Level 0

Challenge :-

The goal of the level is we have to login into the game using ssh

Username - bandit0

host - bandit.labs.overthewire.org

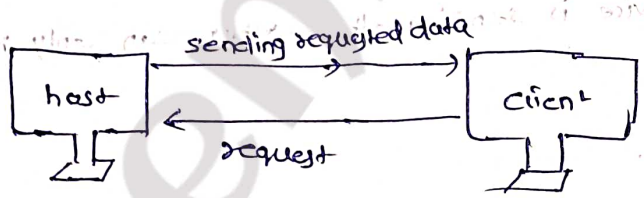
Port - 2220

SSH

↳ Secure shell (or) Secure Socket Shell

It is a cryptographic network protocol for operating network services securely over an unsecured network.

Ssh is secure because it transfers the data in encrypted form between the host and the client

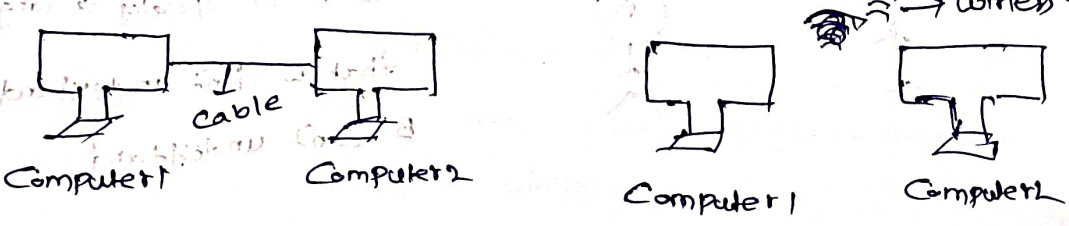


(Server/Computer which accepts a request from another Computer and send data to the requested Computer)

(It is a computer which request another computer for certain data)

What is network Protocol?

Network is the connection of atleast two computer systems, either by a cable (or) Wireless connection



Uses of network

It helps to share the resources b/w the computers

↓

applications/diskspace -etc

Data Communication

↳ means exchange of data(or) b/w to two computers via some form of link (transmission medium) such as cable

types of data flow

1) simplex

Communication is always unidirectional

one device can transmit and other device will recieve

Ex: Keyboard, Traditional monitors.

2) Half-duplex

Communication is in both directions

If one device is sending the other can only recieve and vice versa.

Ex: Walkie-Talkies.

3) Duplex/full-duplex

Communication is in both directions simultaneously

Ex Telephone

Now assume these are 2 persons (A and B) and they are talking to each other.

Now if A speaks a language that B don't understand then B can't understand

Also if A B speaks faster then A may not be able to understand.

So In order to make this correct we need to set some rules such as

1) Speak only in common language ex's say English

2) Speed 'don't Speak faster than normal talking speed (Ex 5 words per minute)

like that.

Similarly there are some rules for computer and these rules are called protocols.

Ex:- 1) What is communicated!

2) How it is communicated!

3) When it is communicated!

Protocols used in network communication also define

1) message encoding

2) message formatting and encapsulation

3) message timing

4) message size

5) message delivery options

(1) message encoding

message (source) → Encoder (Signal) → transmitter → Transmission medium → Reciever → Decoder → message (Destination)

Encoder used to create a signal/waves

Transmission medium can be a cable/wireless medium

So we use encoder to create a signal if medium is cable and waves if medium is wireless

2) message formatting & encapsulation

Agreed format (by sender & reciever)

Encapsulate the information to identify the sender and reciever

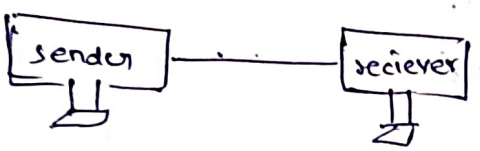
3) message site

Long messages are broken into smaller units.

4) messaging time

- ~~cyclic~~
- flow control
- response timeout

Ex



* if sender is very fast then reciever may not able understand * then entire communication become useless so the data flow should be controlled.

