SSH Port Forwarding

Project-Report

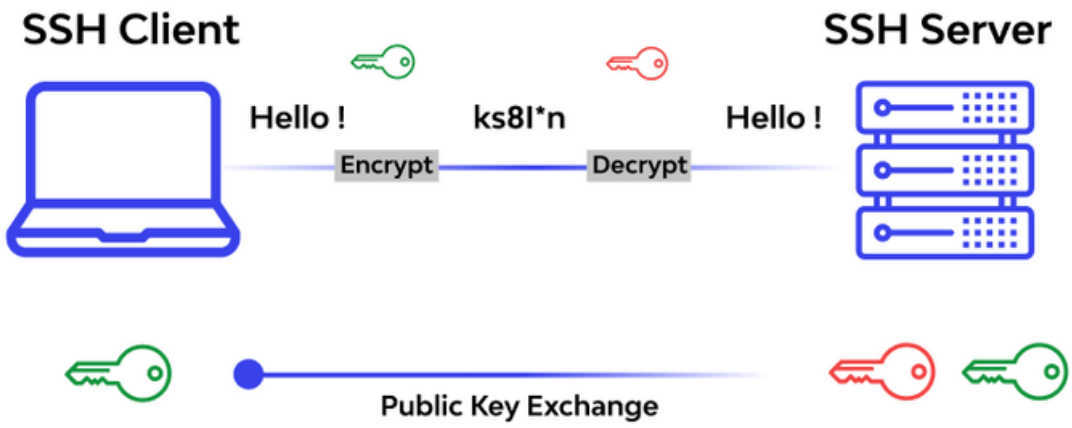
SSH :

Secure Shell (SSH) is a protocol that provides a cryptographically secure connection between two hosts over an unsecure network.

SSH can be used for remote command-line access, file transfer and tunneling.

In the Telnet protocol, anyone with a packet sniffer could eavesdrop on the contents of the packet, which becomes problematic when those packets contain personal or secret information.

SSH service was created as a secure replacement for the unencrypted Telnet and uses cryptographic techniques to ensure that all communication to and from the remote server happens in an encrypted manner. It provides a mechanism for authenticating a remote user, transferring inputs from the client to the host, and relaying the output back to the client



SSH Tunneling :

SSH (or SSH port forwarding) tunneling is a method for transporting arbitrary unencrypted data over a secure SSH connection. SSH tunnels enable connections to a local port to be transferred to a remote computer across a secure channel.

SSH tunneling allows us to access remote resources that we do not have access to because they are internal to that network. It is also used to allow others outside our network to have access to it. It is done through TCP tunneling.

SSH forwarding is useful for transporting network data of services that use an unencrypted protocol, such as VNC or FTP, accessing geo-restricted content, or bypassing intermediate firewalls. Basically, you can forward any TCP port and tunnel the traffic over a secure SSH connection .

There are three types of SSH port forwarding :

- Local Port Forwarding. - Forwards a connection from the client host to the SSH server host and then to the destination host port.

- Remote Port Forwarding. - Forwards a port from the server host to the client host and then to the destination host port.

- Dynamic Port Forwarding. - Creates a SOCKS proxy server that allows communication across a range of ports.

Prerequisites And Configuration :

Install the OpenSSH server package :

Usually, Kali Linux has an OpenSSH server running on it or installed on it . Or we can just use this command to install and enable the remote SSH open server in Kali Linux.

$ sudo apt install openssh-server

This package includes the SSH daemon (sshd), which is the service that runs on the server and listens for incoming connections from clients. The server package allows other machines to connect to it via SSH protocol.

