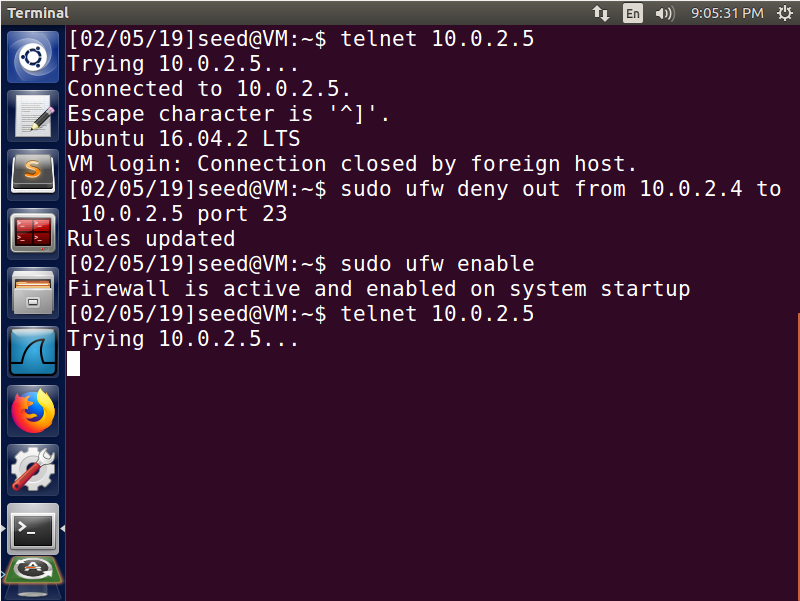
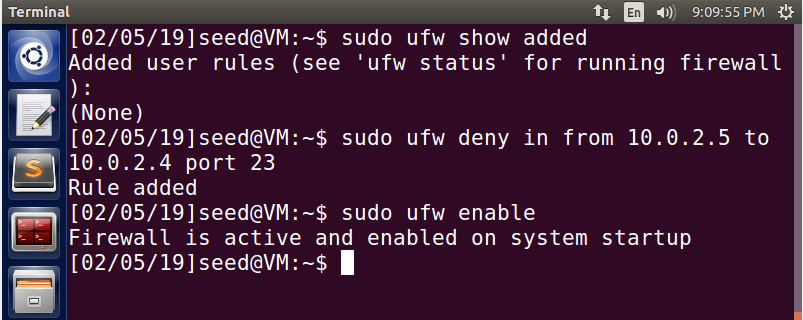
Task 1: Using Firewall

Prevent 10.0.2.4 from doing telnet to 10.0.2.5

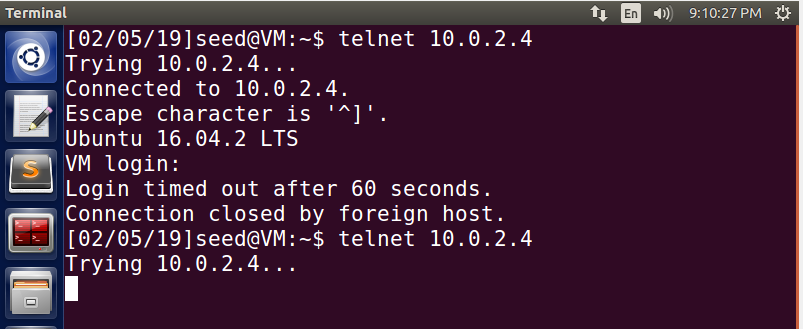


In this example, we first run telnet to another server, we add the rule to deny all packets from our IP to the target IP on port 23. Since telnet runs on port 23, we will be dropping packets. Thus, when we re-try telnet, it fails.

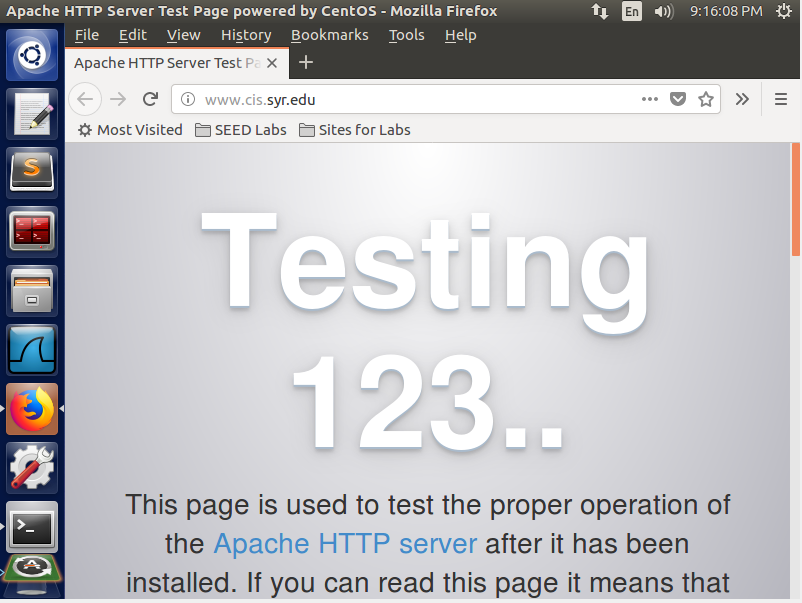
Prevent 10.0.2.4 from doing telnet to 10.0.2.5



Here are the rules that we added to stop a target from telnetting to our host machine. We set the rule to deny all traffic from the target to our host on port 23. Again since telnet runs on port 23, any traffic on port 23 to our host from the target will be dropped.