If the sender is sending data to reciever then reciever should send back a acknowledgment form so that sender know that data is recieved if the acknowledgment is not recieved then sender ~~keeps on sending data~~ have to wait for some time and then resend the data. This time is to be set by protocol

message delivery options

→ unicast → sending to only one device

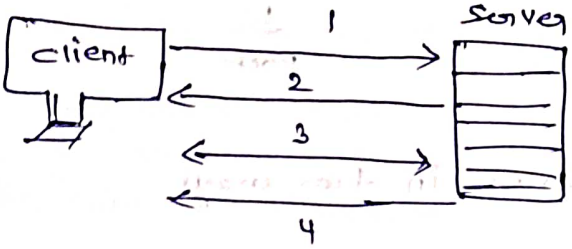
multicast → sending to multiple device but not all

broadcast → sending to all devices

Mostly we use SSH for remote login (i.e using one computer from another) and command-line execution.

How it Works

SSh applications are client-server architecture based that means it has a server and atleast one client



1 → client initialise the connection by contacting the server

2 → Server sends the public key (well we will see this later)

3 → Negotiate parameters and open secure shell

4 → user login to server host operating system

SSh uses Public-key cryptography to authenticate (allow) the remote computer i.e It uses two keys public-key & private-key Public key is distributed to clients & private key is stored on the host/server.

How to Connect to remote server

To connect to a remote server we need an SSH client

Some of them openSSH, Cygwin, Putty --Etc

for windows

openssh is installed by default if not we can install it by using below command

```
$ sudo apt install openssh-client
```

Now- we use ssh command to connect to a remote server

Syntax:

ssh username@remoteaddress

Note: If u want to add port we use -p flag

Ex: ssh username@remoteaddress -p XXXX (Port)

* default port is 22

We can log in into the server in two ways

1) using Password

2) using Public- private ssh key pair

Note: By default SSH protocol version is 2 (there many versions, SSH-1, SSH-2, SSH-wqq --etc)

how ssh key pair is generated — for this, we use ssh-keygen command

The pair of consists of a public key and private key

The public key can be shared but private key needs to stay secure.

#1 $ ssh-keygen — When u run it without any arguments

Generating public/private rsa key pair:

Enter file in which to save the key (/home/your username/.ssh/id_rsa):

This default location where the keys are stored if u don't mention any location

If u want to mention a file mention the path of file-or else Just Press enter

Now It asks for Paraphrase

→ Enter Paraphrase Paraphrase:

Note Enter a Paraphrase or Just press enter then It asks for same Paraphrase (mention the paraphrase entered above, if not mentioned Just click enter)

The key pairs are generated

Note = * by default the names of the key files are id_rsa, id_rsa.Pub
↓ Private key ↓ Public key

* By default it uses RSA algorithm

we have diff algorithms available like ecdsa, ecdsa-sk, ed25519, ed25519-sk, rsa. we can mention the type of algorithm by using -t flag.

Ex = ssh-keygen -t rsa

ssh-keygen -t ecdsa

options/flags

-t → used to mention algorithm type

-b → used to mention sized key

-f → used to mention file in which key should be stored

#2

Now change the permission of private key so that only u can read

```
$ chmod 600 ~/.ssh/id_rsa
```

#3 After that we have to copy the public key to the remote host

for this we can either use scp or ssh-copy-id

with scp

```
scp ~/.ssh/id_rsa.pub <username>@<remoteaddress>:destination
```

with ssh-copy-id

```
ssh-copy-id -i ~/.ssh/id_rsa.pub user@host
```

#4 installing public key in remote Computer

* create a folder on the remote Compute

```
$ mkdir .ssh
```

* Append your key to the authorised files

```
$ cat id_rsa.pub >> .ssh/authorised_keys
```

* change permission for the ssh folder

```
$ chmod 700 ~/.ssh
```

#5 Now u can connect to the remote computer without using Password

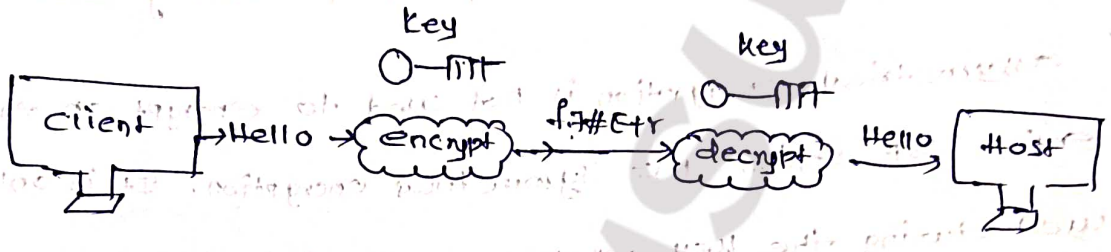
Note Connecting with Public-Private key pair is more secure than Password

SSH is more secure than other protocols such as telnet because of the encryption of the data. There are 3 major encryption techniques used by SSH

1) Symmetrical encryption

In this encryption a single key is used for both encryption & decryption.

