Konta użytkowników w systemie Unix

Witold Paluszyński
Katedra Cybernetyki i Robotyki
Politechnika Wrocławska
http://www.kcir.pwr.edu.pl/~witold/

2000–2013

Konta użytkowników — lokalne

- Konta użytkowników zapewniają możliwość pracy w systemie, i jednocześnie izolację użytkowników od systemu i od siebie nawzajem.
- Konta głównie reprezentują „ludzkich" użytkowników, ale można tworzyć też konta anonimowe, oraz konta techniczne, z możliwością logowania lub bez.
- Podstawowe atrybuty `/etc/passwd`: nazwa (login), numer (uid), grupa (gid), hasło, katalog dyskowy, interpreter poleceń, nazwa własna.
- Inne „rozproszone" atrybuty: grupy dodatkowe, quota, uprawnienia do korzystania z crontaba i at, do drukowania na konkretnych drukarkach, i inne.

```
/etc/passwd:
root:x:0:1:Super-User:/:/sbin/sh
powerdown:x:0:1:Power Down User:/:/usr/local/sbin/powerdown
reboot:x:0:1:Reboot User:/:/usr/sbin/reboot
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
witold:x:1001:1000:Witold Paluszynski,p.307/C-3:/home/witold:/usr/bin/tcsh
```

Konta użytkowników — sieciowe

Środowisko systemów uniksowych sprzyja tworzeniu kont i katalogów sieciowych. Istnieje w zasadzie jedna koncepcja i **standard sieciowego systemu plików NFS**, stworzony przez firmę Sun Microsystems w latach 1980-tych, i pozwalający współdzielić katalogi użytkowników (i inne, patrz np. katalog /usr/share) między uprawnionymi komputerami.

Jednak teoretycznie prostszy system, jakim powinna być **sieciowa** lista / katalog / **baza danych użytkowników**, rodził się długo i w bólach. Sun Microsystems stworzył najpierw system NIS (pierwotnie zwany Yellow Pages), który był prosty, ale wadliwie pomyślany i tworzący zagrożenia bezpieczeństwa (kompletną bazę haseł można w nim ściągać z dowolnego miejsca w Internecie).

Jego poprawioną wersją miał być system NIS+. Jednak okazał się pomyłką — nie tylko nie rozwiązał problemu zabezpieczeń, ale wprowadził imponujący poziom złożoności. Mimo to, systemy NIS i NIS+ nadal są aktywnie używane.

Dopiero standard LDAP, wprowadzony w połowie lat 1990-tych, dostarczył mechanizm pozwalający bez ograniczeń „usieciowić" konta użytkowników systemów uniksowych.

Konta użytkowników — LDAP

LDAP (*Lightweight Directory Access Protocol*) oryginalnie był uproszczonym protokołem dostępu do baz danych telekomunikacyjnych usług katalogowych X.500. Może być jednak używany jako samodzielny, z własną bazą danych. LDAP stał się standardem serwisu nazw na systemach uniksowych, ale jest wieloplatformowy, i bywa używany w środowiskach heterogenicznych.

LDAP (podobnie jak systemy NIS i NIS+) pozwala przechowywać atrybuty konta użytkownika w sieciowej bazie danych, i udostępniać je uprawnionym systemom. Istotną cechą serwerów katalogowych, która różni je od zwykłych baz danych, jest szybkość odpowiedzi. Serwer katalogowy przechowuje niewielkie dane, o niezbyt rozbudowanej strukturze, ale musi odpowiadać szybko.

```
diablo-522> ldaplist -l passwd witold
dn: uid=witold,ou=people,ou=lab07,o=zpcir
        uid: witold
        cn: Witold Paluszynski,C3 p307,320-2741,
        objectClass: account
        objectClass: posixAccount
        objectClass: top
        objectClass: shadowAccount
        shadowMax: 7000
        shadowLastChange: 13000
        loginShell: /usr/bin/tcsh
        uidNumber: 1001
        gidNumber: 1000
        homeDirectory: /home/witold
        gecos: Witold Paluszynski,C3 p307,320-2741,
        userPassword: {crypt}s0t5VDcndmk1.
```

Sieciowe katalogi użytkowników — automounter

W sieciowym systemie, w którym informacje o kontach użytkowników mogą znajdować się w serwerach NIS lub LDAP, katalogi użytkowników mogą być **lokalne**, lub **montowane z sieciowego serwera plików**

W systemie, w którym konta różnych użytkowników zlokalizowane są w różnych miejscach (niektóre na lokalnym dysku, a inne na jednym lub więcej sieciowych serwerach plików), często używanym mechanizmem jest `automounter`. Jest on mechanizmem **montowania na żądanie systemów plików**, albo katalogów. Próba otwarcia pliku w katalogu, który jest pod kontrolą `automountera`, powoduje chwilowe zawieszenie tej operacji, i **zamontowanie** (*mount*) odpowiedniego katalogu, czyli uzyskanie do niego dostępu i włączenie go do struktury katalogów we właściwym miejscu, w przypadku katalogów użytkowników typowo w katalogu /home. `Automounter` monitoruje użycie katalogów użytkowników, i odłącza (*umount*) je po wylogowaniu użytkownika.