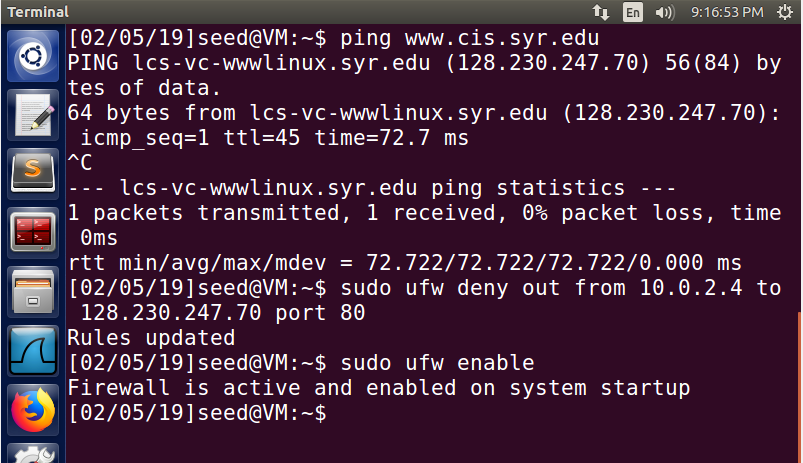


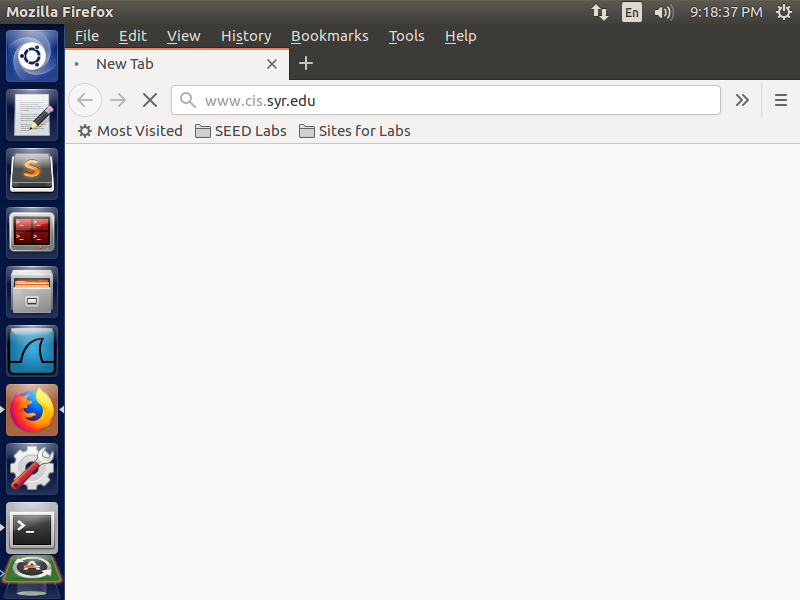
Here we see that we initially get a pass when trying to telnet to our host. Then after our rule is set, we get the packets dropped.

Prevent 10.0.2.4 from visiting www.cis.syr.edu

Here we have a successful vistitation of a website

☺☺☺

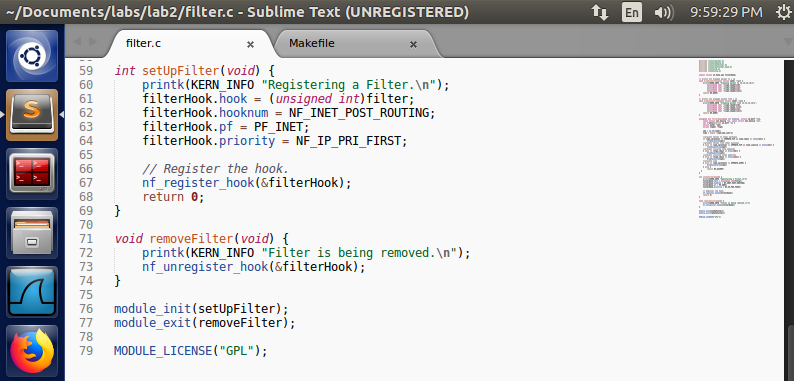
We first got the IP of the website by pinging it. Once we have the IP, we define the rule to stop all traffic from our host machine to the target IP on port 80. Port 80 is used for websites so this will not load next time.



Here we see that we do not get to the site after adding the rule.

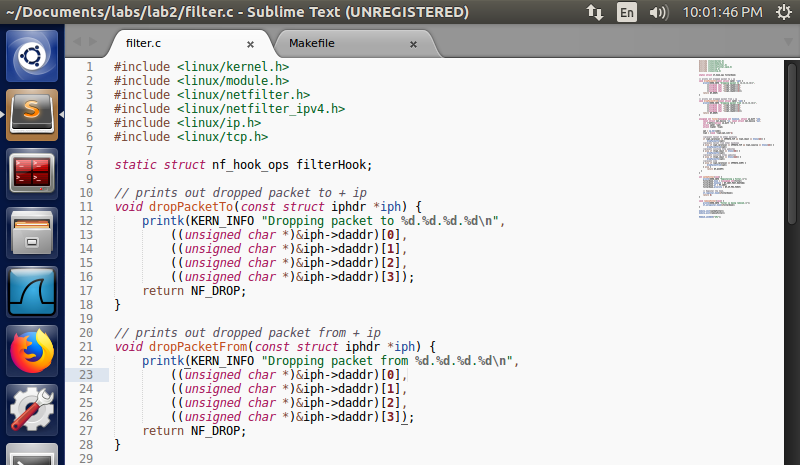
Task 2: Implementing a Simple Firewall

setUpFilter & removeFilter code

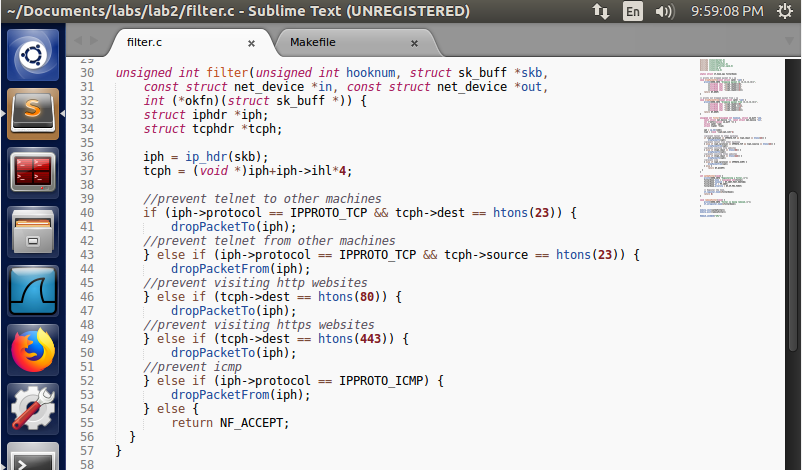


Here we have the code for setUpFilter and removeFilter which sets up our hook and prints this to the debug messages.