Ex



It is also called as shared key/shared secret encryption.

It is usually one key (or sometimes a pair of key where one key can easily be calculated using the other key

* The process of creating a symmetric key is carried out by key exchange algorithm. What makes this algorithm particularly secure is the key is never transmitted b/w client and host Instead the two computers share public pieces of data and then manipulate it to calculate the secret key

Now if anyone capture these public data they won't be able to calculate the key because they don't the encryption algorithm.

There are so many encryption algorithms some examples are

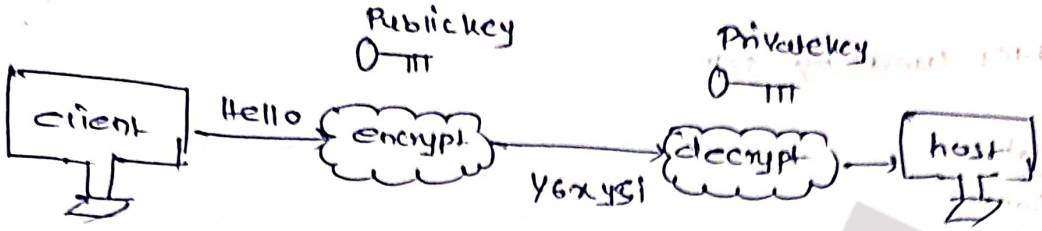
AES (Advanced encryption standard)

Before establishing the connection the client & host decide which cipher to use

Asymmetric encryption

Here we use two keys Public key & Private key

Ex



Public key can be used to encrypt data and can only be decrypted by the person who possess private key

Note

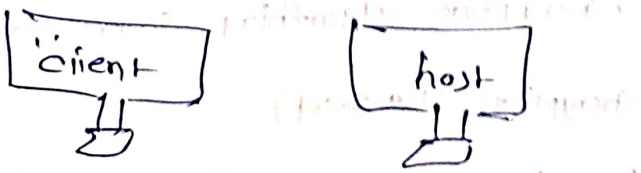
asymetrical encryption is not used to encrypt an entire SSH session. like symmetrical encryption. It is only used during the key exchange algorithm of symmetric encryption

Asymetrical encryption is used during the intial key exchange Process used to set up the symmetrical encryption. Both the Computers Produce tempory key pairs and exchange the public key in order to Produce the shared secret that will be used for symmetrical encryption.

(Public key)$_c$ (Public key)$_h$

(Private key)$_c$ (Private key)$_h$

Now they exchange Public keys



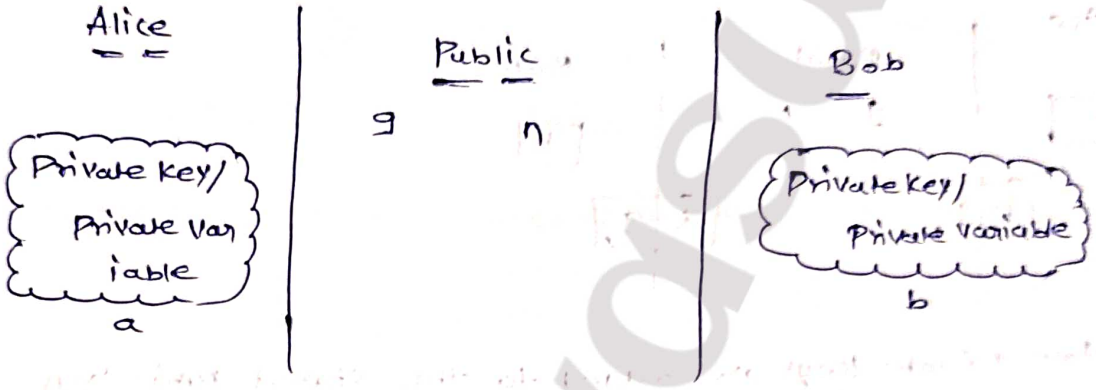
(Publickey)$_h$ (Public key)$_c$

(Private key)$_c$ (Private key)$_h$

How the Symmetrical encryption is setupped

Now after the exchange of Public keys an exchange algorithim is used to create a symmetric key. This algorithim known as Diffie-Heliman key exchange Algorithim.