Install the OpenSSH client package :

$ sudo apt install openssh-client

```
┌──(kali㉿kali)-[~]
└─$ sudo apt install openssh-client
[sudo] password for kali:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
openssh-client is already the newest version (1:9.0p1-1+b2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

This package includes tools such as ssh, scp, and sftp, which are used to connect to remote machines, transfer files securely, and tunnel other protocols over SSH. The client package allows you to connect to remote machines using the SSH protocol.

Install the Apache HTTP Server :

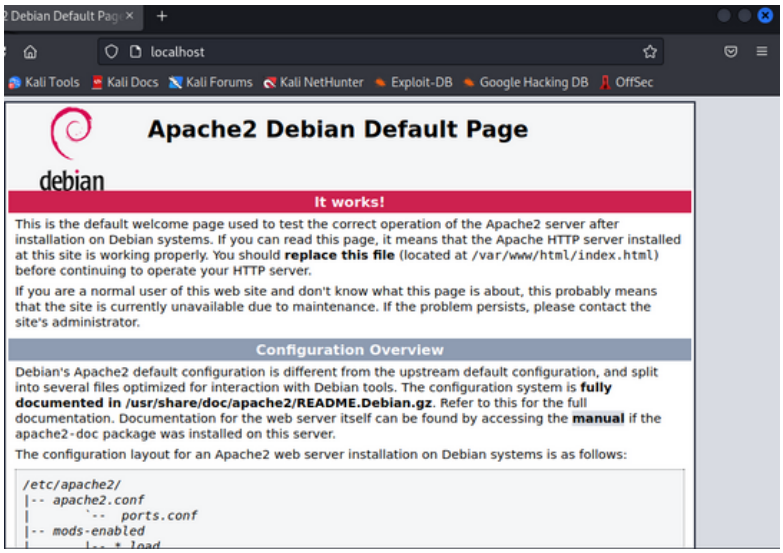
The Apache HTTP Server, also known simply as Apache, is a widely used open-source web server. It's used to serve and host websites, web applications, and APIs.

We will be utilizing the Apache2 web server to host multiple websites on a remote machine, then utilize SSH Tunneling to establish secure connections to these websites from different locations and scenarios.

$ sudo apt install apache2

```
┌──(kali㉿kali)-[~]
└─$ sudo apt install apache2
[sudo] password for kali:
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
apache2 is already the newest version (2.4.54-3).
apache2 set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

By typing in the browser "http://localhost" ("http://[server's IP address]") we can check if the server is running .



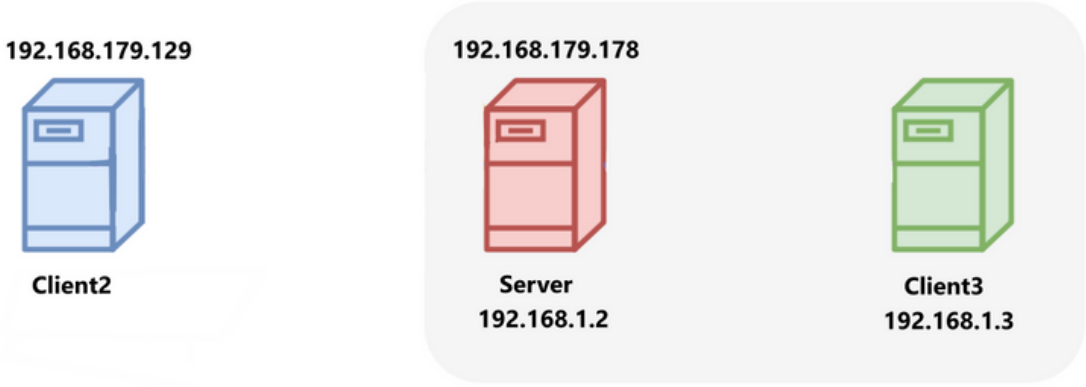
It display the default Apache page

we should first start apache2 service with the following command :

$ service apache2 start

What we need in our project :

In this project, we have three machines: (server 1, server 2 ,server 3)



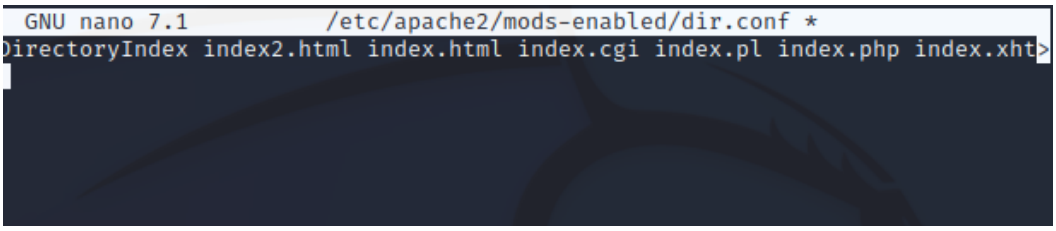
Server1(192.168.179.129) and Server2 (192.168.179.128) are connected on the same NAT network, Server2 (192.168.1.2) connected too with Server3 (192.168.1.3) on the same LAN network.