Zatem w systemach korzystających z `automountera` do montowania katalogów użytkowników, informacja o lokalizacji katalogu użytkownika jest jednym z dodatkowych atrybutów konta.

Name service switch

Ponieważ szereg mechanizmów może być źródłem informacji o atrybutach konta użytkownika, narzędzia systemowe muszą być dostosowane do komunikowania się z serwisami: NIS, NIS+, i LDAP. Należą do nich, między innymi: program `login` tworzący terminalową sesję użytkownika (przy włączaniu się przez sieć), program `passwd` pozwalający użytkownikom zmieniać swoje hasło, i inne.

Współczesne systemy uniksowe posiadają plik konfiguracji serwisów nazw /etc/nsswitch.conf. Obejmuje różne serwisy, i jest rodzajem pojemnika na „różne różności". W zakresie kont użytkownika, obejmuje: bazę haseł (passwd), drugą bazę haseł (shadow), bazę grup, i bazę katalogów użytkowników (automount).

```
hera-500> cat /etc/nsswitch.conf
...
passwd:         compat ldap
group:          compat ldap
shadow:         compat ldap
automount:      files ldap
```

Istnieje polecenie `getent` pozwalające odczytywać wpisy w poszczególnych bazach, zgodnie ze specyfikacją `/etc/nsswitch.conf`. Niestety, nie jest napisane całkiem ogólnie i nie obejmuje wszystkich baz, w tym np. automount.

Grupy użytkowników

Mechanizm grup pozwala nadawać użytkownikom uprawnienia do plików dyskowych nie będących ich własnością. Na przykład, wykładowca może udostępniać pliki tylko studentom danego kursu, jeśli należą do grupy tego kursu. (Oryginalnie było to wykorzystywane do tworzenia „projektów”.)

Mechanizm ten jest niewystarczający, ponieważ każdy użytkownik należy tylko do jednej grupy. Student nie mógłby w ten sposób korzystać z dostępu do plików innych kursów. Istnieje więc mechanizm grup dodatkowych.

```
/etc/group
root::0:root
other::1:
bin::2:root,bin,daemon
sys::3:root,bin,sys,adm
adm::4:root,adm,daemon
```

```
/etc/group
...
floppy:x:19:witold,tyciu,maciek,pludwiko,kosiu
dialout:x:20:uucp,witold,tyciu,psadowsk
cdrom:x:24:hallor,psadowsk,pludwiko,tyciu
```

Koncepcja grup dodatkowych oryginalnie przewiduje dostęp do nich przez hasło. Jednak ten mechanizm jest mało wygodny, i we współczesnych systemach sam wpis do grupy dodatkowej w pliku `/etc/group` wystarcza do uzyskiwania dostępu. Jednak wygoda tworzenia praw dostępu tą metodą nadal jest niska, zwłaszcza dla wielu użytkowników i dynamicznie powstających grup.

Nabywanie uprawnień innych użytkowników

Przypomnijmy: każdy proces ma identyfikator użytkownika, który go uruchomił. Może również mieć zdefiniowany drugi identyfikator użytkownika (lub grupy), i uzyskiwać uprawnienia tego drugiego użytkownika (grupy).

Odbywa się to przez ustawienie specjalnych bitów praw dostępu dla pliku z programem. W ten sposób działa w systemach uniksowych wiele programów systemowych, które potrzebują praw specjalnych dostępu. Na przykład, użytkownik może zmienić swoje hasło programem `passwd`. Program ten modyfikuje plik `/etc/passwd` dzięki mechanizmowi set-uid.

Zwykły użytkownik również może w ten sposób nadać swoje uprawnienia jakiemuś programowi, i w ten sposób przekazać je każdemu użytkownikowi, który uruchomi ten program. (Pozostaje jednak problem: jak ograniczyć prawo uruchamiania programu do określonej grupy użytkowników?)

Uwaga: mechanizm set-uid nie działa dla skryptów, ponieważ uruchamianie skryptów w rzeczywistości uruchamia interpreter poleceń, który interpretuje dany skrypt. Gdyby jednak nadać uprawnienia set-uid samemu interpreterowi, to możnaby następnie wykonywać nim dowolny inny skrypt lub polecenie, i korzystać z uprawnień set-uid.