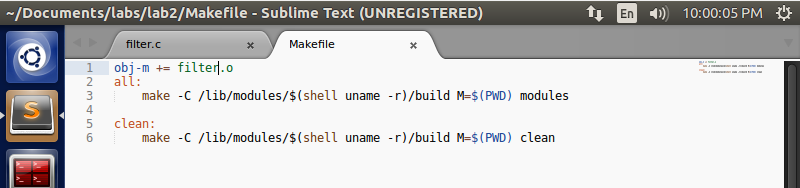
dropPacketsTo & dropPacketsFrom code



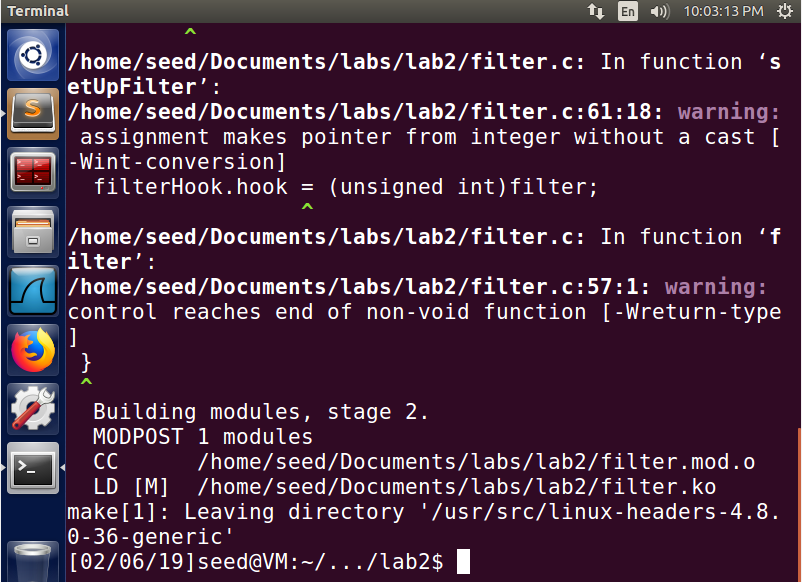
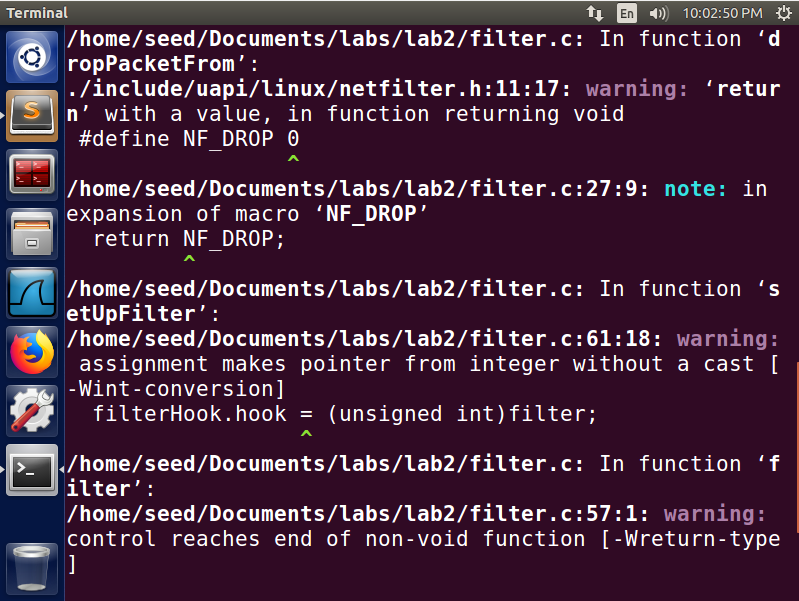
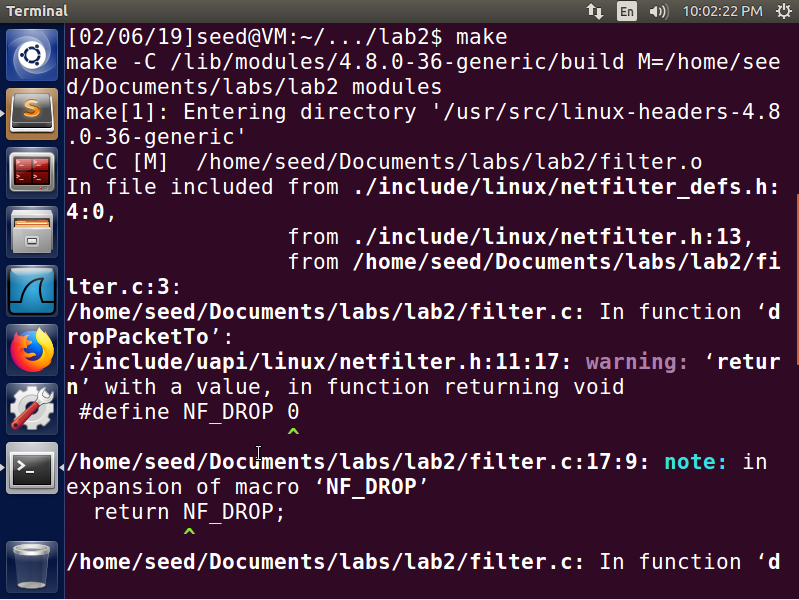
This code above is used to drop packets and print out a debug message showing the IP its dropping to/from

filter code

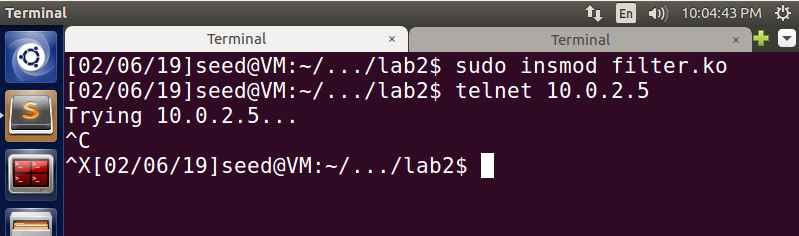
The filter code here takes the buffer received and splits it up to an ipheader and tcpheader. Once we have both, we can use the header details to filter. We created 5 filters. The first filter is if the destination port is 23 meaning that if we have a telnet request, we will drop it. Second, we have a filter if the source port is 23, this way the sending of the telnet request is dropped. Next, we block all HTTP sites by dropping packets from port 80. Following that we drop all packets from port 443 which in turn blocks all HTTPS sites. Lastly, if the protocol is ICMP, we will be dropping them. If the filters don’t catch anything, then the packet is accepted.

