSH-alagút segítségével együtt.