System quota

System *quota* jest mechanizmem ograniczania użytkownikom wielkości dysku, jaki mogą zajmować, jak również liczby plików, jakie mogą utworzyć. Limity określane są dwustopniowo: miękkie limity mogą być przez użytkownika przekroczone, i powodują jedynie wyświetlanie ostrzeżeń, oraz mogą spowodować blokadę konta po ustalonym czasie.

Przekroczenie twardego limitu jest niemożliwe, i może mieć nieprzyjemne skutki, np. niemożność zapisania pliku w edytorze. Oczywiście, użytkownik zwykle może łatwo uniknąć tych skutków przez skasowanie części swoich plików.

Limity określane są oddzielnie dla każdego systemu plików, i mogą być również sprawdzane i wymuszane dla systemów plików zaimportowanych przez sieć. Jednak uwaga na zjawisko „oddawania" swoich plików innym użytkownikom.

Polecenia systemu *quota*:

```
quota [-v] - sprawdza ustawienia limitów uzytkownika i aktualne wykorzystanie
repquota   - sprawdza limity dla katalogów w systemie plików
quotaon    - polecenie administracyjne: wlącza system quota
quotaoff   - polecenie administracyjne: wylącza system quota
edquota    - polecenie administracyjne: pozwala modyfikować limity uzytkowników
```

Hasła

W systemach unixowych wprowadzono oryginalny system ochrony kont użytkowników hasłami. Hasła są szyfrowane jawnym, znanym algorytmem, który nie ma operacji odwrotnej, i przechowywane w postaci zaszyfrowanej w ogólnie dostępnym pliku haseł `/etc/passwd`.

W tej koncepcji dowolny, nieuprzywilejowany program może zweryfikować hasło podane przez użytkownika, ale nie może oryginalnego hasła odtworzyć.

Niestety, ta prosta i — zdawałoby się — piękna w swej prostocie koncepcja okazała się na wiele sposobów wadliwa.

Na przykład, każdy mógł łatwo znaleźć użytkowników, którzy hasła w ogóle nie posiadali. Początkowo można było też stwierdzić, że użytkownik posiada to samo hasło na różnych komputerach. Na koniec, pojawienie się tanich komputerów o dużej mocy obliczeniowej pozwoliło hasła zwyczajnie łamać.

Dlatego wprowadzono modyfikację tej koncepcji polegającą na usunięciu haseł z pliku haseł i umieszczeniu ich w chronionym `/etc/shadow`.

Powstał również mechanizm **starzenia** haseł konfigurowany w tym pliku. Pozwala on wymusić zmianę hasła w określonych odstępach czasu, albo zabronić zmiany hasła, jeśli w danym systemie hasła generuje administrator.

Zasady bezpieczeństwa dla haseł

- plik shadow
- zapewnienie haseł niepustych, nietrywialnych
- zakaz przechowywania haseł przez użytkowników „gołym tekstem”, nieprzesyłanie ich przez sieć, itp.
- niedopuszczenie do współużywania jednego konta/hasła przez różne osoby
- używanie różnych haseł na różnych komputerach (zwłaszcza root)
- blokada logowania się przez sieć na konto root
- zakaz logowania się bez podania hasła (.rhosts, klucze)
- okresowe zmienianie haseł — „starzenie” haseł (ang. *password aging*)
- wymuszanie „mocnych” haseł
- inne, mocniejsze zabezpieczenia w razie potrzeby

Podstawowe zasady bezpieczeństwa

Niezależnie od istniejących w danym systemie zabezpieczeń, trzeba pamiętać, że ostatecznie bezpieczeństwo konta zależy od istniejących praktyk i postępowania użytkowników. Warto mieć na uwadze poniższe zasady.

> Bezpieczeństwo jest odwrotnością wygody.

Z jednej strony, trzeba liczyć się z koniecznością pewnych niewygód, jeśli chcemy uzyskać podwyższone bezpieczeństwo konta lub systemu. Z drugiej strony, ustawianie zbyt wysokich wymagań bezpieczeństwa tworzy niewygody, i często zdarza się, że aby pokonać te niewygody, zasady bezpieczeństwa są łamane!!

> System jest bezpieczny tylko w takim stopniu,
> jak jego najmniej bezpieczny element.

Tworzenie zabezpieczeń niekoniecznie podnosi bezpieczeństwo systemu. Aby podnieść ogólny poziom bezpieczeństwa, trzeba szukać luk w zabezpieczeniach, i je zamykać. Zamiast tego, czasami wprowadza się pewne standardy zabezpieczeń, które komuś „pasują”, ale w rzeczywistości nic nie wnoszą.