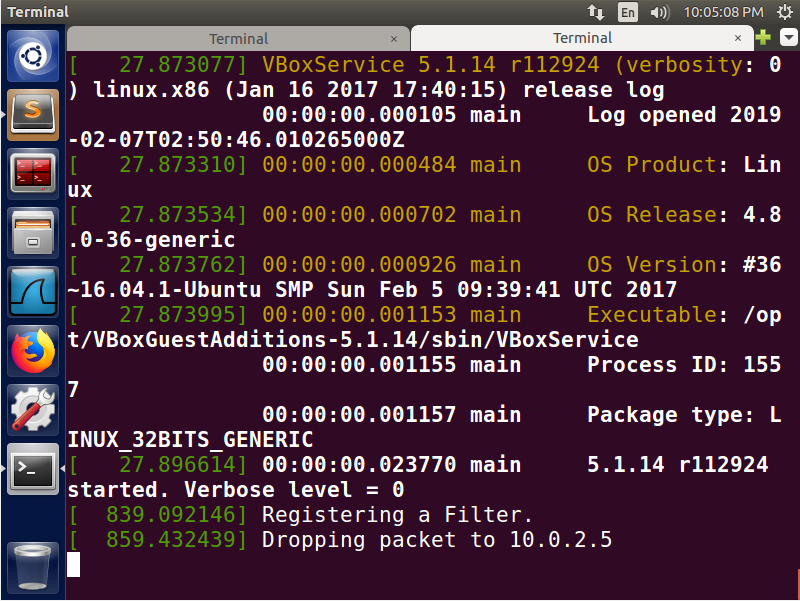


In our makefile, we show the object file that we want to create and then set the two make commands. We use ‘make’ to run and compile this to create the modules.

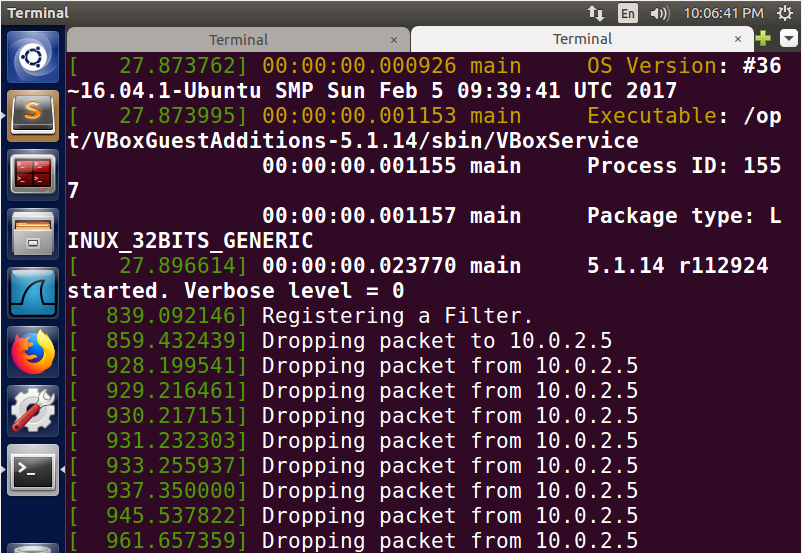
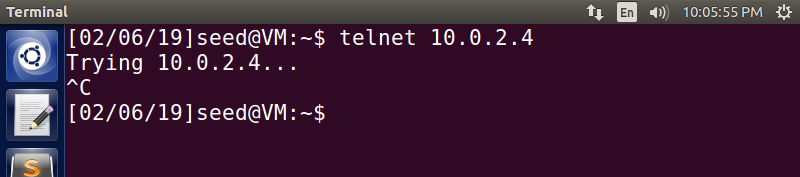


Here we have the output from the make command.

