SSH-TUNNEL LAB

1 Lab Overview

You should read through this document before you start the lab.

The goal of this very simple lab is to illustrate the use of tunneling to access remote systems via a chain of *ssh* commands. See the network configuration in Figure 1.

The scenario of the lab is that an attacker has done appropriate reconnaissance to find out the IP addresses, host names and at least one *userid* on each host, on a path from **base** to **hostd**. You might think the the attacker could set up a direct *ssh* connection to **hostd** but the gateways along the way, except for *gw4* don't know about **hostd**'s network and hence wouldn't be able to route the packets. Hence the need for the tunnel.

The attacker found out that **the *userid* for all the hosts is *ubuntu*. The password for the *userid ubuntu* is *ubuntu* on all the hosts.**

The user will use *ssh* to create a tunnel to **hostd**. Unfortunately, the routing table in **hosta** only knows how to route packets to **base** and **hostb**. The situation is similar for hosts **hostc** and **hostd**. Hence **base** cannot connect directly to **hostd**. The goal of the lab is to use the tunnel to copy a file from **hostd** to **base**.

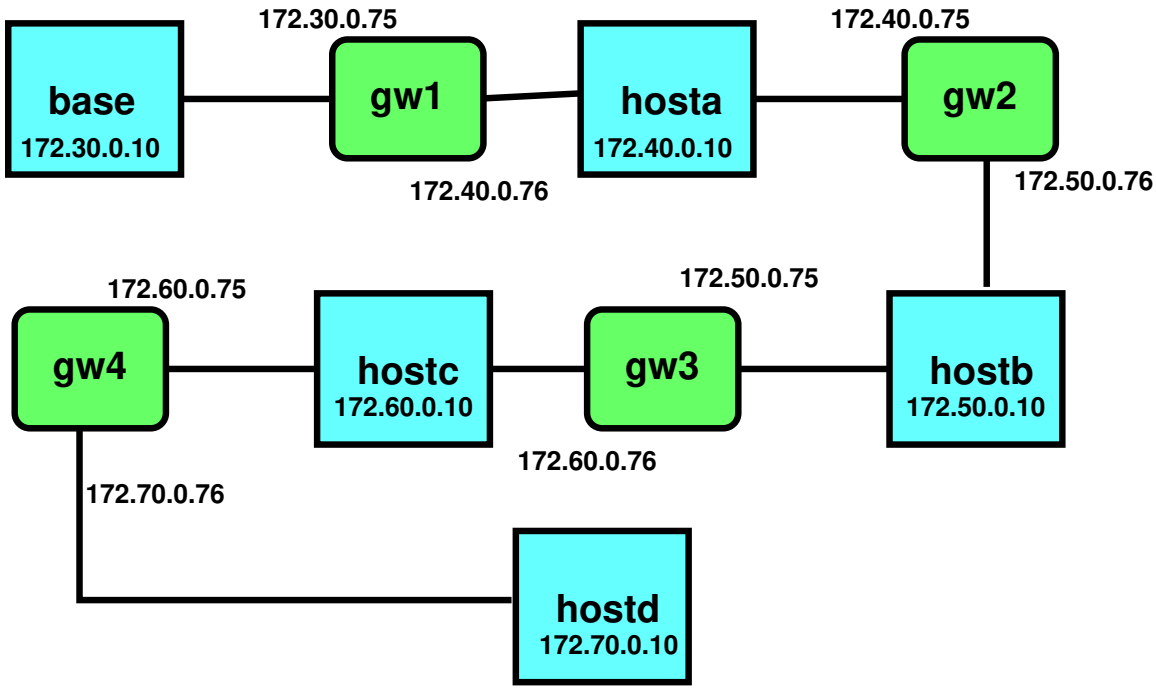

Figure 1: Lab Network

The addresses on the routers are as described in the figure. You won't be dealing with those addresses.

2 Lab Tasks

2.1 Initial setup

The lab is started from the Labtainer working directory on your host, e.g., a Linux VM with Labtainer installed. From there, issue the command:

Command 1:

```
labtainer ssh-tunnel
```

The resulting terminal is connected to the **base** computer. You do all your work inside this terminal.

We will create the tunnel by creating *ssh* connections, step by step from one host to the next. We will do this in a way, so that we can issue commands to **hostd** directly from the **base**. When **hostd** investigates its connections as part of its auditing, it will only see an *ssh* connection to **hostc**, IP address 172.60.0.10, which would probably be expected.

2.2 Trying to Access the hosts

Just to check, you should try to ping hosts **hosta** . . . **hostd** and the gateways, to determine which devices are visible from the **base** host. You could use IP addresses or host names to test.

2.3 Testing the Environment

You should probably look at the documentation of *ssh* using the command:

Command 2:

```
man ssh
```

The documentation is extensive. There are a lot of options. You should concentrate on the `-f`, `-4` and the `-L` options. These are the ones we will use. If you are familiar with *ssh*, you may not have seen the `-4` option. This version of *ssh* support IPV6 and will attempt to use that unless the `-4` (use IPV4) is specified. We are not using IPV6 so we need the `-4`.

From above you should have determined that **base** can access **hosta**. Our reconnaissance has determined that **hosta** has an *ssh* server running so we will set up an *ssh* connection to **hosta**, just to test things out. Try the following commands:

Command 3:

```
ssh -4 hosta
```

You will need to "allow" the connection and enter a user name and password (see above) to log into **hosta**.

You should then issue the following command on **hosta** to get a sense of its environment.