System szyfrowania kluczem publicznym PGP i GnuPG

System PGP (*Pretty Good Privacy*) został stworzony przez Philipa Zimmermanna w celu udostępnienia zwykłym użytkownikom systemów komputerowych dobrodziejstw wynikających z zastosowania nowoczesnych algorytmów szyfrowania z wykorzystaniem kluczy publicznych. Zaproponowana przez niego koncepcja okazała się na tyle dobra, że program natychmiast został zaakceptowany i powstała społeczność jego użytkowników, jak również serwery kluczy publicznych tego systemu.

Jednak wkrótce pojawiły się problemy, wynikające z szybkiego postępu w tworzeniu nowych algorytmów szyfrowania, i chęci ich użycia w systemie PGP, a z drugiej strony wprowadzanej ich ochrony prawnej przez różne firmy (patenty), a także blokowania przez rząd amerykański możliwości eksportowania z U.S.A. oprogramowania zawierającego nowe algorytmy szyfrowania. Powstawały nowe wersje oprogramowania PGP, lecz niektóre z nich naruszały istniejące patenty, a inne nie mogły być rozpowszechniane z terenu U.S.A., i z konieczności nie wszystkie były ze sobą kompatybilne.

Dobrym rozwiązaniem tych problemów jest system GnuPG, kompatybilny z systemem szyfrowania PGP, i należący do niego program gpg, z interfejsem użytkownika zgodnym z oryginalnym programem pgp.

System PGP i GPG — tworzenie i przesyłanie kluczy

1. Wybranie kartoteki na pliki konfiguracyjne (musi być zabezpieczona):

```
PGPPATH=~/.pgp                  # domyslna dla PGP
GNUPGHOME=~/.gnupg              # domyslna dla GnuPG
export PGPPATH GNUPGHOME
```

2. Wygenerowanie sobie kompletu kluczy: prywatnego i publicznego.

```
pgp -kg
gpg --gen-key
```

3. Wprowadzenie czyjegoś klucza publicznego do bazy danych.

```
pgp -ka plik_z_czyims_kluczem
gpg --import plik_z_czyims_kluczem
```

4. Utworzenie pliku tekstowego z własnym kluczem publicznym.

```
pgp -kxa user_id plik_z_kluczem
gpg --export --armor --output plik_z_kluczem user_id
```

System PGP i GPG — szyfrowanie

1. Zaszyfrowanie danych w pliku czyimś kluczem publicznym

   ```
   pgp -eat plik_danych mw ekr
   ```

   bezpośrednie wysyłanie zaszyfrowanych danych programem mailx

   ```
   cat tresc_listu | pgp -eatf ekr | mailx -s "list ode mnie" ekr
   ```

   zaszyfrowanie "w locie" treści listu pisanego pod programem mailx

   ```
   ~| pgp -eatf ekr
   ```

2. Szyfrowanie symetryczne (jednym kluczem)

   ```
   pgp -c plik_danych
   ```

System PGP i GPG — deszyfrowanie i podpisy cyfrowe

```
pgp nazwa_pliku
cat cos_tam | pgp
```

z poziomu programu mailx

```
pipe 'pgp -m'
```

Podpisy cyfrowe:

```
pgp -s nazwa_pliku
pgp -sta nazwa_pliku    # wygodne z flaga CLEARSIG
pgp -esta nazwa_pliku witold
```

Plik konfiguracyjny `$PGPPATH/config.txt`

```
CLEARSIG = on
TEXTMODE = on
ARMOR = on
```

System PGP i GPG — konfiguracja i manipulacje kluczami

1. wyświetlanie zapamiętanych kluczy publicznych

   ```
   pgp -kv
   gpg --list-keys
   ```

2. wyświetlanie zapamiętanych kluczy prywatnych (tylko sygnatur)

   ```
   gpg --list-secret-keys
   ```

3. Sprawdzanie przez telefon sygnatury klucza

   ```
   pgp -kvc witold
   gpg --list-sigs
   ```

Szyfrowane połączenia: `ssh` i `scp`

Programy ssh/scp umożliwiają połączenia terminalowe i kopiowanie plików, z wykorzystaniem szyfrowania kluczem publicznym. Wymaga to wygenerowania pary kluczy szyfrowania dla komputera, do wstępnego nawiązywania połączenia. Ponadto, każdy użytkownik musi mieć wygenerowaną własną parę kluczy:

```
> ssh-keygen
...
> ls -l ~/.ssh:
total 36
-rw-------   1 witold gurus    826 2007-11-25 08:50 authorized_keys
-rw-r--r--   1 witold gurus     15 2007-11-25 10:47 config
-rw-------   1 witold gurus   1675 2007-11-25 08:46 id_rsa
-rw-r--r--   1 witold gurus    630 2008-06-04 16:52 id_rsa.keystore
-rw-r--r--   1 witold gurus    396 2007-11-25 08:46 id_rsa.pub
-rw-------   1 witold gurus  14160 2008-10-24 10:16 known_hosts
```

Plik z kluczem prywatnym może być zabezpieczony hasłem (*passphrase*). Hasło trzeba podawać każdorazowo przy korzystaniu z klucza prywatnego (pewne udogodnienie zapewnia program `ssh-agent`, o czym za chwilę). Można zrezygnować z hasła, wtedy klucz jest chroniony tylko prawami dostępu do pliku.

