Luke Evans

February 5, 2021

Lab 2: Linux Firewall Exploration Lab

*Task 1: Using Firewall*

In this task Machine A has IP address 10.0.2.15 and Machine B has IP address 10.0.2.4.

The first task is to prevent Machine A from telnetting to Machine B:

Text, letter

Description automatically generated

Before we implement any firewall rules on Machine A, we can see that we are able to telnet out to machine B. In the following screenshots we will set a firewall to prevent this from occurring.

Graphical user interface, text, application, chat or text message

Description automatically generated

After using ufw to deny telnetting on port 23 from 10.0.2.15 to 10.0.2.4 we are unable to telnet from machine A from machine B. This is shown by the trying message however a connection will never be established. In the previous screenshot, we were able to telnet without any issue when our firewall was not running.

Next, we will prevent B from being able to telnet to machine A by setting a firewall rule on machine A.

Before adding a rule on machine A we are able to telnet from machine B:

Text

Description automatically generated

Next, we add our rule to Machine A preventing machine B from telnetting to machine A:

Text, letter

Description automatically generated

Now, when we try to telnet from Machine B to Machine A, our request will be blocked:

Graphical user interface

Description automatically generated with medium confidence

Finally, we will prevent Machine A from visiting [www.mastercard.com](http://www.mastercard.com):

A screenshot of a computer screen

Description automatically generated with medium confidence

This shows that before adding the rule to machine A’s firewall we are able to visit [www.mastercard.com](http://www.mastercard.com). Next, we will need to determine the websites IP address. We can do this with the host command:

Text, chat or text message

Description automatically generated

Now that we know the IP address, we will set our firewall rule. However since [www.mastercard.com](http://www.mastercard.com) has TLS enabled we need to block port 443 instead of port 80. We will block the IP address 104.96.108.184:

Text, letter

Description automatically generated

Now we try to navigate to [www.mastercard.com](http://www.mastercard.com) in a private firefox window and see that we are no longer able to now that our rule has been applied:

Graphical user interface, text

Description automatically generated

We first determined the IP address by using the host command. Then we set our rule to deny traffic on port 443 since the website has TLS enabled (https). If it we http, we would have to set our rule to port 80. As shown in the screenshot above. We can no longer access [www.mastercard.com](http://www.mastercard.com) it just stays on the screen above and eventually times out.

*Task 2: Implementing a Simple Firewall*

First we write our setUpFilter and removeFilter functions:

Text

Description automatically generated

The code above sets up our ho and prints out debugging messages to the kernel log.

Next we need to write our code to perform our packet filtering. This will be done in the dropPacketsTo and dropPacketsFrom functions shown in the screenshots below:

Text

Description automatically generated

The code above is used to drop packets and print out debugging information show the IP its dropping to/from.

The previous code was pretty standard in setting up the netfilter hook. The code below is our custom code that will implement five firewall packet filtering rules:

Text

Description automatically generated

The filter code above takes the packet from the buffer skb and extracts the ip header from the start of the buffer and then extracts the tcp header by moving the pointer from the start of the buffer to the ip header length \* 4; essentially moving the pointer to the start of the data portion of the ip packet which in this case is the tcp header.

Next, 5 filters are created:

1. Block incoming telnet requests, destination port 23. Drop packet.
2. Block outgoing telnet requests, source port 23. Drop packet.
3. Block all HTTP packets, port 80. Drop packet.
4. Block all HTTPS packets, port 443. Drop packet.
5. Block ICMP protocol, we could also optionally use port 7. Drop packet.

If the packets do not meet any of the above criteria, we will return packet accepted and allow the packets to pass through.

In order for our code to work we must leverage the loadable kernel model functionality that is supported by Linux. To do this we create the following Makefile:

Text

Description automatically generated

Next, we compile our code using the make command (some of the warnings are cut off):

Text, letter

Description automatically generated

…

Text

Description automatically generated

Now that the modules have been created, we must insert the LKM using the insmod command. Then we will attempt to telnet out to another machine. We expect the packets to be dropped and the kernel to log this for us:

Text

Description automatically generated

The log messages show that the packets are dropped going to 10.0.2.4:

Text

Description automatically generated with low confidence

Above we can see that the telnet command from 10.0.2.15 to 10.0.2.4 is not allowed and the logs view by running the dmesg command show the packets being dropped.

Next we will try to telnet to 10.0.2.15 from 10.0.2.4. Again we expect this to be unsuccessful and should see the packets FROM 10.0.2.4 logged.

Text

Description automatically generated

Text

Description automatically generated

Our firewall rule blocking incoming telnet request successfully dropped the packets.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated with low confidence

Visiting websites over http are blocked. This is show by our inability to visit [www.cis.syr.edu](http://www.cis.syr.edu). This is blocked by are rule not allowing outgoing traffic to port 80.

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with medium confidence

Visiting websites over https are blocked. This is show by our inability to visit www.mastercard.com. This is blocked by are rule not allowing outgoing traffic to port 443.

Text

Description automatically generated

Text

Description automatically generated

Next, we show that ICMP protocol is blocked. We can demonstrate this by using the ping command to google.com which we know is up and running. We see 100% packet loss due to our rule to drop all ICMP packets.

*Task 3: Evading Egress Filtering*

The rules that we created in our last exercise block all incoming and outgoing telnet requests as well as blocking all http and https traffic. See previous exercise for code snippets.

We can show that telnet is blocked by trying to telnet to 10.0.2.4 from 10.0.2.15:

Text

Description automatically generated with medium confidence

To evade the egress filter settings that exist, we will use an ssh tunnel. First we will create a ssh tunnel via port 8000 on our localhost to 10.0.2.4 and port 23 for telnet traffic. This is shown in the screenshot below:

Text

Description automatically generated

Text, letter

Description automatically generated

We can see that our telnet request is successful when using our ssh tunnel since our firewall on machine A is allowed to ssh out. We can then forward our telnet requests through the tunnel to a machine that does not have the same telnet restrictions.

Next, we show that we are unable to access facebook.com:

Graphical user interface, text, application, email

Description automatically generated

Now we will create an ssh tunnel to by pass the egress filter on facebook.com:

Text

Description automatically generated

Now that our tunnel is created, we can connect to port 9000 and proxy the request to 10.0.2.4 which is not restricted by our firewall rules.

Graphical user interface, application

Description automatically generated

We set our firefox browser to proxy our request to our localhost on port 9000 which will use the tunnel that we previously created. Now we are able to access facebook.com.

Graphical user interface, text, application

Description automatically generated

We then break the tunnel and refresh and we can see the proxy server is refusing connections:

Graphical user interface, text, application, email

Description automatically generated

We re-establish the tunnel:

Text

Description automatically generated

When we refresh we can once again access facebook since the traffic is being routed through our ssh tunnel to 10.0.2.4 which does not have the same firewall restrictions.

Graphical user interface, text, application

Description automatically generated

This shows that we can evade egress firewall filtering by using ssh tunnels to tunnel traffic to a different machine that does not have the same egss firewall rules. This allows us to circumvent the firewall and perform restricted actions.

We can see the packets flowing from 10.0.2.15 to 10.0.2.4 then the request flowing to facebook from 10.0.2.4. The data is sent from facebook.com back to 10.0.2.4 and then that data is encrypted and passed back to 10.0.2.15 through the ssh tunnel we had previously established. We can view this by sniffing the packets using Wireshark. Screenshot below:

A picture containing table

Description automatically generated