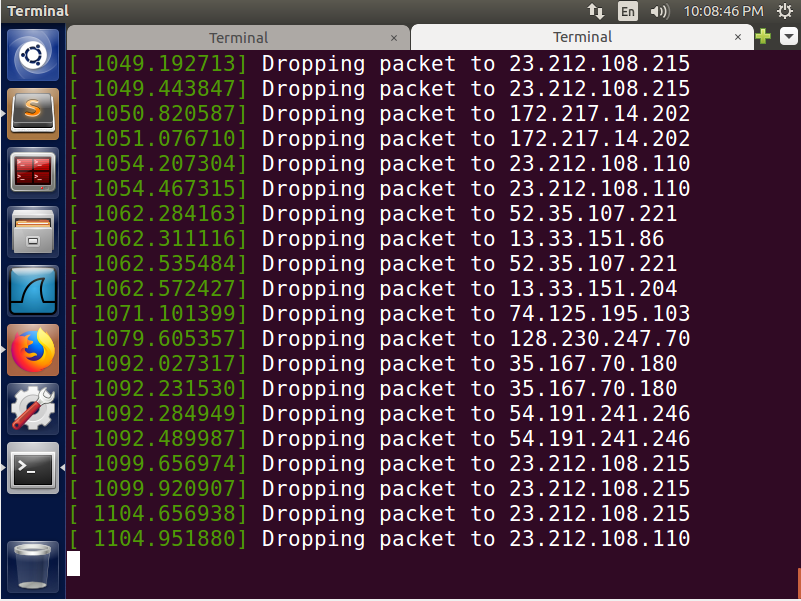
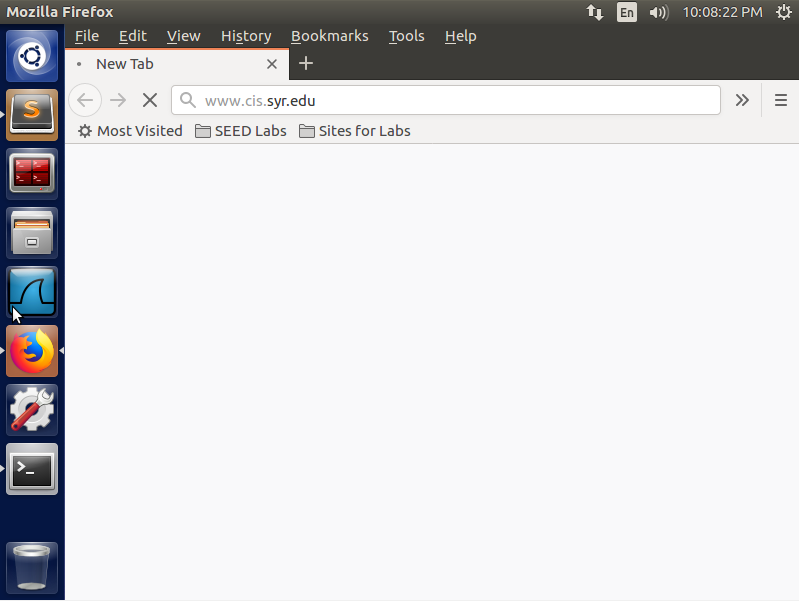


We then insert the module created!

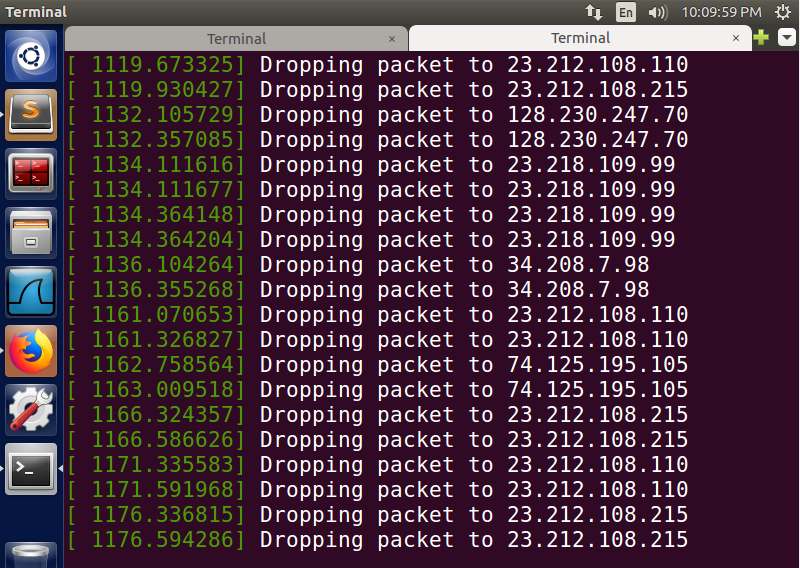
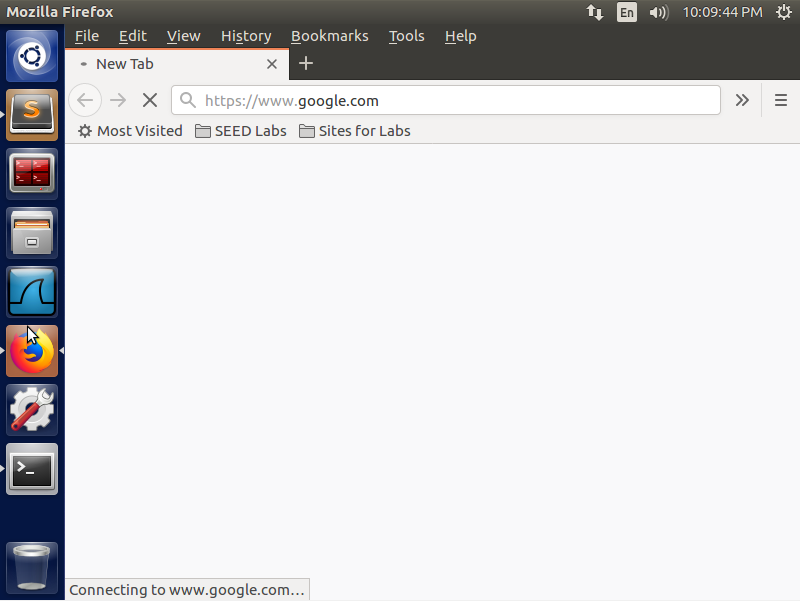
Here we see that when we run the telnet command, it never reaches and we can see in the dmesg output that we dropped the packet.