Nawiązywanie połączenia

Normalnie przy nawiązywaniu połączeń ze zdalnymi systemami za pomocą `ssh` lub `scp`, zdalny system wymaga autoryzacji przez podanie hasła:

```
shasta-3184> scp -p ~/mozilla.ps sierra:/tmp
The authenticity of host 'sierra (172.16.0.1)' can't be established.
RSA key fingerprint is 1b:9d:e6:cd:df:fd:ac:2a:6c:40:bd:0b:40:2b:50:9d.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'sierra,172.16.0.1' (RSA) to the list of known hosts
witold@sierra's password:
mozilla.ps           100% |*****************************|   937 KB    00:01
```

W tym przypadku okazało się, że program `ssh` na komputerze lokalnym (shasta) nie znalazł zapamiętanego klucza publicznego komputera zdalnego (sierra), i ostrzegł użytkownika, że zapyta bezpośrednio zdalny komputer o jego klucz. Następnie zapisał otrzymany zdalny klucz w pliku lokalnym. Normalnie jest bezpieczniej użyć od razu klucza publicznego zdalnego komputera nawet do nawiązania wstępnego połączenia, ze względu na możliwość podszywania się.

Przy kolejnych połączeniach system znajdzie i użyje zapamiętany klucz publiczny zdalnego komputera. Warto zwrócić uwagę na ten szczegół, ponieważ jest to pierwszy stopień zabezpieczenia programu `ssh`.

Nawiązywanie połączenia (2)

```
-bash-3.00$ ssh sierra.ict.pwr.wroc.pl
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
1b:9d:e6:cd:df:fd:ac:2a:6c:40:bd:0b:40:2b:50:9d.
Please contact your system administrator.
Add correct host key in /home/witold/.ssh/known_hosts to get rid of this messag
Offending key in /home/witold/.ssh/known_hosts:3

RSA host key for palnet.homeunix.net has changed and you have requested strict
Host key verification failed.
```

W tym przypadku klucz przysłany przez zdalny serwer nie zgadza się z kluczem pamiętanym dla serwera przez program `ssh`. Może to być wynikiem zmiany klucza na zdalnym serwerze (np. w wyniku reinstalacji), ale może sygnalizować fałszerstwo.

„Odcisk palca" klucza

Aby potwierdzić poprawność klucza zdalnego systemu, do którego nie mamy innego dostępu niż połączenie `ssh` na swoje konto, możemy sprawdzić jego „odcisk palca" (*fingerprint*), który jest rodzajem sumy kontrolnej, łatwej do przeczytania. Użytkownik zdalnego systemu (np. administrator) może go uzyskać następującym poleceniem, i następnie np. podyktować przez telefon:

```
ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
```

Do uzyskania odcisku palca klucza nie jest potrzebny dostęp do klucza prywatnego, do którego zwykły użytkownik nie ma dostępu, ponieważ jest to klucz całego systemu, a nie konkretnego użytkownika.

Po weryfikacji poprawności uzyskiwanego klucza, należy wykasować edytorem tekstowym stary klucz z pliku pamiętanych kluczy, co pozwoli nawiązać poprawne połączenie.

Autoryzacja połączenia

Program `ssh` obsługuje mechanizm równoważności maszyn zgodny z programami `r*`. Uwzględniany jest zarówno plik systemowy `/etc/hosts.equiv` (oraz dodatkowo `/etc/ssh/shosts.equiv`), jak również pliki użytkowników `~/.rhosts` (i dodatkowo `~/.shosts`). Ten mechanizm jest bardziej bezpieczny w przypadku `ssh` niż w przypadku `r*`, ze względu na weryfikację kluczy, oraz szyfrowanie połączeń. Jednak administrator systemu może wyłączyć ten mechanizm autentykacji użytkownika, co nie blokuje obsługi połączeń `ssh`, a tylko wyklucza tę możliwość autentykacji użytkownika.

Inną metodą autentykacji jest użycie kluczy szyfrowania. Użytkownik może wpisać klucze publiczne z systemów, z których chce mieć możliwość logowania się, do pliku `~/.ssh/authorized_keys`. Połączenie jest nawiązywane automatycznie, jednak program `ssh` potrzebuje w tym procesie dostępu do klucza prywatnego. Jeśli klucz prywatny jest chroniony hasłem (*passphrase*), to użytkownik musi podać to hasło. Warto pamiętać, że autentykacja przez klucze może również być zablokowana przez administratora zdalnego systemu.

