2. SERVERUL SSH

2.1. Protocolul SSH

2.1.1. Concepte

SSH este un protocol client-server care foloseste resurse criptografice pentru a crea un canal securizat de comunicatie intre doua aplicatii in retea. El a fost gandit initial sa inlocuiasca *telnet*, *rlogin* si *rsh* – solutii Unix care ofereau accesul de la distanta la linia de comanda sau resursele unei alte statii. Problema protocoalelor vechi este ca transmiteau toate datele in clar (inclusiv username-ul si parola folosite la autentificare), ceea ce facea informatiile usor de capturat folosind un sniffer/analizor de protocol. SSH ofera un set de servicii mai bogat decat protocoalele originare, mentinand in acelasi timp confidentialitatea si integritatea datelor.

Exista doua versiuni majore de protocol SSH: 1 si 2. Versiunea initiala este considerata in ziua de astazi depasita, ea prezentand vulnerabilitati care o fac de evitat. Versiunea 2 reprezinta o rescriere a protocolului care adauga flexibilitate si securitate, noua specificatie fiind standardizata de catre IETF in cadrul catorva documente de tip RFC (notabile fiind RFC 4250-4256).

Software-ul client/server folosit pentru SSH a evoluat si el: versiunea initiala, dezvoltata de catre creatorul protocolului, a devenit de la un punct proprietara; ultima sa varianta open-source a fost preluata de catre comunitatea Unix si transformata in OSSH care ulterior a devenit OpenSSH-ul zilelor noastre.

OpenSSH ofera o intreaga gama de facilitati, care cuprind si depasesc cu mult capabilitatile vechilor utilitare Unix telnet, rcp, rsh, rlogin:

- accesul securizat la un interpretor de comenzi ruland pe o masina aflata la distanta, prin retea/internet
- copierea securizata de fisiere prin retea/internet
- port forwarding (un port al masinii client este redirectionat prin canalul criptat creat de SSH catre un port al masinii pe care ruleaza serverul SSH)
- redirectionarea output-ului aplicatiilor grafice rulate pe server, prin canalul criptat SSH, catre serverul grafic al clientului

2.1.2. Arhitectura

Intern, protocolul SSH este format dintr-o suita de sub-protocoale, fiecare cu un rol bine stabilit:

- protocolul de transport – este cel care implementeaza partea de stabilire a canalului de comunicatie criptat (schimbul de chei, negocierea parametrilor canalului (algoritmi de criptare, verificare integritate) etc). Acest canal este folosit ca baza pentru serviciile oferite de restul de protocoale componente
- protocolul de autentificare a clientului – implementeaza diferite mecanisme de autentificare prin care serverul verifica identitatea clientului: host-based, public key, password etc (vezi capitolul dedicat autentificarii)
- protocolul de gestionare a conexiunilor – este cel care permite definirea, in cadrul unui canal de comunicatie deschis de protocolul de transport, a mai multor sesiuni independente. Fiecare dintre ele poate fi folosita pentru un serviciu separat - un shell interactiv, o redirectionare de port (client--> server sau server-->client) etc.

2.2. OpenSSH

2.2.1. Utilizarea clientului ssh

2.2.1.1. Sintaxa generala

Distributia OpenSSH cuprinde tot software-ul necesar crearii unui server SSH si accesarii lui:

- soft de server SSH – executabilul **sshd**
- soft de client ssh – utilitarul numit chiar **ssh**

Prezentam in acest punct utilitarul client *ssh* deoarece prin intermediul sau se obtine accesul la functiile oferite de server, ceea ce ne permite diagnosticarea serverului in diversele sale scenarii de utilizare.

Sintaxa generala a comenzii *ssh* este urmatoarea:

```
ssh <optiuni> <username@>statie <comanda>
```

Optiuni de interes:

- **-2** – forteaza folosirea exclusiva a protocolului SSH2
- **-4** – forteaza utilizarea exclusiva de adrese Ipv4
- **-i *fisier*** – specifica fisierul ce contine identitatea clientului (cheia privata) atunci cand autentificarea clientului se face pe baza de cheie
- **-L *port_local:statie_remote:port_remote*** – realizeaza redirectionarea unui port local (de pe client) prin conexiunea criptata SSH catre un port al unei statii aflate de aceeasi parte a conexiunii cu serverul
- **-R *port_server:statie_locala:port_local*** – redirectionarea unui port de pe server prin conexiunea criptata catre un port al unei statii aflata de aceeasi parte a conexiunii ca si clientul
- **-o *optiune*** – folosit pentru a seta optiuni la lansarea in executie. Este gandit pentru directivele din fisierul de configurare al ssh care nu au optiuni corespondente in linia de comanda
- **-p *port*** – specifica portul serverului la care se va conecta clientul ssh, atunci cand acesta difera de cel implicit (22)
- **-v** – activarea modului verbose. Optiunea poate fi folosita de pana la 3 ori in cadrul aceleiasi comenzi *ssh*, crescand de fiecare data gradul de detaliu al informatiilor afisate
- **-Y** – activeaza X11 forwarding (redirectionarea output-ului aplicatiilor grafice rulate pe serverul SSH catre serverul grafic X al clientului)