We install Apache2 on Server3 , And in order to change the default web page hosted by Apache2 in Server3 we create a new html file in "/var/www/html" directory using :

$ sudo nano /var/www/html/index2.html

But by default, Apache2 looks for a file named "index.html" in the document root directory to serve as the default page when navigates to "http://localhost" in the web browser. We can change the default file name that Apache2 looks for by editing the Apache2 configuration file

$ sudo nano /etc/apache2/mods-enabled/dir.conf



SSH Login Authentification :

SSH (Secure Shell) login authentication is a process of verifying the identity of a remote computer in order to allow secure access to a network or system.

SSH uses public-key cryptography to authenticate the remote computer and (optionally) to allow the remote computer to authenticate the user. This means that instead of using a password, a unique key pair (a public key and a private key) is generated on the client machine, and the public key is copied to the server.

Generate a new SSH key pair

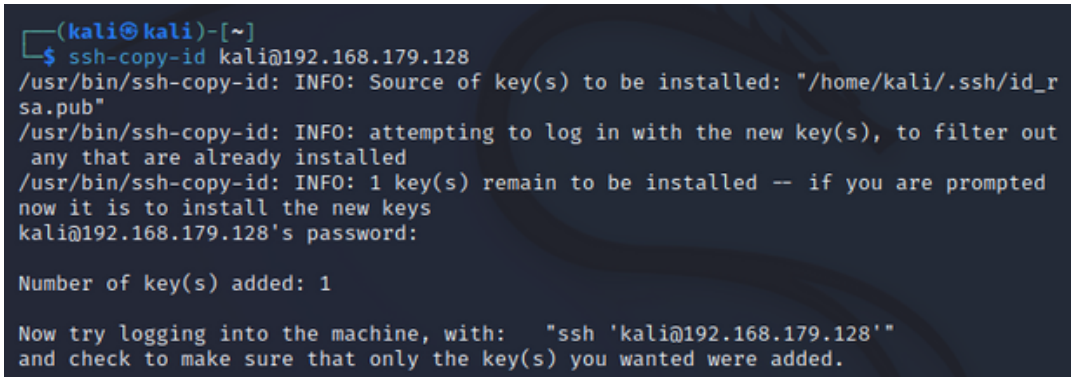
$ ssh-keygen -t rsa

Generate a new SSH key pair. The -t option specifies the type of key to create, in this case rsa

Now, we should Copy the public key to other machines using :

$ ssh-copy-id user@server_ip_address

This will add the public key to the authorized_keys file on the remote server



We can do this process for our machines, especially server1 with server2 and server2 with sever3

Now all our SSH connections connections between servers will be authenticated using the key pair.

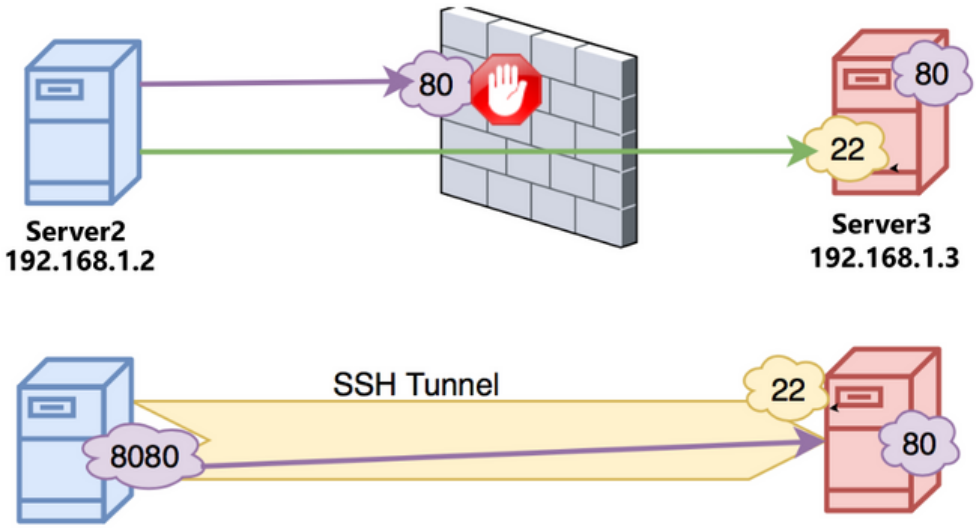
Local port forwarding :