Ostatnią metodą autentykacji jest bezpośrednie podanie przez użytkownika hasła do konta na zdalnym systemie. Ta metoda nie jest na ogół blokowana. Hasło jest przesyłane bezpiecznym szyfrowanym kanałem.

ssh-agent

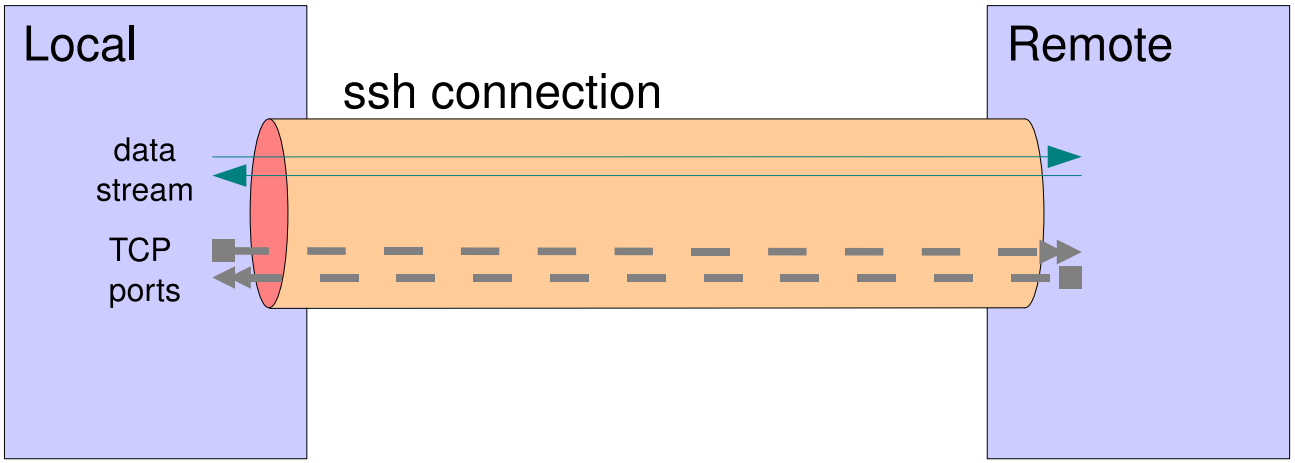
W przypadku gdy klucze prywatne użytkownika są chronione hasłem (*passphrase*) użytkownik musi podać to hasło przy każdorazowym nawiązywaniu połączenia, co jest równie niewygodne, jak podawanie hasła do zdalnego konta, i wygoda wynikająca z logowania się przez klucze znika.

Istnieje sposób na usprawnienie tego procesu za pomocą programu `ssh-agent`. Program ten służy do przechowywania kluczy w pamięci, i dostarczanie ich przy każdorazowej próbie nawiązywania połączenia przez `ssh`. Program `ssh-agent` po uruchomieniu instaluje się w pamięci, i tworzy zmienne środowiskowe umożliwiające innym programom kontakt z `ssh-agent` w celu zapytania o klucze. Klucze dodaje się programowi `ssh-agent` programem `ssh-add`.

Jak z tego wynika, program `ssh-agent` będzie najlepiej spełniał swoje funkcje, jeśli będzie korzeniem całej sesji użytkownika, czyli przodkiem wszystkich jego procesów. W przypadku logowania terminalowego można to zorganizować poleceniem typu: `"exec ssh-agent $SHELL"` umieszczonym w pliku startowym interpretera poleceń (uwaga na pętle). W przypadku logowania do sesji okienkowej odpowiednie wywołanie musiałoby się znaleźć w konfiguracji sesji.

Tunelowanie połączeń przez ssh

Dodatkową, niezwykle przydatną własnością programu `ssh` jest zdolność **tunelowania połączeń**, czyli tworzenia portów sieciowych TCP po dowolnej stronie połączenia `ssh`. W chwili próby otwarcia tego portu program `ssh` tworzy port po drugiej stronie połączenia `ssh`, i połączenie TCP między nimi. Nawiązywanie połączenia i komunikacja TCP przez łącze `ssh` są szyfrowane.