Algorithim

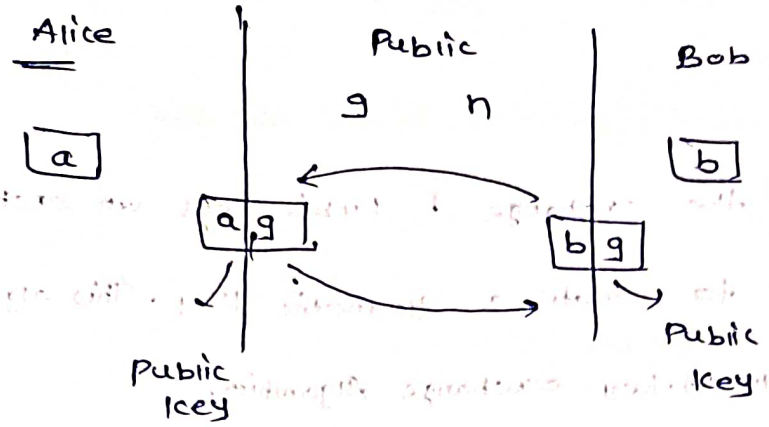


At the begining both Alice & Bob should agree certain Parmatex like generator (g) like AES-128 and a large prime number (n)

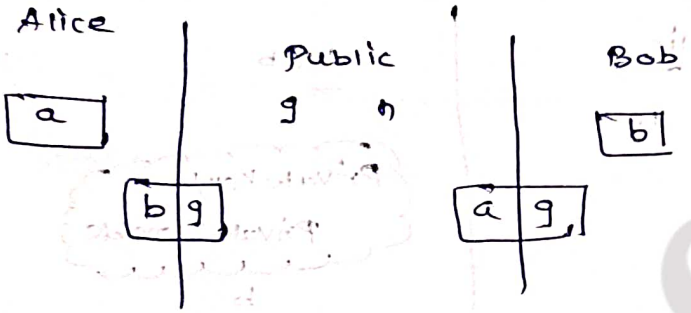
n
also known as seed value

Now both of them generate a private key/ private variable let's say a & b respectively. Now ~~ocean~~ Private key & generator are used to generate a public key.

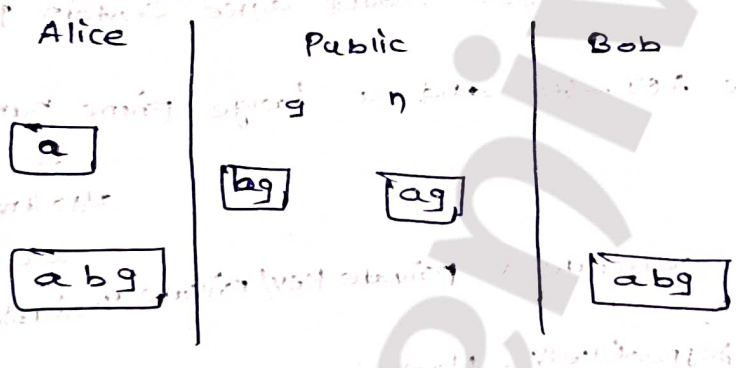
Ex Let's take a as Red color of g as yellow color and Now if we mix them we get a hybrid color (generally orange but it may vary depending on quantitiy added). Now we know g because its public and we also know as (hybrid color) but we don't know how the as is generated so we don't know the private key. To know the private key we should seperate the colors from hybrid color which is difficult



Now the Public keys are exchanged



Now Private keys are added to the shared Public keys



Now nothing in the Public helps by to obtain to a & b So it makes secure

abg is number which is used to generate secret key

• Other examples of assymetric encryption is we use it to authenticate a client as shown in ssh key Pair generation before

3) Hashing

In this we manipulate the data to create a some string

The main idea of hashing is that they are never meant to be decrypted

message + hash function ⟶ hash string

It is practically impossible to reverse the above process

∴ hashes are mainly used to check if the data is corrupted (data integrity) & authentication

SSH uses hashes to verify the authencity of messages. This is done using HMACs (Hash-based Message Authentication Codes). This ensures that the message recieved is not tampered in any way

When establishing the connection Message authentication code (MAC) algorithm is selected. Each message that is transmitted must contain a MAC which is calculated using the symmetric key, Packet sequence number and message contents.