Local port forwarding allows us access to remote content or resources to which we don't have access but a ssh server that you have access to can.

It is used to forward traffic on a port from of the client machine to the server machine, and next, that will be forwarded to the destination machine.
The client machine listens on a given port and tunnels the connection from that port to the particular port of the server machine in this type of forwarding. Here, the destination machine can be any remote server or another machine.

Scenario 1 : Firewall

In this first scenario we want to create an SSH local port forwarding between server2 and server3, allowing communication between the two servers while they are on the same network.



As server2 and server3 are on the same network and the Apache2 web server on server3 is configured to listen on port 80, then it would be possible to access the web server directly by typing "http://192.168.1.2" in a web browser on server2.

So we are going to modified the firewall rules on server3 to stop port 80, then it would not be possible to access the Apache2 web server directly. The traffic on port 80 will be blocked, and the web server will not be accessible.
To do this we use:

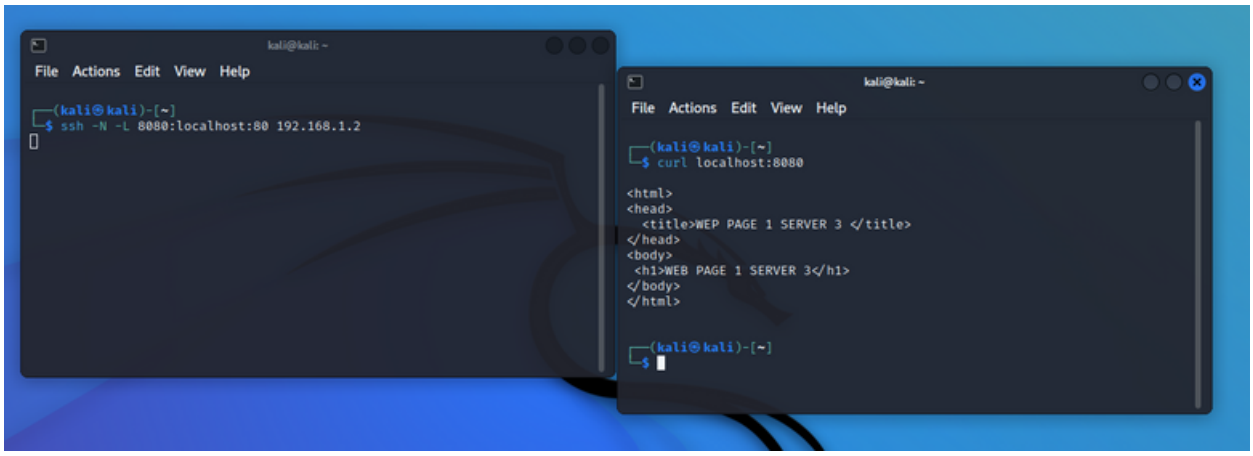
```
$ sudo iptables -I INPUT ! -i lo -p tcp --dport 80 -j DROP
```

This command is adding a new rule to the Systeme`firewall that drops all incoming TCP Traffic to port 80,

However, if SSH Tunneling is used to forward a local port on server2 to the port on server3 where the Apache2 web server is running, then it would be possible to access the web server.