Mechanizm tworzenia tuneli TCP jest „mocny” i daje wiele możliwości. Przeanalizujemy kilka przykładowych konfiguracji.

ssh: przykłady tuneli (1)

Na przykład, chcemy komunikować się ze zdalnym serwerem HTTP, który może wymagać od nas wprowadzenia danych (przez formularze), które niekoniecznie chcielibyśmy przesyłać przez sieć otwartym tekstem. Serwery HTTP często chronią takie dane udostępniając połączenia szyfrowane HTTPS. Jeśli jednak dany serwer tego nie robi, możemy sami otworzyć swoje połączenie w szyfrowanym kanale. W tym celu należy zainstalować tunel `ssh` według poniższego schematu:

```
ssh -L 8080:localhost:80 remote.ssh.server
```

Komunikacja ze zdalnym serwerem polega teraz na łączeniu się z portem 8080 na komputerze lokalnym.

ssh: przykłady tuneli (2)

W powyższym przykładzie tworzone jest połączenie ze zdalnym serwerem ssh aby umożliwić otwieranie w jego ramach połączeń do innych serwisów na tej samej maszynie. Rozważmy teraz scenariusz, że serwer `ssh` znajduje się w sieci, gdzie istnieją serwisy na innych komputerach, dostępne w sieci wewnętrznej, ale nie bezpośrednio, w Internecie. Może to wynikać z konfiguracji bramy do tej sieci, albo ze względów praktycznych.

Typowo tak są skonfigurowane serwisy, które wymagają autoryzacji otwartym tekstem, na przykład: POP3 (110), Microsoft Windows (139), albo serwisy prywatne, których nie chciano udostępnić w Internecie, tylko w sieci lokalnej, np.: CVS (port 2401), HTTP proxy (2138), i inne.

Poniższe wywołanie `ssh` otworzy połączenie, a w nim cztery tunele dla połączeń TCP do różnych serwisów, na różnych serwerach, widocznych i dostępnych ze zdalnego serwera ssh. Dodatkowy argument -N mówi, żeby program `ssh` w ogóle nie otwierał połączenia terminalowego, i obsługiwał tylko tunele TCP.

```
ssh -N -L 10110:pop3.server:110 -L 10139:windows.server:139 \
       -L 12138:squid.server:2138 -L 12401:cvs.server:2401 remote.host
```

ssh: przykłady tuneli (3)

Wcześniejsze przykłady zakładały tworzenie portów dla nowych połączeń na maszynie lokalnej. Jest to schemat często przydatny, gdy łączymy się ze zdalnym komputeram, np. za zaporą (*firewall*). Zdalny komputer pełni wtedy rolę bramy do sieci wewnętrznej, i jeśli taka brama istnieje, możemy tworzyć wygodne kanały dostępu do serwisów wewnętrznych.

Rozważmy teraz inny scenariusz, kiedy ponownie chcemy uzyskać dostęp do sieci wewnętrznej za zaporą, lecz nie istnieje komputer dostępowy przyjmujący połączenia `ssh` z Internetu. Możemy jednak utworzyć tunele jeśli posiadamy komputer wewnątrz sieci lokalnej, z którego można wykonywać połączenia na zewnątrz. Wtedy połączenie `ssh` wykonane z zewnętrznym serwerem może zainstalować na tym serwerze port pozwalający otwierać połączenie do naszej własnej maszyny, a za jej pośrednictwem do innych serwisów w sieci lokalnej.

Poniższe wywołanie utworzy tunele dla tych samych serwisów co w poprzednim przykładzie. W tym wypadku serwisy muszą być widoczne z naszego komputera, a porty o numerach > 10000 zostaną utworzone na zdalnej maszynie.

```
ssh -N -R 10110:pop3.server:110 -R 10139:windows.server:139 \
       -R 12138:squid.server:2138 -R 12401:cvs.server:2401 remote.host
```

ssh: przykłady tuneli (4)

Rozważmy teraz sytuację, kiedy chcemy nawiązać połączenie `ssh` pomiędzy komputerami, pomiędzy którymi nie istnieje możliwość bezpośredniego połączenia. Możemy wtedy zbudować tunele przez kolejne połączenia (`klient->brama1->brama2->serwer`), które zapewnią otwieranie zagregowanego połączenia `ssh`:

```
# polaczenie z "klient" przez "brama1" i "brama2" do "serwer"
# na klient:
ssh -L 12345:localhost:12345 brama1
# na brama1:
ssh -L 12345:localhost:12345 brama2
# na brama2:
ssh -L 12345:localhost:22    serwer
```

Efekt jest taki, że wywołanie: `ssh -p 12345 localhost` na komputerze `"klient"` otwiera połączenie `ssh` do komputera końcowego `"serwer"`. W celu kopiowania plików pomiędzy tymi komputerami należy wywołać polecenie `scp -P 12345 ...` (w tym przypadku niestandardowy port podajemy przez argument duże „P").

ssh: przykłady tuneli (5)

W poprzednim przykładzie zestaw tuneli `ssh` pozwolił nawiązywać połączenia `ssh` pomiędzy docelowymi komputerami, gdzie bezpośrednie połączenie nie było możliwe. Rozważmy teraz trochę inny wariant tej sytuacji. Chcemy połączyć komputery klient1 i klient2, które znajdują się w sieciach lokalnych, niedostępnych z Internetu, za pomocą komputera serwer dostępnego w Internecie. Musimy tworzyć tunele od strony izolowanych klientów:

```
# polaczenie z "klient1" przez "serwer" do "klient2"
# na klient1:
ssh -L 12345:localhost:12345 serwer
# na klient2:
ssh -R 12345:localhost:22    serwer
```

