

ITS Assignment 4

David hbd126, Rasmus dht579

8. October 2023

Task 2.A: Protecting the Route

Start by setting up the docker using:

```
docker-compose build
```

Then you start the docker using

```
docker-compose up
```

Now you have a fully set up dock. We can then get the ID of the containers using this command:

```
docker ps
```

This gives us the following addresses:

```
30c5a47e5f4b  host1-192.168.60.5
81a659309078  host3-192.168.60.7
899c751c5b81  host2-192.168.60.6
be9b41db15e4  hostA-10.9.0.5
f5f44c584472  seed-router
```

We can now use the command `docksh ID#` to start the shell on that container. If we want to use 81a659309078 host3-192.168.60.7 we write the following: `docksh 81`

We are now root in that shell and can do our tests from there. That is the setup for our tests so we have a replicable setup if we need to redo tests later.

At this point we have no rules for connecting to our network.

To add rules, we would then **docksh** into our router and use 'iptables' commands to modify it.

With the given commands from the assignment, we are blocking access from the router except for pings. Below is an example of what it looks like to connect using telnet:

```

root@1c48277b28b2:/# telnet 192.168.60.11
Trying 192.168.60.11...
Connected to 192.168.60.11.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
dda0af3fbdcdb login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@dda0af3fbdcdb:~$ █

```

Figure 1: Successful telnet connection

After connection to the router using the **docksh ID#** command, we can then add the given rules.

- 1) `iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT`
- 2) `iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT`
- 3) `iptables -P OUTPUT DROP`
- 4) `iptables -P INPUT DROP`

Trying to connect using telnet after the router has setup its rules gives us the following result:

```

root@1c48277b28b2:/# telnet 192.168.60.11
Trying 192.168.60.11...

```

Figure 2: Telnet trying to no avail

The telnet connection stalls on **'Trying ...** until it times out or we force close it.

Analysing the rules we see that rule 1 and 2, allow incoming icmp echo-requests

and outgoing icmp echo-replies. echo-request and echo-reply are the types of packets used by ping requests, so this allows us to continue using pings. We need to **ACCEPT** both incoming and outgoing, so that the pings function properly. Rule 3 and 4 then **DROP** all other input and output, meaning that our tries at connecting using **telnet** are dropped, which aligns with our observations.

Before moving on to the next task we are to clean our container of rules using:

```
docker restart <Container ID>
```

Task 2.B Protecting the Internal Network

We start by entering our router, then we enter the following 3 commands:

```
iptables -A FORWARD -p icmp --icmp-type echo-request -s
192.168.60.0/24 -d 10.9.0.0/24 -j ACCEPT
```

```
iptables -A FORWARD -p icmp --icmp-type echo-reply -s
10.9.0.0/24 -d 192.168.60.0/24 -j ACCEPT
```

```
iptables -P FORWARD DROP
```

These allow external hosts to ping the router but not the internal hosts. For instance ifwetry to ping HostA using Host1, then it just hangs. But ifweenter the command ping router, then Host1 can do that. So that fulfills the first two conditions. The next is that internal hosts can ping outside hosts, so we try to ping google.com from HostA, which it does fine. Next is the final line which is our default that we just drop everything else.

Here we see that we can ping the router, but not the internal hosts:

```
root@30c5a47e5f4b:/# ping 10.9.0.0
PING 10.9.0.0 (10.9.0.0) 56(84) bytes of data.
^C
--- 10.9.0.0 ping statistics ---
16 packets transmitted, 0 received, 100% packet loss, time 15
352ms

root@30c5a47e5f4b:/# ping router
PING router (192.168.60.11) 56(84) bytes of data.
64 bytes from seed-router.net-192.168.60.0 (192.168.60.11): i
cmp seq=1 ttl=64 time=0.075 ms
```

Here we see that the internal hosts can ping google.com:

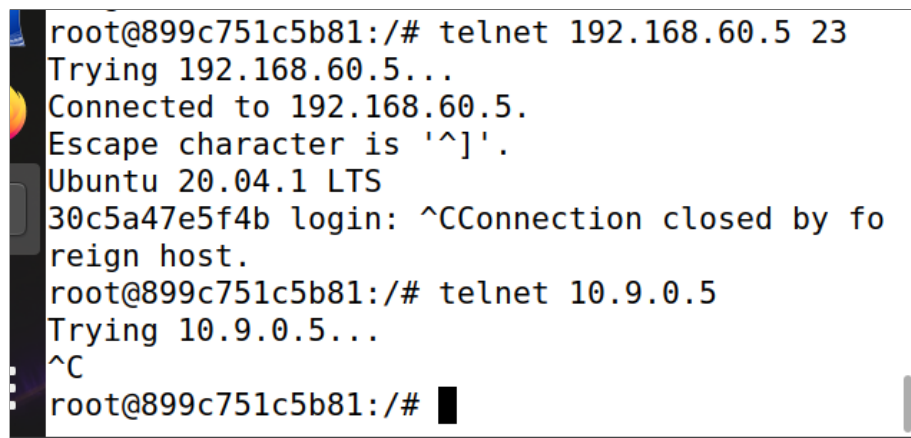
```
pot@be9b41db15e4:/# ping google.com
PING google.com (142.251.36.14) 56(84) bytes of data.
4 bytes from ams15s44-in-f14.1e100.net (142.251.36.14): icmp_seq
1 ttl=118 time=24.2 ms
4 bytes from ams15s44-in-f14.1e100.net (142.251.36.14): icmp_seq
```

Task 2.C: Protecting Internal Servers

In this section we are asked to use TCP, which should be relatively straight forward as the rules are more or less the same as the previous task, so we can take our rules from there and modify them to use TCP instead of ICMP:

```
iptables -A FORWARD -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT
iptables -A FORWARD -p tcp -s 192.168.60.0/24 -d 192.168.60.0/24 -j ACCEPT
iptables -P FORWARD DROP
```

When we implement this and test it using Host1 as an internal server and HostA as an external server then we see the following results:



```
root@899c751c5b81:/# telnet 192.168.60.5 23
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
30c5a47e5f4b login: ^CConnection closed by fo
reign host.
root@899c751c5b81:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@899c751c5b81:/#
```

Figure 3: Internal behaviour

From this we can see that we are not able to access external telnet servers from an internal server (Host1), but we can reach the internal ones. We were also able to telnet into the router, which should be possible from both external and internal servers.

```

root@be9b41db15e4:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
root@be9b41db15e4:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
be9b41db15e4 login: ^CConnection closed by fo
reign host.
root@be9b41db15e4:/# telnet router
Trying 10.9.0.11...
Connected to router.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f5f44c584472 login: ^CConnection closed by fo

```

Figure 4: External behaviour

Here we see that we are not able to connect to the internal server (Host1) from an external server (HostA) but we are able to connect to the router and other external servers. The problem here is that we are not able to connect to the internal host 192.168.60.5 which is part of the assignment, so our firewall is not working fully as expected.

Task 3

Task 3.A Experiment with the Connection Tracking

First we clean the dock from last time with the instructions given in task 2.A, then we ping Host1 from HostA to see if it works, which it does, we then cancel that ping command. So now we can start experimenting with the Connecting Tracking. We write the following command to get the initial connection tracking stats:

```
conntrack -L
```

This gives us the following output:

```
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
```

What this means is that we currently have 0 flow entries, but if we ping Host1 from HostA again and keep the connection open, and enter the `conntrack -l` command again, then we will have one flow entry, which means that we are tracking the connection between HostA and Host1.

Now we experiment with UDP, so we start by setting up a UDP server on Host1 with the following command:

```
nc -lu 9090
```

Then on HostA we enter the following command:

```
nc -u 192.168.60.5 9090
```

This now keeps an open UDP connection between HostA and Host1 where we can type messages which are then sent between the two hosts. While the connection is open we can use the `conntrack -L` command and see that one connection is open. If we close the connection on HostA then it drops to zero as there is no longer any other hosts communicating with the UDP server on Host1.

When doing it with TCP we enter the following commands on Host1:

```
nc -l 9090
```

Then on HostA we enter the following command:

```
nc 192.168.60.5 9090
```

When we now check on the router using `conntrack -L` then we get one open connection. If we now drop the HostA connection then there is still a connection being tracked. This is due to the TCP protocol trying to reestablish connection and keep the communication between the two hosts secure.

Task 3.B: Setting Up a Stateful Firewall

In this task we are to set up rules, which maintain a stateful firewall. This means that using connection tracking, we can monitor the states of connections, and manage the flow of data through these states. The task is to rewrite our rules from 2.C using this newfound statefulness.