Command 4:

```
route -n
ping hostb
netstat -n | head
ps -ax | grep ssh
cat /etc/hosts
```

The *route* command will show you the routes the host knows. The *netstat* command will show you the current connections and port numbers. The *ps* command will show you the *ssh* connections (via the filter *grep*). You will need to stop the *ping* with `<CTRL-C>`. You could try pinging other hosts either by name or by IP address to see what is connected. When you are done, you can "leave" **hostb** via:

Command 5:

```
exit
```

which terminates the shell on **hostb**. You should see from the prompt that you are no typing at **base**.

2.4 Creating First Step for Tunnel

To set up the first leg of the tunnel, you should issue the command:

Command 6:

```
ssh -4 -f -L 1111:hostb:22 hosta -N
```

Here are the descriptions and rational for the arguments:

`-4` As mentioned above, we are not using IPV6 so we tell *ssh* that we are using IPV4.

`-f` This argument says that we are setting up an *ssh* connection that will **NOT** connected to the terminal. It will run in background mode.

`hosta` We skipped to nearly the end of the command because this *ssh* actually directly connects to **hosta**.

`-L 1111:hostb:22` This argument is the "trick" in creating the first (and subsequent) legs of the tunnel. In this case, it says:

> On **hosta**, the destination of the *ssh* command, create an *ssh* connection from **hosta** to port 22 on **hostb**. Assign the TCP port `1111` on **base** to reference this connection.

The result is that when **base** makes an *ssh* connection to port `1111` it is actually communicating with **hostb**, transparently through **hosta**.[1]

`-N` Do not execute a command on the remote host. Normally *ssh* expects a command as the last set of arguments. the `-N` arguments tells *ssh* that this is a tunnel.

To tests the result of the command, the creation of the first leg of the tunnel, you should try the command:

Command 7:

```
ssh -4 -p 1111 localhost
```

The `-p 1111` argument tells *ssh* that it should connect to port `1111` and the `localhost` argument says that that port is on the `localhost`, in this case, **base**.

Again, what is happening is that you have *ssh*ed to the port `1111` on the local host. From the previous command the port is connected through **hosta** to **hostb**.

[1] For those of you with some *Unix* background, we access this tunnel on localhost, IP address `127.0.0.1` and port `1111`.

You should figure out which host your terminal is now connected to. You can tell by the *prompt*. Issue the commands we listed above and note which hosts are associated with the *ssh* connection. You can tell that in the *netstat* command, one of the ports is `22`.

When you are done experimenting, you can type

Command 8:

```
exit
```

Your prompt should now indicate that you are back typing to the **base** host.

2.5 Next Step in the Tunnel

Now type the command:

Command 9:

```
ssh -4 -f -p 1111 -L 2222:hostc:22 localhost -N
```

As above, you should

The port `1111` on the local host (hostname: *localhost*) is connected (from the First Step Command) to the *ssh* port `22` on **hostb**. So typing an *ssh* command on the local host is in effect, telling **hostb** to make that connection. Since the *ssh* command is going through the tunnel from **base** to **hostb**, the `2222` port number will be a port on **base**.

As a result, any *ssh* command on base using the port number `2222` will be directed at hostc.

Now, using port `2222`, *ssh* to **hostc** and experiment as above.

2.6 Third Step in the Tunnel

It is your job to now figure out and execute the *ssh* command that connects port `3333` on localhost (**base**) to the *ssh* port on **hostd**. You should mimic the command in 9.

Once you have made the tunnel connection to **hostd** you should try logging into **hostd** and explore as you did before. Note that **hostd** will not have a direct connection to either **base** or **hostb**.

2.7 "Proving" You Made the Tunnel

You now need to provide to the automagic lab grader that you have, in fact make the tunnel. To do that we have provided a file, **copyfile.txt** on **hostd**. You should copy this file over the tunnel to **base**. To do that you can use the *scp* command, copy a file over an *ssh* tunnel. Here is the command:

Command 10:

```
scp -P 3333 localhost:copyfile.txt /home/ubuntu
```

Note that the port is identified with a capital **`P`**. The `/home/ubuntu` is the home directory where you should have seen the `copyfile.txt` file when you were looking in **hostd** above.

When using *scp*, normally one would say something like `remotehostname:filename` to identify a file on the remote host to copy. We are, in fact doing the same thing here. Port `3333` on localhost, in this case, refers to **hostd**, so `-P 3333 localhost:copyfile.txt` refers to `copyfile.txt` in the home directory of the user *ubuntu* on **hostd**.

You can check that the file was copied using the command:

Command 11:

```
ls -l
```

It should appear in the listing. If it is there, you can look at its contents with the command:

Command 12:

```
less copyfile.txt
```

which will write the contents of the not very interesting file to *stdout*, the terminal window.

2.8 Stopping the Lab

Be sure to stop the lab with the command

Command 13:

```
stoplab
```

in the same terminal window you issued the `labtainer` command.

We need this, since this will save much of your output, and the automagic grader will be able to determine whether you were successful or not.

3 Final Thoughts

We have used port `22` as our final destination port on **hostd**. The reason is that we want to copy files from **hostd** to **base**. If we wanted to access a web server on **hostd** we could have used a a destination port of say `443` on the final stage of our tunnel instead of `22`. To access the web server we would use the *url*:

Command 14:

```
https://localhost:3333
```

in our local web browser. The same idea would work for other services. We only need *ssh* for the intermediate legs of the tunnel.