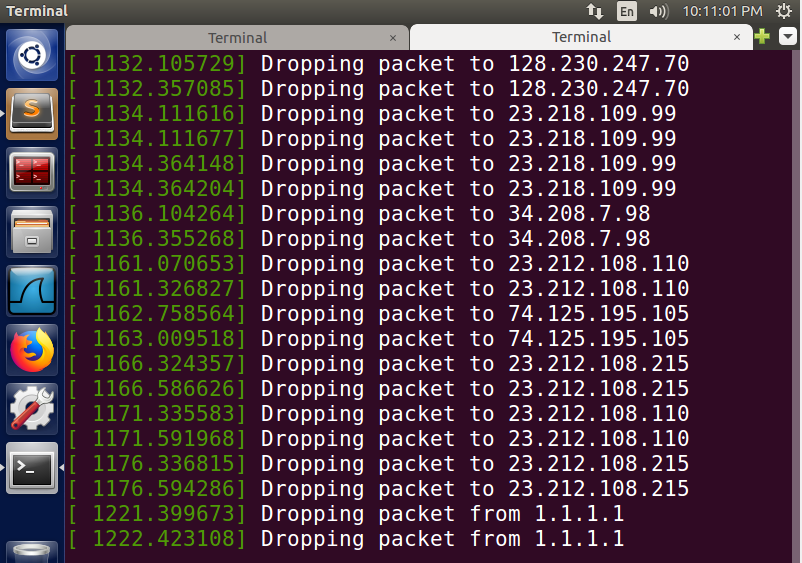
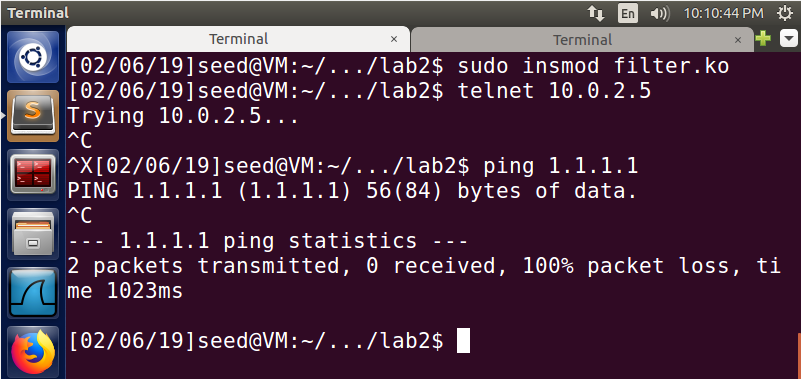
Here we see that we run the telnet command from 10.0.2.5 and the dmesg output shows the packets being dropped



Here we try to visit an HTTP site (cis.syr.edu) but due to our filter dropping all packets to port 80, we see the packets dropped.

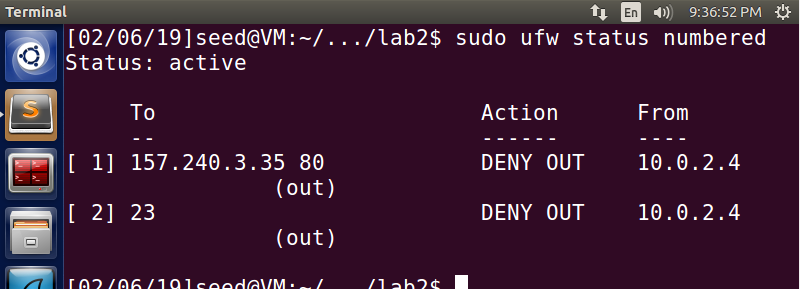


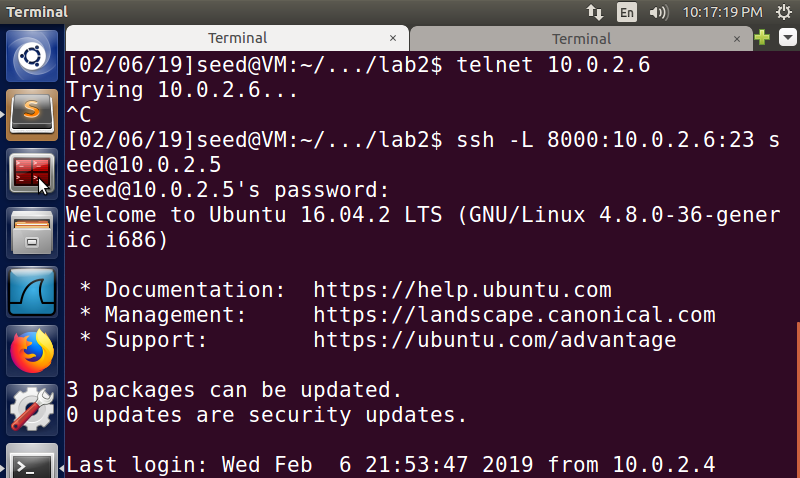
Here we try to visit an HTTPS site (https://www.google.com) but due to our filter dropping all packets to port 80, we see the packets dropped.



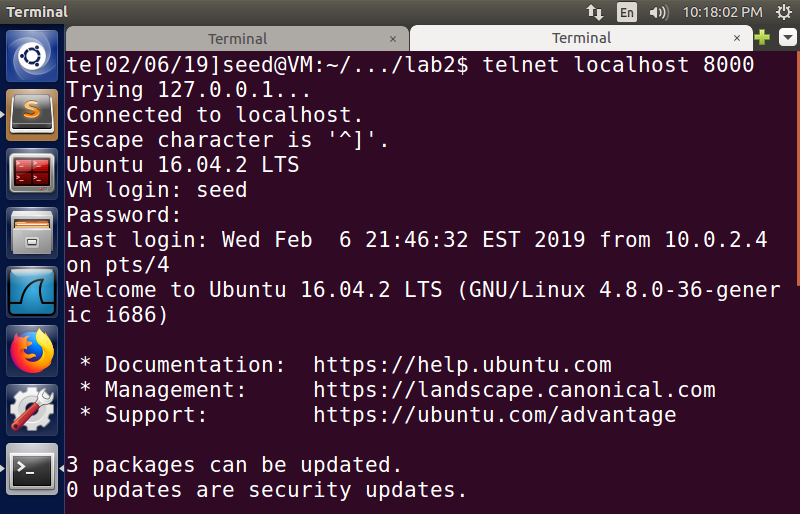
We also try pinging a random server using IP packets and we see that no packets are returning, thus we get 100% packet loss. Looking at the debug messages, we see the two sent packets were dropped. Thus since they never reached 1.1.1.1, we never got a response.