Powyższe tunele umożliwiają połączenia `ssh` i `scp` z klient1 na klient2, oczywiście tylko dopóty, dopóki oba połączenia istnieją. Aby umożliwić nawiązywanie połączeń między tymi klientami w odwrotnym kierunku, należy dodać odpowiedni komplet tuneli biegnących w drugą stronę.

ssh: przykłady tuneli (6)

Rozważmy ponownie dokładnie ten sam scenariusz, co w porzednim przykładzie. Chcemy połączyć komputery klient1 i klient2, w niedostępnych sieciach lokalnych, za pomocą komputera serwer dostępnego w Internecie. Jesteśmy na komputerze klient1, i jeszcze nie utworzyliśmy tunelu na tej maszynie. Wcześniej, na komputerze klient2 utworzyliśmy odpowiedni tunel:

```
# na klient2:
ssh -R 12345:localhost:22    serwer
```

Na pozór, zamiast tworzyć tunel do portu 12345, i potem łączyć się z końcówką tego tunelu na localhost, moglibyśmy bezpośrednio łączyć się z tym portem na serwer. Tak rzeczywiście jest, ale wymaga to dwóch dodatkowych operacji.

Po pierwsze, domyślna konfiguracja serwera `sshd` nie pozwala udostępniać tworzonych przez tunele portów na zewnętrznych interfejsach sieciowych. Aby tak się stało, należy ustawić flagę `"GatewayPorts yes"` w konfiguracji `sshd`.

Po drugie, operacje tworzenia tuneli (`-L`, `-R`, i `-D`) domyślnie uaktywniają port (bind) tylko na interfejsie wewnętrznym (loopback), pomimo ustawionej powyższej flagi serwera. Aby uaktywnić port również na interfejsach zewnętrznych, należy dodatkowo użyć opcji `-g` w powyższym wywołaniu `ssh`.

Tunele ssh — uwagi

W powyższych przykładach tworzone były tunele `ssh` z portem dostępowym na interfejsie sieciowym localhost. Powoduje to łączenie się protokołem `ssh` z adresem i portem lokalnego komputera, w rzeczywistości nawiązując połączenie z innym systemem. Niestety, zaburza to system zapamiętywania w pamięci podręcznej kluczy publicznych do zdalnych komputerów. Pierwsze takie połączenie powoduje przypisanie zdalnego klucza maszynie `"localhost"`. Jeśli kolejne połączenie odbywa się przez inny tunel do innego komputera, to klucze się nie zgadzają, i `ssh` protestuje.

Jednym sposobem, dość doraźnym i radykalnym, radzenia sobie z problemem polega na każdorazowym kasowaniu poprzednich kluczy, po czym `ssh` zapamiętuje nowe.

Innym sposobem, bardziej systematycznym ale uciążliwym, jest utworzenie pliku pamięci podręcznej zdalnych kluczy publicznych oddzielnie dla każdego połączenia, i podawania ścieżki do odpowiedniego pliku przy każdym połączeniu. Wywołanie `ssh` nie ma argumentu dla bezpośredniego zadawania ścieżki tego pliku, jest tylko flaga konfiguracyjna (`GlobalKnownHostsFile`). Można ustawiać ją chwilowo w wywołaniu `ssh` za pomocą opcji `-o`.

Komunikacja z klientami X Window przez tunel ssh

Często przydatnym schematem, gdzie tunelowanie TCP jest przydatne jest łączenie się ze zdalnymi klientami X Window. Powody tworzenia takich konfiguracji są podobne jak dla innych tuneli: chęć zapewnienia bezpieczeństwa połączenia przez szyfrowanie, lub chęć uzyskania dostępu dla określonych usług w sieci, której połączenie nie przepuszcza takich usług.

Aby po połączeniu się ze zdalnym komputerem uruchamiać na nim klienty X Window trzeba zapewnić możliwość łączenia się przez nie z portem X Window naszego lokalnego komputera (port 6000 dla serwera :0). Odpowiedni tunel `ssh` można stworzyć za pomocą odpowiednio skonstruowanego argumentu `-R`.

Program `ssh` posiada jednak specjalny argument `-X`, który znajduje nieużywany na zdalnej maszynie numer serwera X Window, tworzy tunel z portu tego „serwera" (serwer taki istnieje tylko wirtualnie, w postaci portu, obsługiwanego przez `ssh`) do lokalnego serwera X Window, a w środowisku zdalnej sesji tworzy zmienną środowiskową `DISPLAY` określającą wyświetlanie na takim wirtualnym serwerze X Window.