La conectare, clientul SSH ii transmite serverului cu ce identitate doreste sa se logheze. Aceasta poate fi aceeasi cu care utilizatorul s-a logat pe masina client sau una diferita. De aceea username-ul prezent in sintaxa comenzii este optional: daca nu este specificat, conectarea se va efectua cu username-ul curent logat (cel care executa comanda ssh). **Atentie!** Protocolul SSH transmite serverului atat username-ul utilizatorului curent, cat si cel dorit, pentru a putea verifica daca o identitate de pe client are voie sa acceseze una diferita de pe server! De aceea in unele scenarii de autentificare mai deosebite va exista o diferenta intre urmatoarele doua comenzi:

```
# user curent: student; login-ul pe serverul SSH se realizaza cu acelasi username
student@srv$ ssh 10.0.0.100

# user curent: root; login-ul pe serverul SSH se realizeaza cu username-ul student
root@srv# ssh student@10.0.0.100
```

Daca este specificat si ultimul argument din sintaxa (comanda) atunci respectiva comanda va fi cea executata pe server dupa etapa de autentificare. Cand ultimul argument lipseste se executa implicit un shell de login.

Exemple de utilizare:

```
# conectarea la un server care ruleaza pe un port nestandard
ssh -p 10022 root@10.0.0.100

# redirectionarea portului local 59000 catre portul 5900 al unei statii aflate in
# partea cealalta a conexiunii SSH. Serverul SSH are adresa 10.0.0.100 iar statia catre care se
# face redirectionarea 10.0.0.200
```

```
ssh -L 59000:10.0.0.200:5900 root@10.0.0.100
```

2.2.1.2. Fisiere de configurare

Clientul *ssh* citeste la pornire optiuni de configurare din trei surse posibile, in aceasta ordine:

- optiuni pasate in linia de comanda la apelare
- un fisier de configurare per-user. Locatia sa implicita este *~/.ssh/config*. Utilizatorul poate specifica un alt fisier per-user cu ajutorul optiunii *-F* al comenzii *ssh*. Atentie insa! In acest ultim caz, fisierul global va fi ignorat!
- fisierul global */etc/ssh/ssh_config*

In cazul in care acelasi parametru de configurare se regaseste in mai multe locuri, va fi folosita prima valoare a parametrului, conform ordinii specificate mai sus.

Observatie: *privind lucrurile dintr-o alta perspectiva, optiunile setate intr-un cadru mai particular au prioritate fata de cele mai generale. Cele globale sunt modificate/anulate de catre cele per-user, iar acestea din urma sunt la randul lor modificate/anulate de catre cele pasate la apelare.*

Fisierele contin directive de configurare de forma *Parametru valoare*. Doar o parte dintre directive dispun de optiuni corespondente in linia de comanda; restul nu pot fi setate decat din fisiere.

2.2.2. Fisa serverului

Executabil: **sshd**
Fisier de configurare global: **sshd_config**, aflat de obicei in */etc/ssh* sau */etc/sshd* (depinde de distributie)
Director cu fisiere de configurare per-user: **~/.ssh**
Validare sintactica a fisierului de configurare: **sshd -t**
Utilitar pentru generare de chei criptografice: **ssh-keygen**
Utilitar de tip client pentru conectarea la un server OpenSSH: **ssh**
Rulare server in foreground cu afisarea mesajelor de logging in terminal: **`which sshd` -De**
Rularea serverului in mod debug: **`which sshd` -d**(pot fi folosite pana la 3 optiuni -d pentru cresterea detaliului)
Utilitar de colectare a cheilor publice ale altor statii: **ssh-keyscan**

2.2.3. Niveluri de configurare

Serverul OpenSSH este configurabil la doua niveluri:

- configurare globala – configurarea aplicabila tuturor clientilor, formata din fisiere plasate in */etc/ssh* (ex: *sshd_config, shosts.equiv, ssh_known_hosts etc)*
- configurare per-client - indiferent de modalitatea de autentificare folosita, clientul SSH specifica username-ul utilizatorului la resursele caruia doreste sa capete acces. Daca acest user de sistem dispune de fisiere de configurare OpenSSH per-user, serverul va lua in considerare si acele configurari, ele putand modifica/anula efectul celor globale. Fisierele de acest fel se gasesc de obicei in *~/.ssh* sau in ~; exemple: *~/.ssh/known_hosts*, *~/.shosts*

Atentie! Ca si in cazul clientului ssh, setarile din fisierele per-user sunt evaluate inaintea celor globale si pot anula/modifica efectul acestora din urma! (ex: configurarea globala permite accesul unui client, dar setarile per-user il interzic). Suprapunerea celor doua tipuri de configurare poate fi folosita pentru a crea scenarii de acces complexe.

O lista a fisierelor de interes (atat globale cat si per-user) poate fi consultata in Anexa 1 a materialului.