Task 3: Evading Egress Filtering

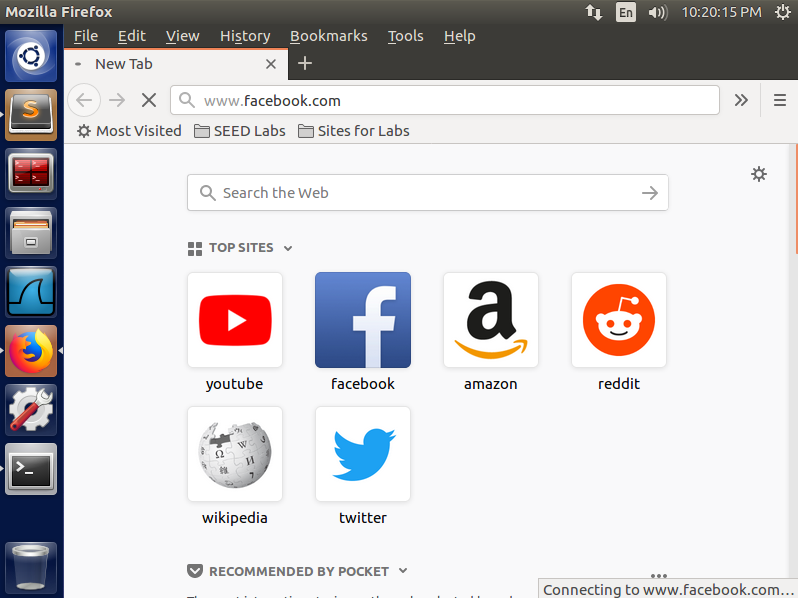
Here is the rule that we are using to block all outgoing traffic to external telnet servers by blocking port 23. We also have the rule to block all outgoing traffic to facebook.com. We captured a ping request to facebook to grab the IP and blocked it on port 80.



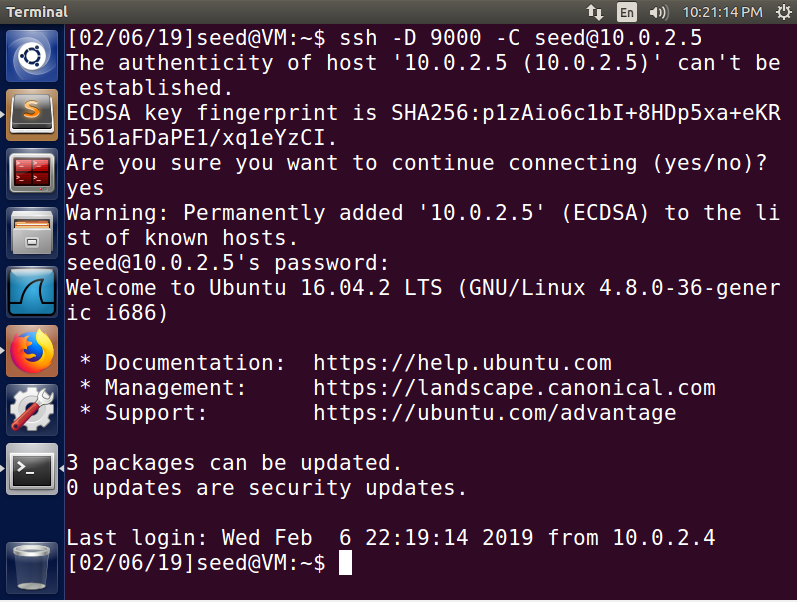
Here we see the telnet request and see it fail. We then run the command to create the SSH tunnel to our remote machine using our localhost port.



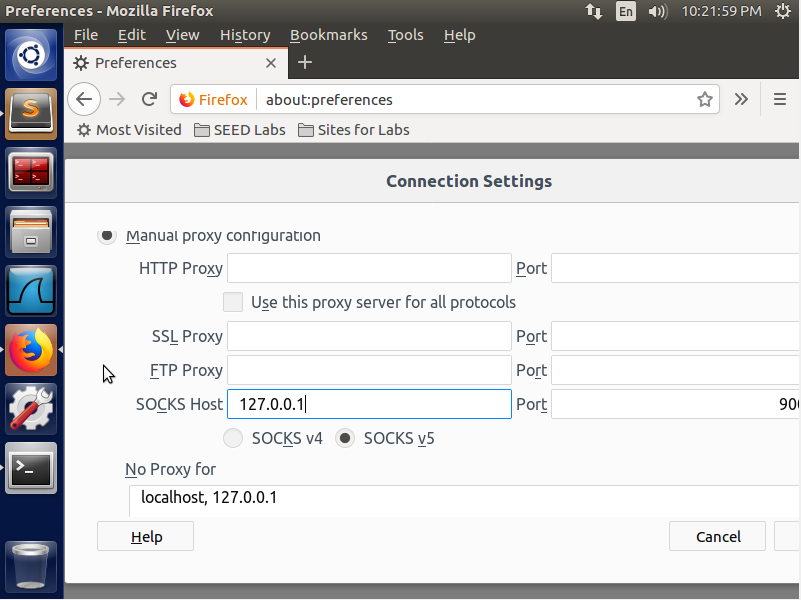
We then switch over to out other terminal and connect to localhost 8000 which allows us to telnet to the target machine via the created SSH tunnel and bypassing our filter.



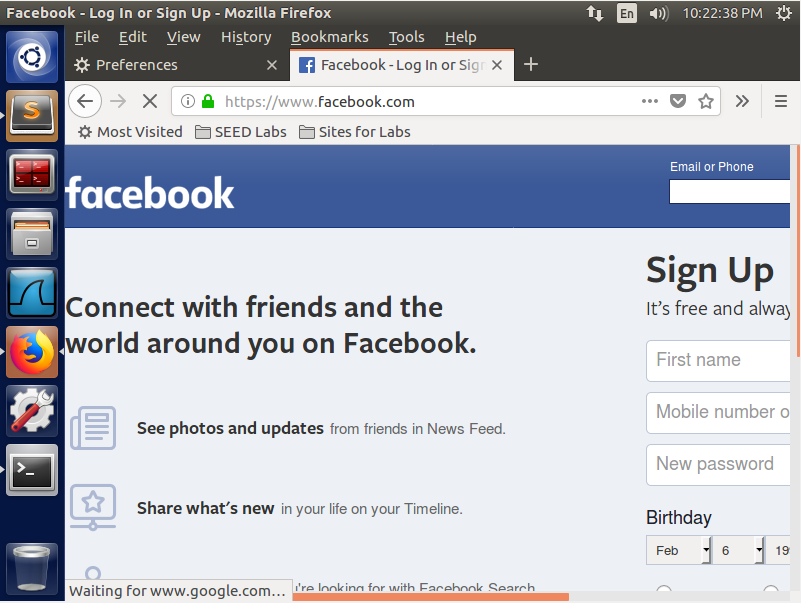
Here we try to access facebook and get no response back due to our filter blocking that IP on port 80