We take a look at our rules from 2.C:

- 1) `iptables -A FORWARD -p tcp -d 192.168.60.5 --dport 23 -j ACCEPT`
- 2) `iptables -A FORWARD -p tcp -s 192.168.60.0/24 -d 192.168.60.0/24 -j ACCEPT`
- 3) `iptables -P FORWARD DROP`

Our first rule accepts connections with destination 192.168.60.6 on port 23.
Our second rule accepts all internal connections.
And the third rule drops all other connections.
Using connection tracking, we are to also allow internal hosts to access external hosts.
For this purpose we have written the following rules:

- a) `iptables -A FORWARD -p tcp -d 192.168.60.5 --dport 23 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT`
- b) `iptables -A FORWARD -p tcp -s 192.168.60.0/24 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT`
- c) `iptables -P FORWARD DROP`

Going through the rules we see that rule a, allows new and established connections with 192.168.60.5 on port 5 to go through. This aligns with our earlier rule 1. For rule b, we see that instead of limiting our internal connections to other internal, we have now allowed all connections with an internal source, both new and established. And rule c, just like before, drops all other connections.

```
root@30c5a47e5f4b:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
^C
--- 8.8.8.8 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5137ms

root@30c5a47e5f4b:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@30c5a47e5f4b:/# telnet 192.168.60.6
Trying 192.168.60.6...
Connected to 192.168.60.6.
```

Figure 5: Internal connection behaviour


```
root@be9b41db15e4:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
root@be9b41db15e4:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
be9b41db15e4 login: ^CConnection closed by fo
reign host.
root@be9b41db15e4:/# telnet router
Trying 10.9.0.11...
Connected to router.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
f5f44c584472 login: ^CConnection closed by fo
```

Figure 6: External connection behaviour

Above we see the rules in action on our router and hosts. In the first picture we see that the internal networks are able to communicate with each other but not external networks. Which is the same issue we had in task 2.C, so it seems there is something in our instructions which are fundamentally wrong.

In the second screenshot we see that the external connections are unable to communicate with internal networks, which is intended, but they are supposed to be able to communicate with host1 on port 23, but that does not seem to work. This is again a reoccurring issue from task 2.C which we have been unable to solve.

We are also tasked with discussing possible ways to achieve this without using the conntrack module. Since the subject is a stateful firewall, we believe the goal would be to achieve such a thing without the connection tracking. With limited knowledge of the capabilities of iptables and its tools, we believe it would be possible to set up chains and rules corresponding to the ctstates used in conntrack. A rough idea is to use the state module, which as far as we can tell differs from conntrack in that it does not track connections, but can still give states. Though we are not sure how this is achieved. Using states we would then associate chains with the states and then have some chains lead to accepts, and others lead to drops. The main drawback of this approach would probably be

the complexity of the setup, since implementing a mimicry of a newer solution using deprecated functions is usually not a great idea and a waste of resources. The solution would probably also buckle under strain as it would be unable to handle a larger amount and more complex connection pathways. We are not sure which disadvantages conntrack would have that differ from our proposed alternative, since we imagine the potential overhead would be the same for either solution. A guess would be the required resources for conntrack since it has to do more things than our simple solution, but it would also be more secure.

Short answer questions:

What is a VPN?

A VPN (Virtual Private Network) is a way to increase privacy and security on public connections. This is done by encrypting your connection and routing it through a server in another location. This provides several benefits, one being that it allows users to bypass geographical restrictions on content or allows companies to let their employees into the network, from outside, in a secure way.

Can a host firewall block malware from communicating from an infected computer back to its controller on the internet?

Taking the seed-lab exercises as an example, we see that we can limit what internal hosts are able to connect to in the outside world. In the same way, a host firewall would be able to block infected internal computers from their controllers on the internet if the rules are set up in such a way. This is of course a very simple way of going about it, and there exists more advanced options that allows host firewalls to inspect the packages and the behaviour of traffic to counter malicious behaviour. We can therefore conclude that the answer is yes, given that we in seed-lab demonstrate this directly.

Is an IPSec connection to your company network secure even if your home router's TCP/IP implementation has a buffer overflow vulnerability?

Security systems are only as strong as their weakest link. This means that if we have a very weak link in our home router, then our entire system is weak. In this case we are dealing with an IPSec connection, which is an encryption tunnel, where our packages are encrypted so they are unreadable if intercepted. But if a malicious person has access to our router, they can perhaps intercept our packages before they are encrypted which negates our entire security system.

What is the difference between IPSec tunnel and transport mode?

To give a short answer: Tunnel is used for connecting two networks, fx a home router or office A's router to office B's router. Whereas transport mode is from one device to another, fx from an employees device to a work server.

IPSec tunnel is primarily used to create VPNs between networks, as it encapsulates the original IP packet in a new IP packet. This is done by giving the packet a new header and treating the old header as payload. A common use case, as stated in the short answer, is creating VPNs for network to network or

site to site communication.

Transport mode differs in that it only encrypts the payload part of the IP packet. This is therefore more commonly used in host to host communication.