To do this we use: **$ ssh -N -L 8080:localhost:80 192.168.1.2**

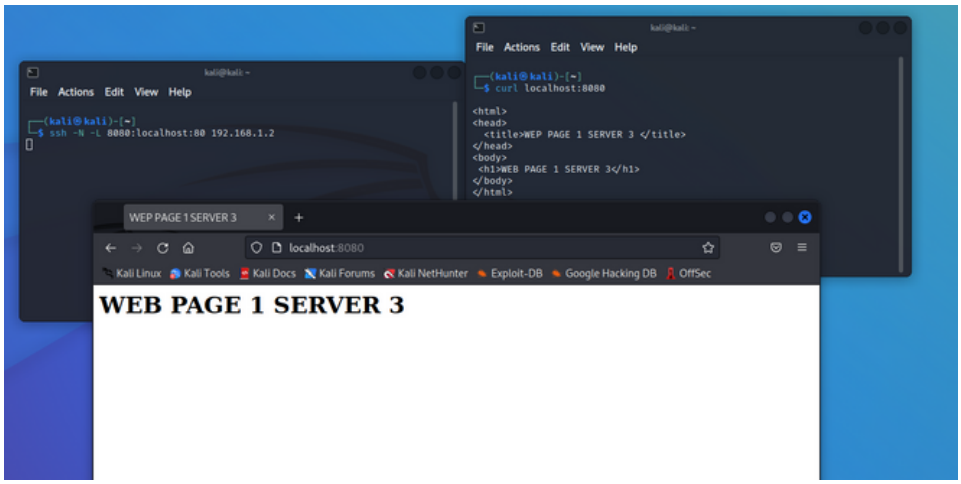
The "-N" flag is used to run the command in the background, it is useful when you only want to establish a port forwarding and don't have to run a command on the remote machine.



So this command forwards all traffic recieved on server2 port 8080 to the remote server (server3) on port 80.

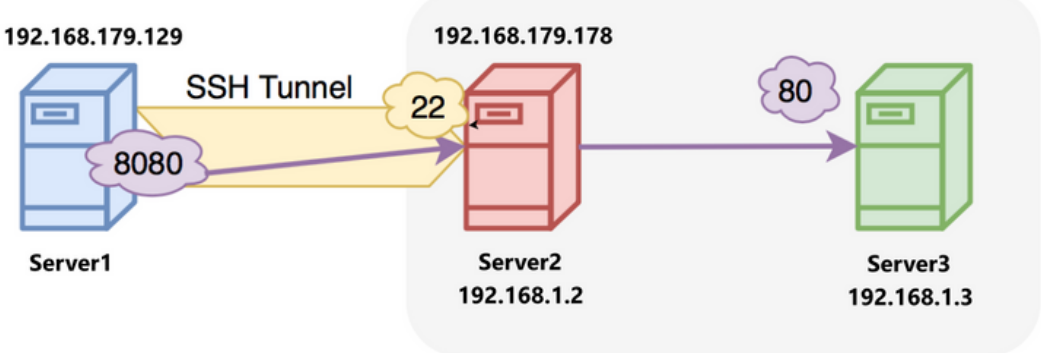
This means that we will be able to access the web server running on the remote server3 at 192.168.1.2 as if it were running on server2.

or we can type in the browser http://localhost:8080



Scenario 2 :

Server1 and server3 are not on the same network and can't communicate directly. Server2 acts as a bridge between them, it's connected to both networks and allows server1 to access services on server3 as if they were running on server1.

