Secure SHell: Basics

Network Infrastructures labs

Marco Spaziani Brunella

SAPIENZA
UNIVERSITÀ DI ROMA

Lecture details

- **Readings:**
  - SSH: The Definitive Guide; D.J. Barret et al.; O'Reilly
- **Lecture outline:**
  - SSH

Remote Managing

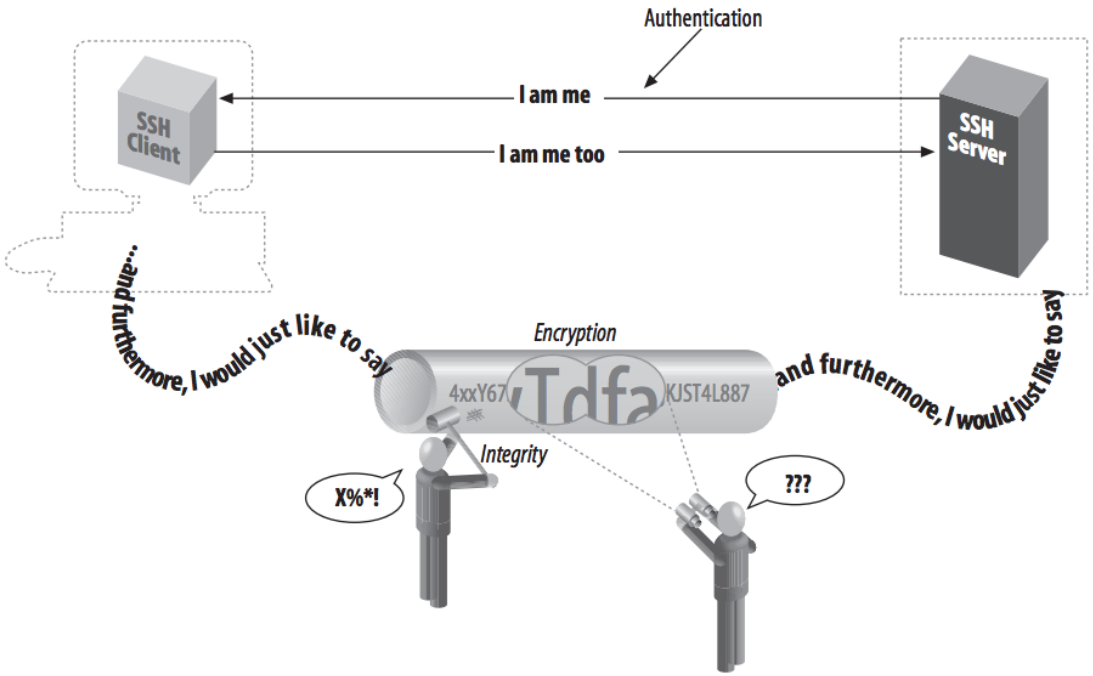
In real life, physical access to network nodes is not always an option.
Often, we need to reconfigure network nodes remotely.
Historically, the first protocol that allowed such things was *telent*.
Telnet has been now deprecated since it does not use any encryption on the sent data!

Try it yourself: open a telnet connection between two hosts and capture packets with tcpdump.

Secure SHell

SSH is a protocol that allows remote managing over a secured "pipe".Is a client-server protocol.
The main features that SSH provides are:

- Authentication
- Encryption
- Integrity

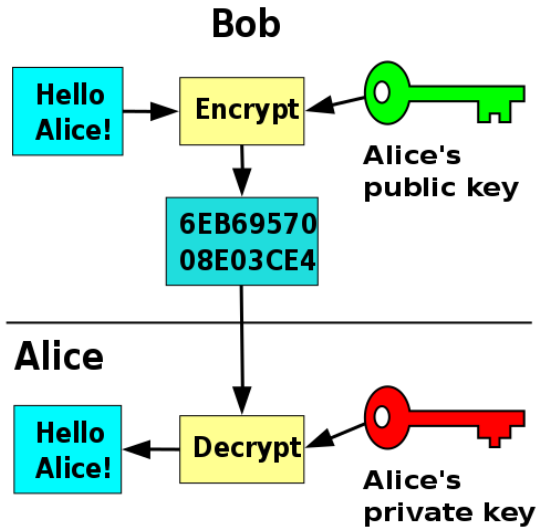


Asymmetric Cryptography

In Asymmetric Cryptography, there are 2 main ingredients:

- Public key
- Private Key

The public key is meant to be shared to everyone who want to use an encrypted language with you. The private key is meant to remain secret. The asymmetry is in the fact that the public key is used to encrypt data and the private key to decrypt them.



Starting of an SSH session

SSH runs over TCP, on the default port 22. When the client starts the session, send a packet to the server specifying the SSH version he supports.
Then, the server answers with the SSH version he supports.
After this initial handshake, the two exchange each other the encryption algorithms they support.
After that, the first that is common to both is chosen.
The client and the server now can move forward and authenticate themselves.

Authentication

The server send his public key (a.k.a the fingerprint) to the client.
The client is now asked to trust or not trust the server fingerprint.

```
iMac-di-Marco:~ marcospaziani$ ssh marcos@stud.netgroup.uniroma2.it
The authenticity of host 'stud.netgroup.uniroma2.it (160.80.221.14)' can't be established.
ECDSA key fingerprint is SHA256:UKRm/pfFevBVndWv6nAjdOpueHhFacxD5XsiswY3Q20.
Are you sure you want to continue connecting (yes/no)?
```

If it trusts the server identity, the client proceeds with authenticating himself. This step can be done in two ways:

- Username and password
- Asymmetric Cryptography

Username and Password

In this case, the client send its username and password to the server using the chosen encryption algorithm.
If the username corresponds to a user on the server and the password is correct, the server grant access to the user and the ssh session can start.