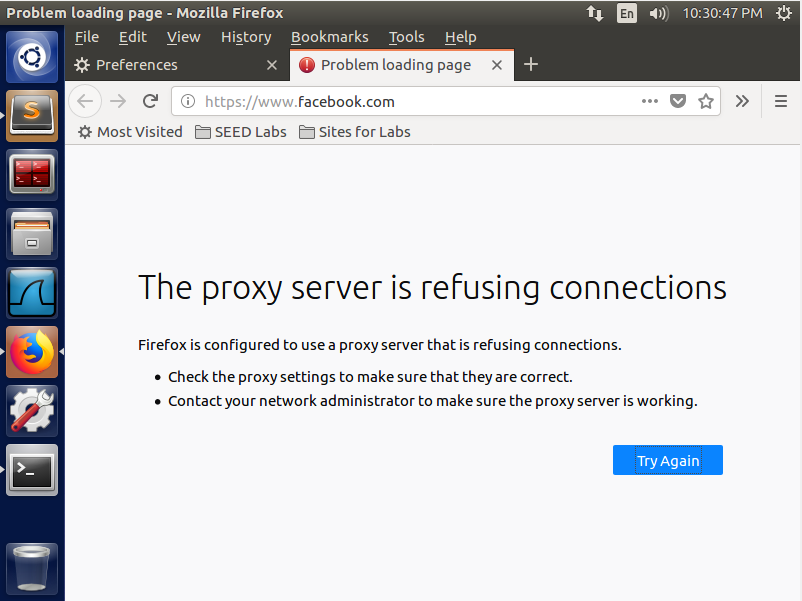
We then set up our tunnel on port 9000 to connect to our remote machine. Now if we connect to our localhost on port 9000, we can proxy to 10.0.2.5 (which is not affected by our firewall).



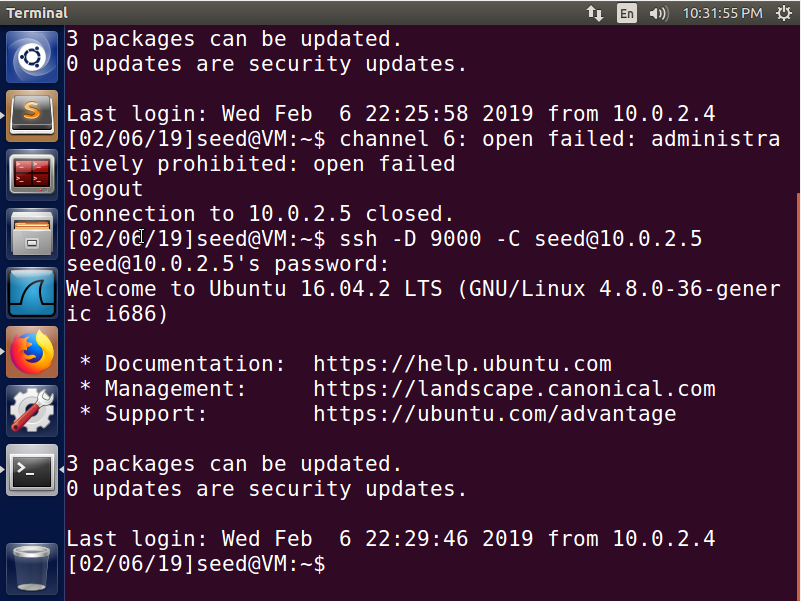
Now, we have to set up Firefox to use the details we set. We point it to localhost and port 9000 to take advantage of the created tunnel.



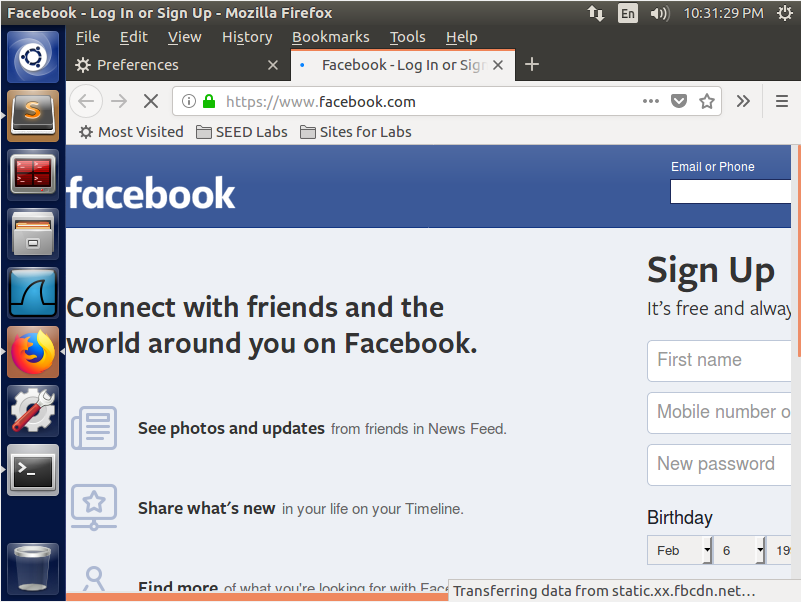
We can now get access to facebook again after setting up the tunnel and allowing Firefox to use the tunnel.



We then break the tunnel, clear history/cache, and then refresh the page. Now we see that we can no longer proxy due to the SSH tunnel being disconnected.



We re-establish the tunnel and refresh again.

  
Here we see that once we refresh after the tunnel has been re-established, we are able to bypass the firewall and access Facebook again. We can tell that while the tunnel is established, our packets go to 10.0.2.5 instead of our local 10.0.2.4 directly. When the tunnel is not there, the packets try to go to 10.0.2.4 but are blocked at port 80 due to the firewall.