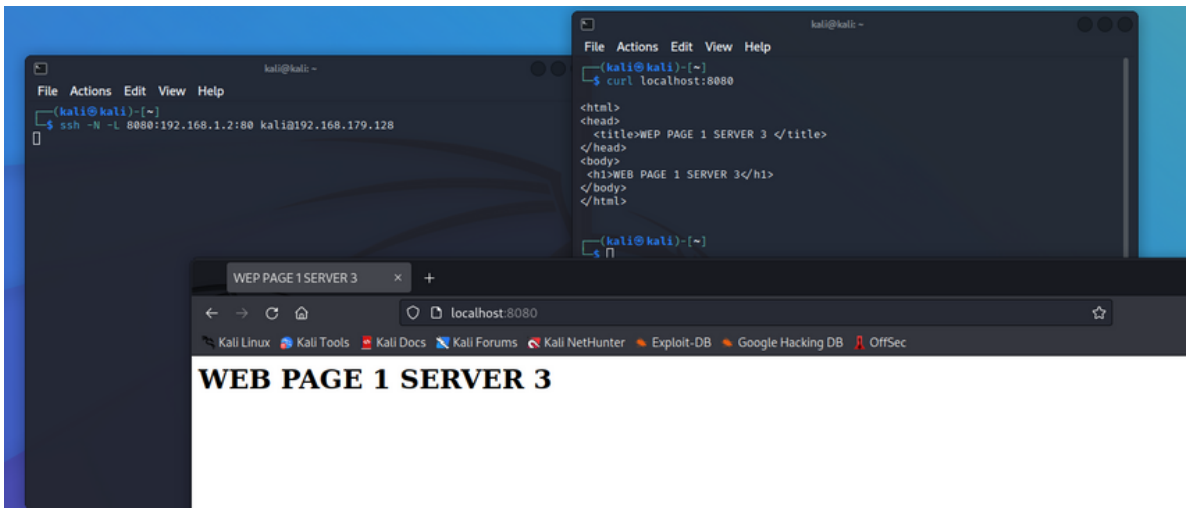


Apache2 web server is running on port 80 on server3, we can forward a local port, 8080 on server1 to port 80 on server3 through server2 by using the following command on server1:

$ ssh -N -L 8080:192.168.1.2:80 kali@192.168.179.128

This creates a local port forwarding, where all the traffic on port 8080 on server1 is forwarded through server2 to port 80 on server3.

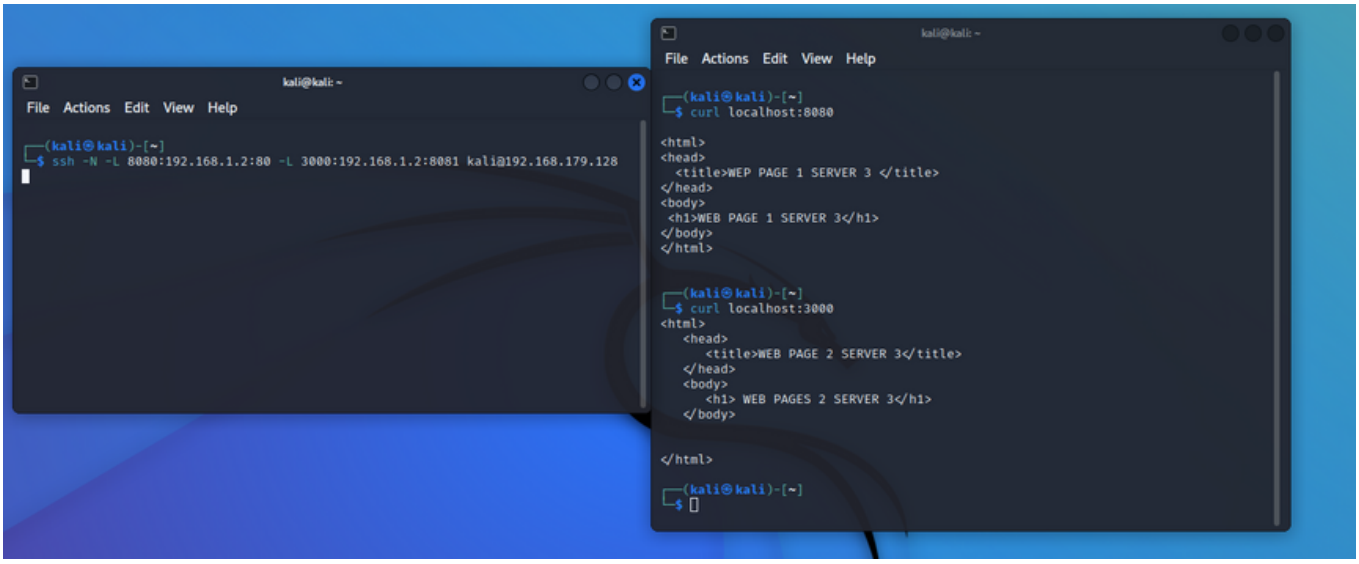
So we can access to web server on server3 as if they were running on server1.



Moreover, you can forward multiple sets of ports in a single SSH command , to do this we create other Web page hosted by the web server and we want each one to be accessible via a specific port. to do this we should add a Virtual Host on "/etc/apache2/sites-available/000-default.conf" and specifying the new port (8081).