No.	Time	Source	Destination	Protocol	Length	Info
59	54.534435109	192.168.81.128	160.80.221.14	SSHv2	95	Client: Protocol (SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.2)
61	54.642808363	160.80.221.14	192.168.81.128	SSHv2	85	Server: Protocol (SSH-2.0-OpenSSH_6.6p1-hpn14v4)
63	54.645554093	192.168.81.128	160.80.221.14	SSHv2	1366	Client: Key Exchange Init
65	55.277933939	160.80.221.14	192.168.81.128	TCP	1514	[TCP segment of a reassembled PDU]
66	55.278392814	160.80.221.14	192.168.81.128	SSHv2	242	Server: Key Exchange Init
68	55.281108576	192.168.81.128	160.80.221.14	SSHv2	134	Client: Diffie-Hellman Key Exchange Init
70	62.337556567	160.80.221.14	192.168.81.128	SSHv2	366	Server: Diffie-Hellman Key Exchange Reply, New Keys
71	62.345902156	192.168.81.128	160.80.221.14	SSHv2	70	Client: New Keys
73	62.346224994	192.168.81.128	160.80.221.14	SSHv2	98	Client: Encrypted packet (len=44)

Public and private key auth

If the client has a couple of public-private key and if the server has the public key of the client, asymmetric cryptography for authentication can be used.
The server sends a "challenge" to the client, meaning that the server picks up a random string and encrypt that with the public key of the client.
This encrypted value is then transferred to the client trough the secured pipe (via the common chosen algorithm).
The client decrypts the challenge with his private key and then crypts it again with the public key of the server. Once back, the server decrypts the challenge response: if the response is the same as the random value sent, with a certain degree of probability the client is the one associated with that public key.
At this point, the session can start.

SSH on Linux

Every Linux distro has an ssh client installed called OpenSSH client. On the other hand, the server daemon, sshd, needs to be manually installed. On netkit, they are both already installed.
The configuration file of the server is /etc/ssh/sshd_config while for the client is /etc/ssh/ssh_config . To start a connection with a server:

```
ssh username@host_ip_addr
#starts ssh session with the host specified by
#host_ip_addr logging in as username
```

Lets go back to lab_0_V3.

Creating a user

Suppose we want to connect from pc1 to pc2 via ssh.
We want to create at startup a user, on pc2, specifically for ssh access.
Let's call it "ssh_user".
In order to be accessed via ssh, every user must have a password. The startup file of pc2 will be:

```
/etc/init.d/networking restart
/etc/init.d/ssh restart
mkdir /home/ssh_user
useradd ssh_user -d /home/ssh_user
chown ssh_user:ssh_user /home/ssh_user
echo -e 'ilovessh\nilovessh\n' | passwd ssh_user
~
```

Mnemonic assignment

Linux allows us to specify string literals to ip addresses inside the /etc/hosts file.
The syntax is ip_addr - tab - string.
For example, we assign the 10.0.1.100 to the string pc2 at startup on pc1:

```
/etc/init.d/networking restart
echo "10.0.1.100        pc2" >> /etc/hosts
```

After that, if we type on pc1:

```
user@localhost:$ ssh ssh_user@pc2
```

We will be prompted for password for ssh_user, which is "ilovessh".

Public-Private key generation

We want to make a step forward and use asymmetric auth.
First of all we have to generate a couple of keys on pc1:

```
user@localhost:$ ssh-keygen
```

Strike "enter" to anything is prompted on screen.
A hidden folder inside the /root dir is created, called .ssh . Inside that there are 3 files:

```
known_host #Database of the trusted fingerprints

id_rsa #private key generated using RSA

id_rsa.pub #public key generated using RSA
```

We now have to transfer the public key of pc1 to the list of trusted client keys of pc2.

Secure copy

We can transfer file s between remote hosts using ssh via the "scp" command:

```
scp local_path user@host_ip:/remote_path
#transfer file at local_path to remote_path over SSH

scp user@host_ip:/remote_path local_path
#transfer file at remote_path to local_path over SSH
```

We copy the pub_key of pc1 to pc2:

```
scp .ssh/id_rsa.pub ssh_user@pc2:/home/ssh_user
```

We could also use:

```
ssh-copyid ssh_user@pc2
```

Authorized keys

We have to put the pub_key of pc1 inside the trusted client keys database of pc2.
The "database" is the file /home/ssh_user/.ssh/authorized_keys .
Neither the folder .ssh nor the authorized_keys file exists on pc2, so we have to create them:

```
user@localhost:$ mkdir .ssh
user@localhost:$ touch .ssh/authorized_keys
```

By default, sshd is running on pc2 on strict mode, so we have to give correct permissions to .ssh and authorized_keys:

```
user@localhost:$ chmod 700 ~/.ssh
user@localhost:$ chmod 600 ~/.ssh/authorized_keys
```

We can now copy the content of id_rsa.pub inside authorized_keys:

```
user@localhost:$ cat id_rsa.pub >> .ssh/authorized_keys
```

We can now access pc2 from pc1 using pub_key authentication.