2.2.4. Generarea si configurarea cheilor

Pentru autentificarea/criptarea comunicatiilor sale, OpenSSH are nevoie de cel putin o pereche de chei (publica+privata). Serverul suporta atat chei RSA cat si DSA; administratorul poate configura simultan ambele tipuri de chei, serverul alegand pe care sa o foloseasca in functie de rezultatul negocierii initiale purtate cu clientul.

Cheile pot fi generate in doua moduri:

- folosind comanda dedicata *ssh-keygen*:

```
ssh-keygen -b 1024  -t rsa -f /etc/ssh/cheie_rsa
```

Optiunea *-b* specifica numarul de biti ai cheii, *-t* tipul de cheie (rsa sau dsa) iar *-f* locatia si numele fisierului in care va fi scrisa cheia privata.

Comanda de mai sus creeaza doua fisiere:

- */etc/ssh/cheie_rsa*, care contine cheia privata RSA generata
- */etc/ssh/cheie_rsa.pub*, care contine cheia publica corespunzatoare. Acest fisier nu este necesar configurarii serverului, dar este util atunci cand avem nevoie de cheia publica – spre exemplu, atunci cand autentificam clientul pe baza cheii sale publice (generarea cheilor de client facandu-se tot cu *ssh-keygen*)

- folosind openssl:

```
# cheia privata
openssl genrsa -out /etc/ssh/cheie_rsa 1024
```

Atentie! *Fisierul care contine cheia privata trebuie sa aiba permisiuni numai pentru owner! In caz contrar serverul OpenSSH emite o eroare si ignora cheia in cauza!*

Configurarea cheilor in OpenSSH se realizeaza cu directiva *HostKey* in *sshd_config*, indicand doar locatia cheii private:

```
HostKey /etc/ssh/cheie_rsa
```

Aceeasi directiva se foloseste si in cazul cheilor DSA. Daca exista atat chei RSA cat si DSA, se vor folosi doua directive *HostKey* in fisierul sshd_config:

```
HostKey /etc/ssh/cheie_rsa
HostKey /etc/ssh/cheie_dsa
```

2.2.5. Configurare generala

Configurarea generala a serverului OpenSSH se realizeaza prin intermediul unui ansamblu de directive plasate in fisierul */etc/ssh/sshd_config*. Iata-le pe cele mai importante:

- adresa/adresele si portul statiei gazda pe care asculta serverul:

```
# serverul asculta pe toate adresele definite in sistem:, pe portul 22
ListenAddress 0.0.0.0
```

```
Port 22

# serverul asculta pe o anumita adresa, portul 2222
ListenAddress 10.0.0.5
Port 2222

# alternativ, portul poate fi specificat chiar in cadrul valorii lui ListenAddress:
ListenAddress 10.0.0.5:2222
```

- versiunile de protocol acceptate in negocierea cu clientul (1, 2 sau ambele). Daca stim de la bun inceput ca toti clientii vor suporta SSH2, ideal este ca SSH1 sa fie evitat.

```
# doar SSH versiunea 2
Protocol 2

# SSH 1 si 2; ordinea nu conteaza, clientul alegand din posibilitatile oferite de server
Protocol 2,1 # echivalent cu a scrie Protocol 1,2
```

- activarea compresiei – SSH poate oferi compresia automata a datelor traficate. Masura isi are utilitatea in cazul legaturilor lente, introducand intarzieri inutile in cazul unei conexiuni rapide. Compresia poate fi dezactivata complet, activata de la bun inceput sau activata abia dupa ce clientul s-a autentificat:

```
# dezactivare
Compression no

# activare de la bun inceput
Compression yes

# activare numai dupa autentificarea clientului
Compression delayed
```

- activarea/dezactivarea DNS - atunci cand ea este activata, OpenSSH obtine prin rezolutie DNS inversa numele corespunzatoare IP-urilor clientilor si verifica daca rezolutia directa si cea inversa corespund

```
UseDNS no
```

- configurarea unui mesaj afisat clientului imediat dupa conectarea la server, inainte de a se autentifica:

```
Banner /etc/ssh/disclaimer.txt
```

- activarea/dezactivarea afisarii automate a unui mesaj de intampinare imediat dupa autentificare. Continutul mesajului este obtinut din fisierul */etc/motd* (“message of the day”):

```
PrintMotd yes
```

- afisarea imediat dupa autentificare a datei/orei precedentului login al aceluiasi client:

```
PrintLastLog yes
```

2.2.6. Logging si debugging

OpenSSH transmite informatiile sale de logging catre syslog; putem regla categoria de logging atasata fiecarei informatii pasate lui syslog si nivelul de importanta minim al informatiilor logate. Nivelurile de importanta

sunt, de la succint catre detaliat: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2, si DEBUG3 (unde DEBUG este echivalent cu DEBUG1):

```
SyslogFacility AUTH
LogLevel INFO
```

Pentru diagnosticarea sesiunilor de client, administratorul are la dispozitie diferite parghii:

- rularea serverului in mod debug
  - prin modificarea nivelului de logging din *sshd_config* si apoi analiza logurilor
  - prin rularea serverului in mod debug de la bun inceput folosind optiunea -d:

```
# folosirea o singura data a optiunii -d inseamna DEBUG1
sshd -d

# folosirea de trei ori inseamna DEBUG3
sshd -ddd
```

Rularea serverului in mod debug poate fi cuplata cu rularea in foreground, realizata cu ajutorul optiunii -D:

```
# rulare in foreground cu nivel de debug 2
sshd -Ddd
```

Atentie! *La rularea in mod debug cu optiunea -d, sshd va servi primul client si apoi se va inchide automat la incheierea sesiunii acestuia!*

- rularea clientului in mod debug, cu optiunea -v (verbose):

```
$ ssh -v student@10.0.0.5
OpenSSH_5.1p1 Debian-5, OpenSSL 0.9.8g 19 Oct 2007
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug1: Connecting to 192.168.1.33 [192.168.1.33] port 2222.
debug1: Connection established.
debug1: identity file /home/ionut/.ssh/identity type -1
debug1: identity file /home/ionut/.ssh/id_rsa type -1
debug1: identity file /home/ionut/.ssh/id_dsa type -1
debug1: Remote protocol version 2.0, remote software version OpenSSH_4.7
[...restul output-ului...]
```

2.2.7. Securizare generala OpenSSH

Exista cateva restrictii de avut in vedere pentru fortificarea unei configurari OpenSSH:

- interzicerea parolelor vide:

```
PermitEmptyPasswords no
```

- interzicerea login-ului cu username-ul root. In aceste conditii, pentru administrarea de la distanta a unui sistem, utilizatorul are doar optiunea login-ului ca user neprivilegiat si folosirea apoi a unor mecanisme de sporire de privilegii (su, sudo)

```
PermitRootLogin no
```

- stabilirea de limite pentru a preintampina atacurile de tip brute-force sau epuizarea resurselor serverului:

- cat timp i se permite unui client sa ramana conectat la server inainte de a se autentifica (implicit 120 de secunde). Clientul va fi automat deconectat dupa expirarea acestui interval
- numarul maxim de incercari de autentificare per conexiune. Un client, odata conectat, poate incerca sa se autentifice de acest numar de ori; dupa epuizarea lor va fi deconectat. De la jumatatea numarului de incercari, esecul va fi consemnat si in loguri, ca masura de siguranta
- numarul maxim de clienti conectati simultan dar neautentificati

```
# timp de login
LoginGraceTime 60

# numar maxim de incercari de autentificare per conexiune
MaxAuthTries 8

# numar maxim de clienti neautentificati conectati simultan
MaxStartups 10
```

- crearea cate unui sub-proces pentru fiecare client servit, proces care ruleaza cu privilegiile userului solicitat de client. Daca serverul a fost configurat sa nu permita accesul ca root, privilegiile de root sunt folosite numai pentru eventuala deschidere a portului pe care asculta serverul, iar clientii vor comunica cu procese care ruleaza in mod neprivilegiat:

```
UsePrivilegeSeparation yes
```

- activarea/dezactivarea capabilitatii de port forwarding, care ar permite clientilor tunelarea de trafic prin intermediul serverului SSH. Dezactivarea are sens numai in masura in care clientului nu i se permite accesul la un shell, caci in caz contrar el va avea alte modalitati de a realiza aceeasi tunelare.

```
AllowTcpForwarding yes
```

2.2.8. Configurarea autentificarii

2.2.8.1. Mecanisme

Tabelul de mai jos prezinta principalele mecanisme de autentificare suportate de OpenSSH, impreuna cu directivele de configurare corespondente. Aceste mecanisme pot fi activate sau dezactivate independent din cadrul fisierului de configurare *sshd_config* al serverului. In momentul conectarii, toate mecanismele activate sunt prezentate clientului, care la randul sau are o lista de mecanisme activate din fisierul propriu de configurare. Clientul va incerca pe rand mecanismele de autentificare comune, in ordinea specificata cu ajutorul directivei *PreferredAuthentications* din *ssh_config*:

```
# *** fisierul ssh_config de pe client ************
PreferredAuthentications  hostbased, publickey, keyboard-interactive, password
```

Directiva	Semnificatie
HostbasedAuthentication	Serverul SSH autentifica statia client si lasa in grija ei autentificarea userului (altfel spus, “ia de bun” username-ul primit, fara a-l mai autentifica, in conditiile in care statia client este trusted)
PasswordAuthentication	Mecanismul clasic bazat pe username si parola, cu posibilitatea schimbarii parolelor expirate.
PubKeyAuthentication	Serverul permite accesul clientului numai daca cheia publica prezentata

	de acesta se regaseste intr-un set de chei autorizate
ChallengeResponseAuthentication	Mecanism interactiv folosit pentru implementarea unei varietati de metode de autentificare (OTP, S/KEY etc)
RhostsRSAAuthentication	Echivalentul SSH1 al HostbasedAuthentication. De evitat.
RSAAuthentication	Echivalentul SSH1 al PubKeyAuthentication. De evitat.
GSSAPIAuthentication	Mecanism de autentificare folosit in combinatie cu Kerberos

Prezentam in continuare modul de functionare si modalitatea de implementare pentru principalele mecanisme de autentificare folosite in OpenSSH.