We should also allow apache2 to listen on 8081 port.

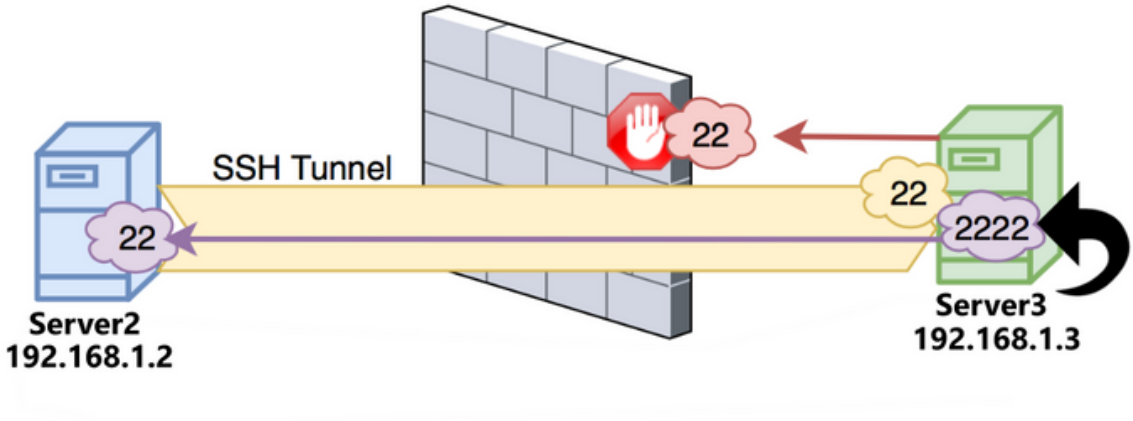
$ ssh -N -L 8080:192.168.1.2:80 -L 3000:192.168.1.2:8081 kali@192.168.179.128



Remote forwarding :

Remote port forwarding is the opposite of local port forwarding. It allows you to forward a port on the remote (ssh server) machine to a port on the local (ssh client) machine, which is then forwarded to a port on the destination machine.

In this forwarding type, the SSH server listens on a given port and tunnels any connection to that port to the specified port on the local SSH client.
It is mostly used to give access to an internal service to someone from the outside.



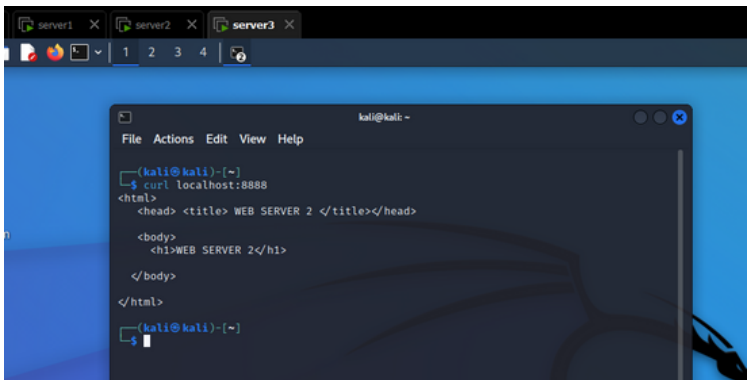
In this case we are creating a reverse ssh tunnel. Here we can initiate an ssh tunnel in one direction, then use that tunnel to create an ssh tunnel back the other way.

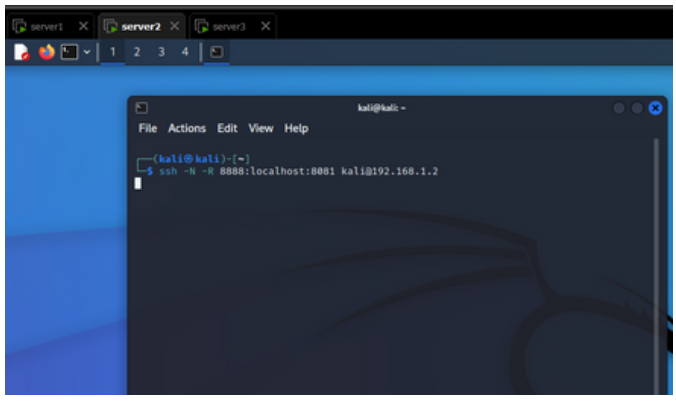
The server3 want to ssh to the server2. However, the firewall blocks this connection directly. Because the server2 can ssh to the server3, we can connect using that, and when the server3 wants to ssh back to the server2, it can ride along this previously established tunnel.

