Evan Smith

Internet Security - Lab 2

**Task 1 – Using Firewall**

The rule for blocking Telent from VM A to VM B:



After ufw is activated on VM A, telnet is blocked when trying to hit VM B:

A screenshot of a computer

Description automatically generated with low confidence

The rule for blocking Telent from VM B to VM A:

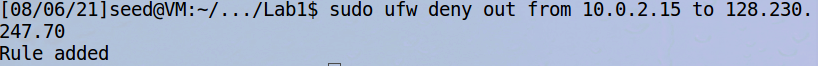


After ufw is activated on VM A, telnet is blocked from VM B:

A picture containing text, device

Description automatically generated

Lookup and generation of rule to block traffic from the SEED website:



Before adding rule:

Timeline

Description automatically generated

After Rule:

Graphical user interface, text

Description automatically generated with medium confidence

**Observations**: I was able to fairly easily block traffic from my machine to another, since I always knew the IP address and port that the request would come from. I found that blocking incoming messages was much tougher – trying to do the symmetrical thing from the last task did not work at all – blocking incoming traffic from SEED to VM A, rather than vice versa had no effect. I’m not sure why this is…

**Task 2 – Implementing Simple Firewall:**

Firewall code, pulled mostly from the lab, as well as book code:

Text

Description automatically generated

Making and assigning the module:

Text, letter

Description automatically generated

Pings still work, but telnets out do not:

Text, letter

Description automatically generated

Telnets in also do not work:

A picture containing text, device, gauge, close

Description automatically generated

And webpages along with all other HTTP traffic is totally blocked:

Graphical user interface, text, application

Description automatically generated

We can see the dropped packets in the log:

Text

Description automatically generated

**Observation**: While the process of adding the linux modules was new to me, I like the ability to hook into core system calls to do arbitrary execution. We are able to block entire ports easily, and could also do the same for specific IP addresses. This can extend into doing entirely different behavior per accessor, including processing calls with a spoofer for certain traffic, if desired.

**Task 3- Evading Egress Filtering:**

We keep the filters in place from Task 2, so there is no telnet activity allowed into or out of VM A, and no HTTP requests will be honored into or out of VM A either.

Setting up the SSH tunnel to VM B (I only used two VMs here):

Text, letter

Description automatically generated

We are now able to telnet out of VM A using localhost 8000:

Text, letter

Description automatically generated

Task 2.b – connecting to HTTP requests through the SSH tunnel

Setting up the SSH tunnel:

Text, letter

Description automatically generated

1. We can now get to the internet from VM A when Firefox is told to go through localhost 9000:

Graphical user interface, application

Description automatically generated

1. After breaking the SSH connection, we can no longer access the site, with a warning that the proxy server is refusing connections:

Graphical user interface, text, application

Description automatically generated

1. As soon as the SSH tunnel is reestablished using the same steps shown above, the site is accessible again. Traffic is flowing.
2. Observations about Task 2 overall- we see that we can evade filtering packets when the traffic is encrypted, such as through SSH. Though there is a firewall blocking both Telnet and HTTP traffic in and out of VM A, with an SSH tunnel we can pass encrypted packets out to VM B, which then executes the desired behavior and sends the response back via SSH. As long as we set our browser or terminal to listen to the proxy port, we can carry on as usual. This is carried out in the Wireshark readings:

Graphical user interface, application

Description automatically generated