2.2.8.2. Autentificarea cu username si parola

Acest mod de autentificare presupune ca serverul OpenSSH va verifica identitatea username-ului primit de la client solicitandu-i acestuia parola corespondenta. Autentificarea este realizata folosind ca back-end bazele de date de sistem (passwd si shadow). Singura configurare necesara este activarea acestui mod de autentificare:

```
PasswordAuthentication yes
```

Nota: *chiar daca un utilizator foloseste combinatia corecta username/parola, accesul sau poate fi interzis prin intermediul directivelor de control al accesului la nivel de user/grup (vezi mai jos)*

2.2.8.3. Autentificarea public-key

2.2.8.3.1. Principii de functionare

Autentificarea public-key presupune ca fiecare client sa dispuna de o pereche cheie privata/publica; va fi permis accesul numai acelor clienti a caror cheie publica este autorizata. Exista doua modalitati de a desemna un set de chei de client ca fiind autorizate:

- prin listarea cheilor in fisiere de configurare dedicate. Setul de chei publice in care serverul are incredere se poate specifica intr-un fisier global sau in fisiere per-user.
- prin definirea unui CA trusted si utilizarea de certificate de client semnate de catre acest CA. In acest fel, orice certificat semnat de CA-ul in cauza este considerat trusted, fara a mai fi necesara inregistrarea pe server a fiecarei chei trusted in parte

Atentie! *Cheile declarate ca trusted sunt chei DE USER, nu de statie! Ele sunt definite pe client, de obicei in fisiere din directorul personal al userului. A nu se confunda cu cheile DE STATIE, prezentate la autentificarea host-based!*

Vom prezenta pe rand cele doua modalitati de configurare.

2.2.8.3.2. Configurarea prin declararea explicita a cheilor trusted

Configurarea acestui scenariu presupune urmatorii pasi:

- activarea mecanismului de autentificare public-key pe server si specificarea locatiei fisierului cu chei trusted. Fisierul poate fi unul global sau unul per-user; locatia sa se specifica cu ajutorul directivei *AuthorizedKeysFile*. Pentru a putea stabili locatii per-user, in cadrul valorii directivei pot fi folosite secvente speciale care desemneaza numele statiei sau directorul personal al userului. Calea catre fisier

poate fi absoluta sau relativa; in cel de-al doilea caz, ea va fi considerata a fi relativa la directorul personal al userului:

```
# activare mecanism de autentificare public-key
PubKeyAuthentication yes

# fisier global cu chei trusted
AuthorizedKeysFile /etc/ssh/authorized_keys

... sau per user, dar configurand calea din fisierul central
AuthorizedKeysFile     %h/.ssh/authorized_keys
# secvente posibile: %u – hostname, %h – directorul personal al userului
```

Atentie! Fisierul ce contine cheile autorizate trebuie sa aiba permisiunile de asa natura incat sa fie accesibil si proceselor care ruleaza ca user neprivilegiat!

- generarea cheilor de client, folosind *ssh-keygen* sau *openssl*:

```
ssh-keygen -b 1024 -t rsa -f ~/.ssh/cheie_client_rsa
```

- popularea fisierului cu chei autorizate de pe server. Operatia poate fi efectuata prin diverse modalitati (administratorul va decide in functie de caz):
  - copierea prin mijloace externe a cheii publice de pe masina client pe cea server (FTP, stick etc)
  - daca pe statia server exista deja un server SSH functional si care permite accesul clientului:
    - folosirea utilitarului *scp* pentru copierea pe server a fisierului ce contine cheia publica de client
    - folosirea utilitarului *ssh-copy-id* pentru a copia automat cheile publice de pe client in fisierul *~/.authorized_keys* de pe server:

```
ssh-copy-id -i ~/.ssh/cheie_client_rsa   username@adresa_sau_nume_server
```

Avantajul utilitarului este ca creeaza automat pe server directorul ~/*.ssh* in directorul personal al userului logat, creeaza in interiorul sau fisierul *.authorized_keys* si seteaza permisiunile corecte pe acesta. Dezavantajul este ca nu se poate specifica un alt nume de fisier per-user si de asemenea nu poate actiona asupra unui eventual fisier global (ex: */etc/ssh/authorized_keys*).

- restart de server si testare. La conectarea cu clientul *ssh* trebuie specificata cheia de client folosita:

```
# cu optiunea -i specificam identitatea dorita – primeste ca parametru calea catre cheia PRIVATA!
ssh student@192.168.1.3 -p 2222 -i ~/.ssh/cheie_client_rsa
```

2.2.8.3.3. Configurarea folosind certificate de client semnate de catre un CA

In acest scenariu de configurare vom defini un CA cu ajutorul caruia vom semna certificatele de client. Managementul certificatelor se realizeaza tot prin intermediul utilitarului *ssh-keygen*. OpenSSH foloseste un alt format al certificatului decat X509 – unul simplificat si adaptat necesitatilor serverului SSH.