Server2 initiates ssh tunnel like this:

$ ssh -N -R 8888:localhost:8081kali@192.168.1.2

This opens port 8888 on the server3, which is then port forwarding that to port 8081on the server2. So if the server3 were to ssh to itself on port 8888 it would then reach the server2.





Now, we can get access to the web server on Server 2 from the outside (Server3)

Dynamic Port Forwarding :

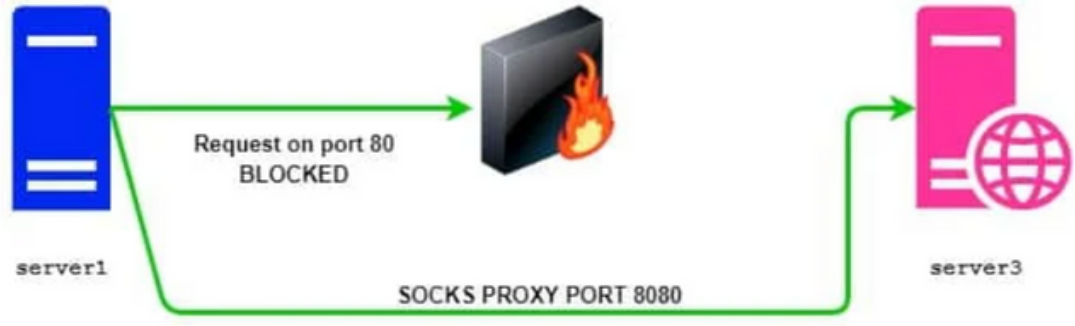
Dynamic forwarding is used to forward all traffic sent to a specific port on the local machine to the remote server through an encrypted SSH tunnel. This can be useful when you want to use the remote server as a proxy for all network traffic.

Dynamic port forwarding turns your SSH client into a SOCKS5 proxy server. SOCKS is an old but widely used protocol for programs to request outbound connections through a proxy server.

Dynamic port forwarding allows a communication not on a single port, but across a range of ports.

Without Dynamic Port Forwarding :

Wit Dynamic Port Forwarding :



The syntax for dynamic forwarding is as follows :

$ ssh -D local_port user@remote_server

In our example the firewall block connection to port 80 on Server3, so we are going to use the following command on server2

$ ssh -N -D 8080 kali@192.168.1.3

providing the local port to be forwarded is enough because the destination is determined dynamically, and can be different for each connection. also the server towards which we want to create the secure tunnel can be different each time

Verify SSH Tunnel setup :

Now we will try to use SOCKS proxy to connect to the apache server from server3:80 using server1:8080

$ curl --proxy socks5h://localhost:8080 192.168.1.2:80

The curl connection was successful with SOCKS proxy

Security Considerations :

When using SSH tunneling, it is important to take steps to ensure the security of your network. Here are some best practices to follow:

- Implement strong authentication methods, such as using complex passwords or public key authentication, to protect against unauthorized access to the remote server.
- Limit access to the remote server by restricting the number of accounts that have permission to connect and monitoring access to private keys.
- Utilize network security measures like firewalls to protect both the remote server and local network.
- Regularly monitor the remote server and local network for suspicious activity and take immediate action if any is detected.
- Keep all software and operating systems up-to-date with the latest security patches to protect against known vulnerabilities.

By following these guidelines, wecan help secure your SSH tunnel and protect against potential threats.

Conclusion :

In conclusion, SSH port forwarding is a powerful tool that enables secure communication between a local machine and a remote server, even when the remote server is located behind a firewall.

With three different types of port forwarding available, such as local forwarding, remote forwarding and dynamic forwarding, it is important to understand their specific uses and how they can be applied to access remote resources and services.

By having a clear understanding of how to use these different types of port forwarding, we can utilize SSH more effectively and securely access the resources you need.