Conclusion: SSH is more secure because it encrypts data whereas telenet, etc send data in plaintext

ssh config file ⟶ It is stored at ~/.ssh/config

The .ssh directory is created automatically when we first run ssh command if not create it:

mkdir ~/.ssh && chmod 700 ~/.ssh

By default ssh config file may not exist so we need to create it

$ touch ~/.ssh/config

$ chmod 600 ~/.ssh/config (The file must be readable & writeable & not accesible by others)

Use of ~/.ssh/config file

Now if u are connecting to the multiple remote system it becomes difficult to remember all of the remote Ip address, username, port — etc

So we store the details in this config file & when we use ssh openssh-client ready this file and use the details mentioned here to connect

Syntax/structure of the config file

```
Host hostname1
        SSH_OPTION  Value
        SSH_OPTION  Value
        !
Host hostname2
        SSH_OPTION  Value
Host *
        SSH_OPTION  Value
```

The contents are divided into Stanzas (sections) Each Stanza Starts with Host directive and Contains specific SSH options used when establishing a connection

Note = Indentation is not required but it makes the file easier to read

Ex =

```
Host bandit0
    Hostname bandit.labs.overthewire.org
    User bandit0
    Port 2220
```

Now we don't need to run whole command such as

$ ssh bandit0@bandit.labs.overthewire.org -p 2220

we can simply run

$ ssh bandit0

* The Host directive can contain one pattern or a white space seperated list of patterns. Each pattern can contain zero or more non-whitespace character or one of the following pattern specifiers:

* = → matches zero or more characters

Ex = Host xyz* means all hosts starting with xyz

xyz
xyza
xyz1 —etc

Host * means all hosts

! → matches exactly one character

Ex = we have to levels level0 to level9

then we can level? means level[0-9]

! - when used at the start of pattern it negates the match

Ex = Level1 !level5 means level0 to level9 except level5

Note =

* Ssh client read the config file stanza by stanza from top to bottom and if more than one pattern matches the options from the first matching stanza take precedence. Therefore more host specific declaration should be given at the begining of the file & more general overrides at the end of file.

* always use capital letter at start for ssh_options

  Ex Port, User, Hostname --etc

* Port 2220
  Port=2220
  Port = 2220
  } These three are identical we can use any thing. The only difference if we Port=2220 then we don't have to specify an option on the command line with quoting

Ex =

Ssh -o "Port=2220" ---
↳ for Port 2220
Port = 2220

Ssh -o Port=2220 ---
for Port=2220

Configuring Shared options

Let say we have to use the same user name for home & work then we can write

```
Host home
    Hostname example.com
    User appolo
    Port 4567

Host work
    Hostname company.com
    User appolo
```

but here we are repeating the values.

So the best way is to break the shared option out into seperate section

Ex Let sally all our machines use name apollo then we can write

```
Host home
    Hostname example.com
    Port 4567

Host work
    Hostname company.com

Host *
    User apollo
```

Now first ssh matches the host with first one (home) if it matches then option Hostname is set to example.com Port 4567 & next it checks if it matches the next Host and so on

Now Host * matches so User option is set apollo

final options are

```
Hostname example.com
User apollo
Port 4567
```

Now ssh uses these details to connect

Now let's say some host ripper of user name is not apollo then we can write Host directive for that host and set user option (Note we need to write it above Host *)

```
Host home
    Hostname example.com
    Port 4567

Host work
    Hostname company.com

Host ripper
    Hostname weird.com
    User zeus

Host *
    User apollo
```

Let's say hostname is ripper Now ssh checks for ripper 3rd directive matches it so

Hostname → weird.com
User → zeus

Now 4th also matches but user option is already set to zeus so it ignore the user option written here

final details

```
Hostname weird.com
User zeus,
```

overriding ssh config file options

The ssh client reads its configuration in the following order

1) options specified from the command line

2) options defined in the ~/.ssh/config.