Pentru configurare se vor urma in prima faza pasii prezentati in scenariul anterior:

- se activeaza autentificarea public key
- se stabileste locatia fisierului cu chei autorizate
- se genereaza chei pentru clienti

Din acest punct configurarea difera fata de scenariul precedent, deoarece nu mai este nevoie sa declaram fiecare cheie de client ca fiind trusted; singurul element adaugat in fisierul cu chei autorizate este cheia publica a CA-ului, avand grija sa o desemnam ca atare.

Operatiile necesare sunt:

- generarea unei perechi de chei pentru CA – se poate realiza folosind utilitarul *ssh-keygen* sau *openssl*:

```
ssh-keygen -t rsa -b 2048 -f cakey
```

- semnarea certificatelor de client folosind cheia privata a CA-ului. Sunt necesare doua optiuni suplimentare pasate lui *ssh-keygen*: *-s* (sign) ce primeste ca parametru calea catre cheia privata a CA-ului, si *-I* (identity) care are ca parametru un string ce reprezinta identitatea clientului. Comanda primeste ca argument calea catre cheia publica a clientului ce trebuie inclusa in certificat. Certificatul va fi depus intr-un fisier al carui nume este format automat prin adaugarea sufixului *-cert* la numele fisierului ce contine cheia publica:

```
ssh-keygen -s cakey -I client1 firstclient.pub
# ca rezultat, va fi creat fisierul firstclient-cert.pub

# continutul unui certificat poate fi vizualizat folosind optiunea -L a utilitarului:
ssh-keygen -Lf firstclient-cert.pub
firstclient-cert.pub:
        Type: ssh-rsa-cert-v01@openssh.com user certificate
        Public key: RSA-CERT 96:7b:3c:bd:16:ed:af:09:e2:db:c9:f4:ea:c4:2c:3f
        Signing CA: RSA 99:1b:f5:44:77:d6:27:58:2b:62:af:fb:c0:9a:6a:c6
        Key ID: "client1"
        Serial: 0
        Valid: forever
        Principals: (none)
        Critical Options: (none)
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

- configurarea CA-ului ca fiind trusted pentru serverul SSH. In acest scop se adauga cheia publica a CA-ului in fisierul de chei autorizate, prefixand linia ce o contine cu eticheta *cert-authority* (pentru ca serverul sa inteleaga ca aceasta nu este o cheie obisnuita, de client, ci una de CA):

```
echo cert-authority `cat cakey.pub` >> /etc/ssh/authorized_keys

# formatul liniei rezultante:
cert-authority ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAAAgQDKj+lZ+67gwEMlPTt2wtXwCD98Q/gWEnOxZ1STBnJY+ocpekj+Ek+CFvZcYU7ao92UJB
P+QaBkDyzaLRe16kHf709gdAjEKNcG5r5jn5wLi5iMVNQ1dEIxRjTlxIWGzYN1avLDgT1oVcQuGD82EHYEXGpHDDpZIdWAgo7w
nfS9Qw== admin@server
```

- copierea cheilor private/publice si a certificatelor pe clienti
- conectarea clientilor la server folosind optiunea *-i* a clientului ssh, care primeste ca parametru calea catre cheia privata a clientului si incarca automat certificatul corespunzator daca il gaseste:

```
ssh -i firstclient 10.0.0.100
```

2.2.8.4. Autentificarea host-based

In autentificarea host-based, serverul OpenSSH lasa autentificarea userului in grija statiei client; acest lucru se intampla numai pentru un set de statii autorizate, configurate explicit. Serverul nu mai incearca verificarea username-ului primit, ci doar confrunta cheia publica a statiei client cu setul de statii “trusted”. Practic este autentificata doar statia client si se ia “de bun” username-ul pe care aceasta il furnizeaza.

O statie trusted este una a carei cheie publica se afla in lista de chei cunoscute serverului (asa-numitele “known hosts” - vezi mai jos) si care in plus a fost declarata ca trusted prin listarea sa intr-un fisier special. Atat lista de chei cunoscute cat si lista de statii trusted pot fi declarate global sau per-user, cu cele per-user avand prioritate fata de cele globale:

- in cazul global, statiile cunoscute sunt declarate in */etc/ssh/ssh_known_hosts*, iar cele trusted in */etc/ssh/shosts.equiv*. Serverul nu va autentifica username-ul pentru cererile venite de la statiile trusted, indiferent de username-ul solicitat
- in cazul per-user, cheile statiilor cunoscute stau in *~/.ssh/known_hosts*, iar statiile trusted in *~/.shosts*. Atunci cand una dintre statiile trusted solicita username-ul in cauza, serverul nu il va mai autentifica; restul de utilizatori vor fi insa autentificati. In acest fel putem permite accesul la server de la distanta al unui anumit user de pe o anumita statie fara a i se mai solicita parola utilizatorului

Nota: *fisierul ~/.known_hosts este modificat de catre softul de CLIENT ssh la conectarea la un server care nu este inca “known”. Utilizatorului i se va cere acordul pentru adaugarea adresei si cheii publice a noului server in lista de statii trusted, ~/.ssh/known_hosts.*

Pasii necesari pentru configurarea autentificarii host-based sunt urmatorii:

- administratorul trebuie mai intai sa decida daca in compunerea listei de statii cunoscute (known hosts) vor fi incluse si listele per-user (specificate in *~/.ssh/known_hosts*). Exista in acest scop directiva *IgnoreUserKnownHosts*:

```
IgnoreUserKnownHosts no
```

- o a doua decizie consta in a lua in considerare listele de statii trusted definite per-user (fisierele *~/.shosts*). Foloim in acest scop directiva *IgnoreRHosts*:

```
IgnoreRHosts no
```

- colectarea cheilor publice ale statiilor “trusted”. Operatia poate fi realizata si prin mijloace manuale, insa, atunci cand statiile client ruleaza la randul lor servere OpenSSH, putem folosi in acest scop utilitarul *ssh-keyscan* pentru a le colecta in mod automatizat cheile publice. Output-ul produs de *ssh-keyscan* este potrivit pentru a fi salvat direct in fisierul global de statii trusted (sau intr-unul per-user, daca pasul anterior permite aceasta flexibilitate utilizatorilor):

```
# Determinarea cheii pentru statia 192.168.1.3
#       Semnificatia optiunilor:
#       -4 – doar adrese Ipv4
#       -T 1 – timeout-ul conexiunii setat la 1 secunda
#       -t rsa – tip de cheie obtinut (rsa1 pt SSHv1, dsa sau rsa pt SSHv2)

$ ssh-keyscan -4 -T 1 -t rsa 192.168.1.3 >>/etc/ssh/ssh_known_hosts
# 192.168.1.3 SSH-2.0-OpenSSH_5.1p1 Debian-5

# ...continutul fisierului /etc/ssh/ssh_known_hosts imbogatindu-se astfel cu inca o linie:
192.168.1.3 ssh-rsa AAAAB3NzaBIwAAAQEA2EfXk4KSPNoePxrqFGnpzcRG9oEXVs/ZHdZrgDrdbVDbnfrDOo21HwEQ==
```

- editarea fisierului */etc/ssh/shosts.equiv* si adaugarea adreselor statiilor trusted. Alternativ, daca configurarea o permite (directiva IgnoreRHosts), pot fi folosite fisierele per-user, *~/.shosts*

```
# adresele statiilor trusted, cate una pe linie
192.168.1.3
192.168.1.4
```

Nota: *multe statii pot avea statutul de "known hosts", insa numai cele din acest fisier sunt considerate trusted.*

- activarea autentificarii host-based din fisierul de configurare OpenSSH:

```
HostbasedAuthentication yes
```

- configurarea clientului ssh pentru a folosi si mecanismul de autentificare host-based, care este implicit dezactivat. Se editeaza fisierul de configurare al clientului, */etc/ssh/ssh_config* si se adauga urmatoarele directive:

```
# folosirea utilitarului ssh-keysign (care este setUID) de catre clientul ssh pt a avea
# acces la cheia privata a statiei
EnableSSHKeysign yes

# activarea autentificarii host-based
HostbasedAuthentication yes
```

Atentie! *A nu se confunda fisierul de configurare al clientului cu cel al serverului! Primul dintre ele se afla pe masina client si se numeste* ssh_config, *celalalt pe masina server si se numeste* sshd_config*!*

Nota: *in unele distributii pe care ssh-keysign nu functioneaza corect (ex: Mandriva 2008.1) este necesara activarea permisiunii speciale setUID pe executabilul client ssh astfel incat acesta sa poata citi cheia privata a statiei, folosita in autentificarea host-based.*

- testarea configurarii se efectueaza conectandu-ne cu clientul *ssh* de pe una dintre statiile trusted si verificand ca obtinem un shell fara a ni se solicita parola:

```
[student@client] # ssh 192.168.1.3 -p 2222
[student@server] $
```

ATENTIE! Login-ul ca root in cazul autentificarii host-based nu functioneaza daca statia client este declarata ca trusted in */etc/ssh/shosts.equiv*! Este necesara plasarea sa in fisierul */root/.shosts* si folosirea directivei *IgnoreRhosts no* in sshd_config!

Fisierele *shosts.equiv* sau *.shosts* contin pe fiecare linie numele sau IP-ul unei statii trusted, urmata optional de un username. Daca acesta din urma este prezent, el va avea permisiunea de a prelua identitatea oricarui alt user in afara de root. In cazul in care username-ul lipseste, clientii nu vor putea folosi sintaxe de tipul *ssh altuser@server* pentru a capata acces la server, ci li se va permite accesul numai cu username-ul lor curent.

Un exemplu concret: fie serverul 10.0.0.200 avand fisierul */etc/ssh/shosts.equiv* urmator:

```
# ***** fisierul shosts.equiv
10.0.0.100   user1
```

In aceste conditii:

- daca suntem logati ca *root* pe statia 10.0.0.100:
  - vom putea sa ne logam pe server ca root folosind comanda *ssh 10.0.0.200*
  - nu vom putea sa ne logam folosind *ssh user1@10.0.0.200*
- daca suntem logati ca *user1* pe statia 10.0.0.100:
  - vom putea sa ne logam pe server ca user1 folosind comanda *ssh 10.0.0.200*
  - vom putea sa ne logam pe server ca user2 folosind comanda *ssh user2@10.0.0.200*
  - nu vom putea sa ne logam pe server ca root, folosind comanda *ssh root@10.0.0.200*

2.2.9. Controlul accesului la nivel de user/grup

OpenSSH permite controlul mai granular al accesului folosind in fisierele de configurare directivele AllowUsers, DenyUsers, AllowGroups si DenyGroups. Fiecare dintre acestea poate aparea in mai multe exemplare in acelasi fisier de configurare. Ordinea in care sunt luate in considerare directivele este: DenyUsers, AllowUsers, DenyGroups, AllowGroups.

Iata cateva exemple:

```
# userii mihai si vlad au accesul permis, indiferent care este masina de pe care se conecteaza
AllowUsers mihai vlad

# userii victor si paul au accesul interzis, indiferent care este masina de pe care se conecteaza
DenyUsers victor paul

# userului adrian i se permite accesul numai de pe masina cu numele c10.infoacademy.net
AllowUsers adrian@c10.infoacademy.net

# folosirea de wilcard-uri: permitem accesul userilor ion, ioan sau ioana:
AllowUsers Io*n*    # simbolul * inlocuieste 0 sau mai multe caractere

# permitem accesul oricarui user care provine de pe o statie client din reteaua 10.0.0.0/24
AllowUsers *@10.0.0.*
```

2.2.10. Port forwarding

OpenSSH ofera capabilitatea de redirectionare a unor porturi de pe server sau client prin conexiunea criptata SSH. Facilitatea este implicit activata, ea putand fi dezactivata folosind urmatoarea directiva in *sshd_config*:

```
AllowTcpForwarding no
```

Pentru a intelege utilitatea acetei facilitati OpenSSH consideram urmatorul exemplu: fie o retea cu adrese private din gama 192.168.0.0/24, al carei gateway (192.168.0.1) este o statie Linux care efectueaza NAT si pe care ruleaza un server OpenSSH. Fie adresa publica a gateway-ului 1.2.3.4. Dorim ca, din afara retelei, sa accesam serverul web (port 80) care ruleaza pe statia 192.168.0.100. Statia in cauza are adresa privata si este in mod normal inaccesibila din afara retelei. Putem obtine acces la serverul web dorit conectandu-ne prin SSH la gateway si redirectionand un port de pe statia noastra catre portul 80 de pe statia tinta; presupunand ca statia noastra ruleaza Linux, putem scrie:

```
ssh -L 10000:192.168.0.100:80  1.2.3.4
```

Putem apoi accesa serverul web aflat la distanta conectandu-ne la portul local redirectionat de catre SSH, scriind in browser URL-ul:

```
http://localhost:10000
```

Nota: *redirectionarea poate functiona si in sens opus, deschizand un port pe server si redirectionand datele catre o statie/port aflate in capatul nostru de conexiune.*

2.3. BIBLIOGRAFIE

- Carti
  - SSH, the Secure Shell 2nd Edition (O'Reilly 2005)
  - Pro OpenSSH (APress 2005)
- OpenSSH host-based authentication:
  - http://www.snailbook.com/faq/trusted-host-howto.auto.html
  - https://wiki-bsse.ethz.ch/display/ITDOC/OpenSSH+hostbased+authentication+HOWTO
- OpenSSH public key authentication: http://sial.org/howto/openssh/publickey-auth/

2.4. ANEXA 1 - Fisiere de configurare implicite OpenSSH

Scop	Fisier global	Fisier per-user
Configurare server	/etc/ssh/sshd_config	X
Configurare client	/etc/ssh/ssh_config	~/.ssh/config
Cheie privata server	/etc/ssh/ssh_host_dsa_key /etc/ssh/ssh_host_rsa_key	X
Cheie publica server	/etc/ssh/ssh_host_dsa_key.pub /etc/ssh/ssh_host_rsa_key.pub	X
Statii cunoscute (known hosts)	/etc/ssh/ssh_known_hosts	~/.ssh/known_hosts
Chei trusted (pentru autentificare public-key)	X	~/.ssh/authorized_keys
Statii trusted (pentru autentificare hostbased)	/etc/ssh/shosts.equiv	~/.shosts
Cheie privata/publica client (pentru autentificare public-key)	X	~/.ssh/id_rsa, ~/.ssh/id_rsa.pub ~/.ssh/id_dsa, ~/.ssh/id_sda.pub