3) options defined in the /etc/ssh/ssh_config.

for overriding a single option we can use -o

Ex:

```
Host dev
    Hostname dev.example.com
    User John
    Port 2322
```

Now we want to use all other options except we want to use root as user then we can write

```
$ ssh -o "User=root" dev
```

to use a diff config file we can use -F [Configfile]

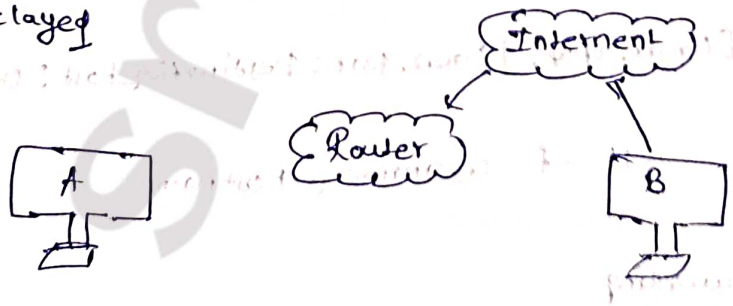
Ex: To completely ignore all the options in config file we can use

```
$ ssh -F /dev/null user@example.com
```

Port forwarding using ssh

SSH tunneling or Port forwarding is a method of creating an encrypted SSH connection between a client and server through which services ports can be relayed

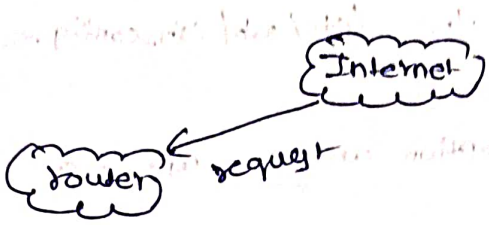
Ex:



Now B wants to connect A then B sends a request & The request contain Public IP of A & specific Port number

Let say 66094.34.13::3389

↑ Public IP ↑ Port-number



Now our router recieve that request. Now router does not know where to send that request. So we tell router send any request from port 3389 to computer A this is called Port forwarding.

But this may be insecure so with the help of ssh we create a tunnel & send this requests in the tunnel

Note:
The two ports dont have to be same.

Ex Port on B may be 4562 and Port on A maybe 2000

There are 3 types of Port forwarding

(1) Local Port forwarding

It allows u to forward connection from client to server & then to destination port

$ ssh -L [Local_IP:]Local_Port:Destination_host:Destination_Port -N -f username@host-name

(2) Remote Port forwarding

opposite to local forwarding

Server → client → destination

$ ssh -R [Remoteaddress:] Remote_port: Destination : Destination_Port -N -f username@hostname

3) Dynamic Port forwarding

Eg: ↳ It creates a socks proxy server that allows communication across a range of ports.

↓ an ssh tunnel is created and the network traffic will be send to server and ssh server sends it to destination

(It helps to browse internet privately)

$ ssh -D [Local_IP]:LOCAL_PORT -N -f username@hostname

Copying files

we use scp(secure copy Protocol) to copy files b/w local host & remote host

Copying files from local computer to remote computer

Syntax: scp <filepath> <username>@<remote>:<destination Path>

- <filepath> ↓ file path on our local computer
- <destination Path> ↓ Path where the file should be stored on remote computer

we can also mention port using -P option

scp -P . . . .

Copying files from remote computer to local computer

Syntax: scp <username@<remote>: <filepath> <destination path>

Copying files b/w two remote computers

scp user1@host1.com:<path> user2@host2.com <destination path>

To copy multiple files we can specify file names

scp file.txt file1.txt file2.txt ... user@host <path>

Note:
- By default scp use port 22
- By default scp use AES-128 algorithm to encrypt file we can mention another algorithm using -c
- If the destination path is blank then file is copied to / (root) directory
- To copy directory we can use -r flag

scp -r filepath user@host: destination

options/flags

-P → used to mention Port number

-p → accesstime, modified times, and modes are provided from original files

-C → used to copy files with compression

-c → to use another encryption algorithm

Note

* Copying file without -C Parameter will result in 1671.3 second delay
* To copy files we must have read permissions on the source file and write permissions on the target system.

Note

* for logging in with ssh keys we use -i flag

Syntax: ssh -i PathtoPrivatekey Username@host