Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

Task1

For task 1, we used Ubuntu 16.04 LTS VMs for the client and the server. The server has apache installed as the webserver and open ssh configured to test our SSH in later tasks. The Client is only connected to the “ClientNetwork”, Server is only connected to the “ServerNetwork”, FirewallA is connected to the “ClientNetwork” and the “Internet”, and FirewallB is connected to the “ServerNetwork” and the “Internet”.

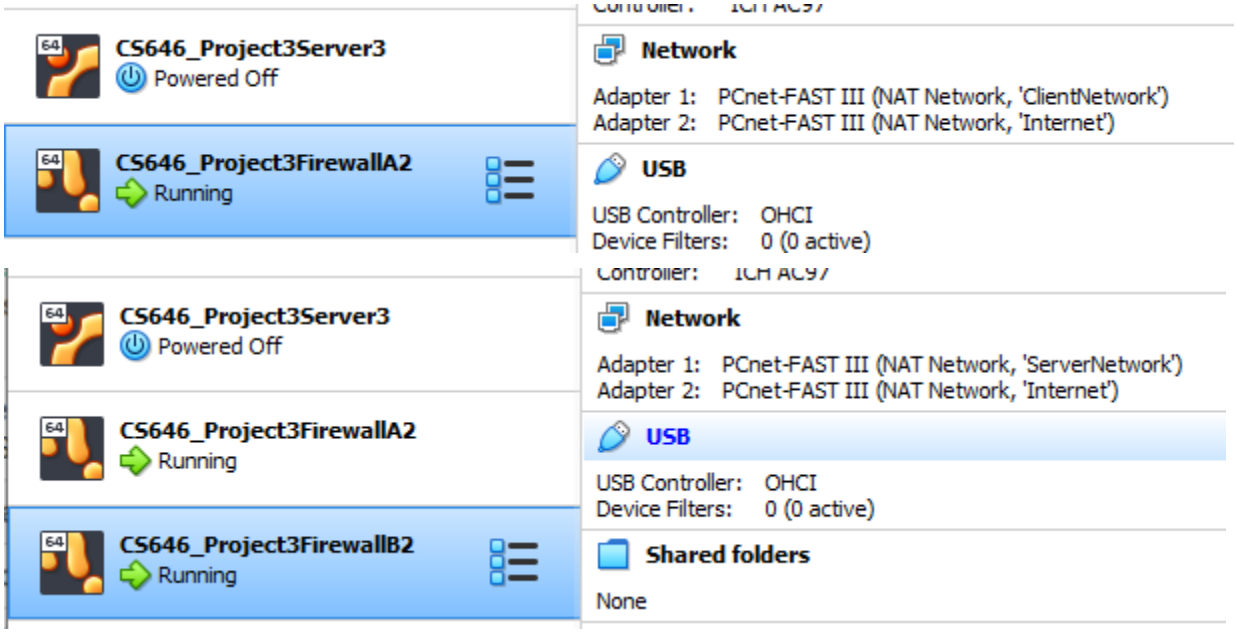
ClientNetwork	10.47.1.0/24
ServerNetwork	10.47.2.0/24
Internet	10.47.3.0/24

The IP addresses are statically assigned as follows:

Client	10.47.1.14
Server	10.47.2.13
FirewallA LAN Interface	10.47.1.5
FirewallA WAN Interface	10.47.3.6
FirewallB LAN Interface	10.47.2.4
FirewallB WAN Interface	10.47.3.7

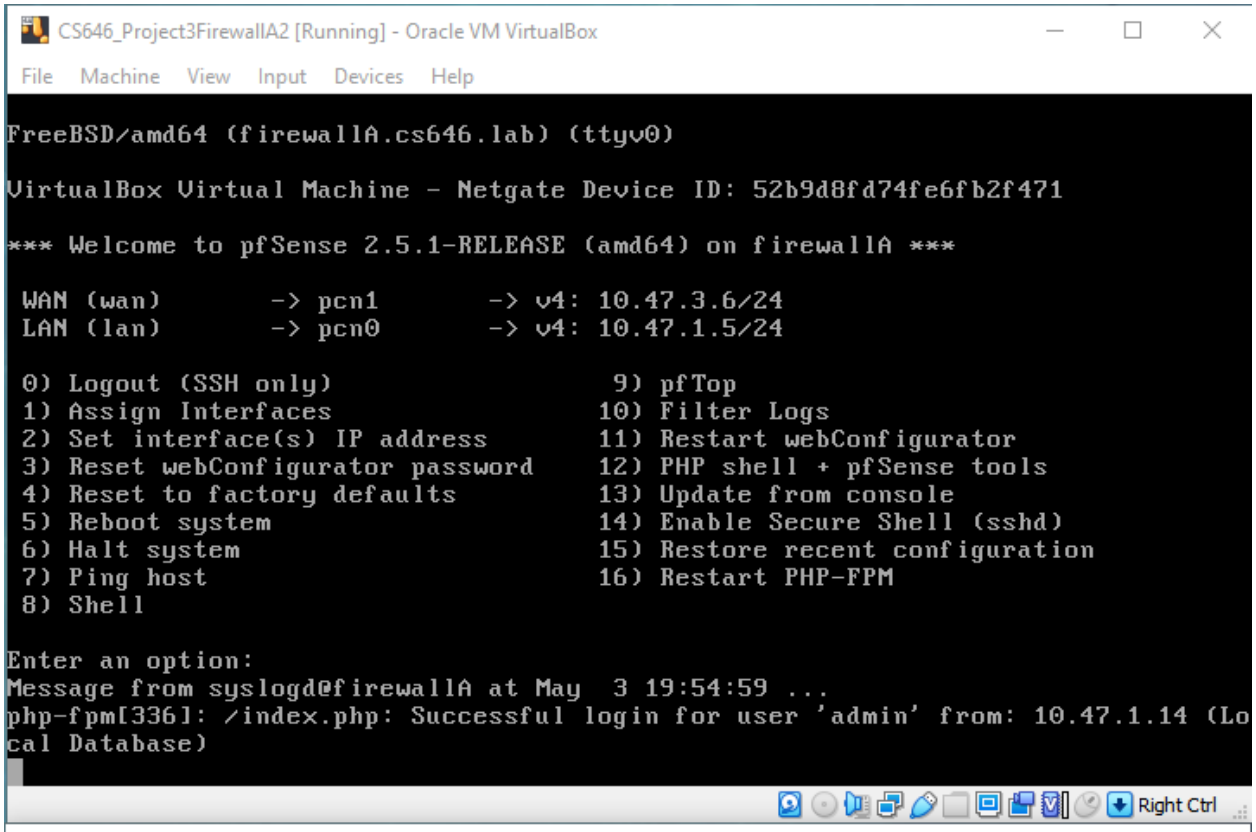
Below are screenshots displaying the IP addresses of the devices used.

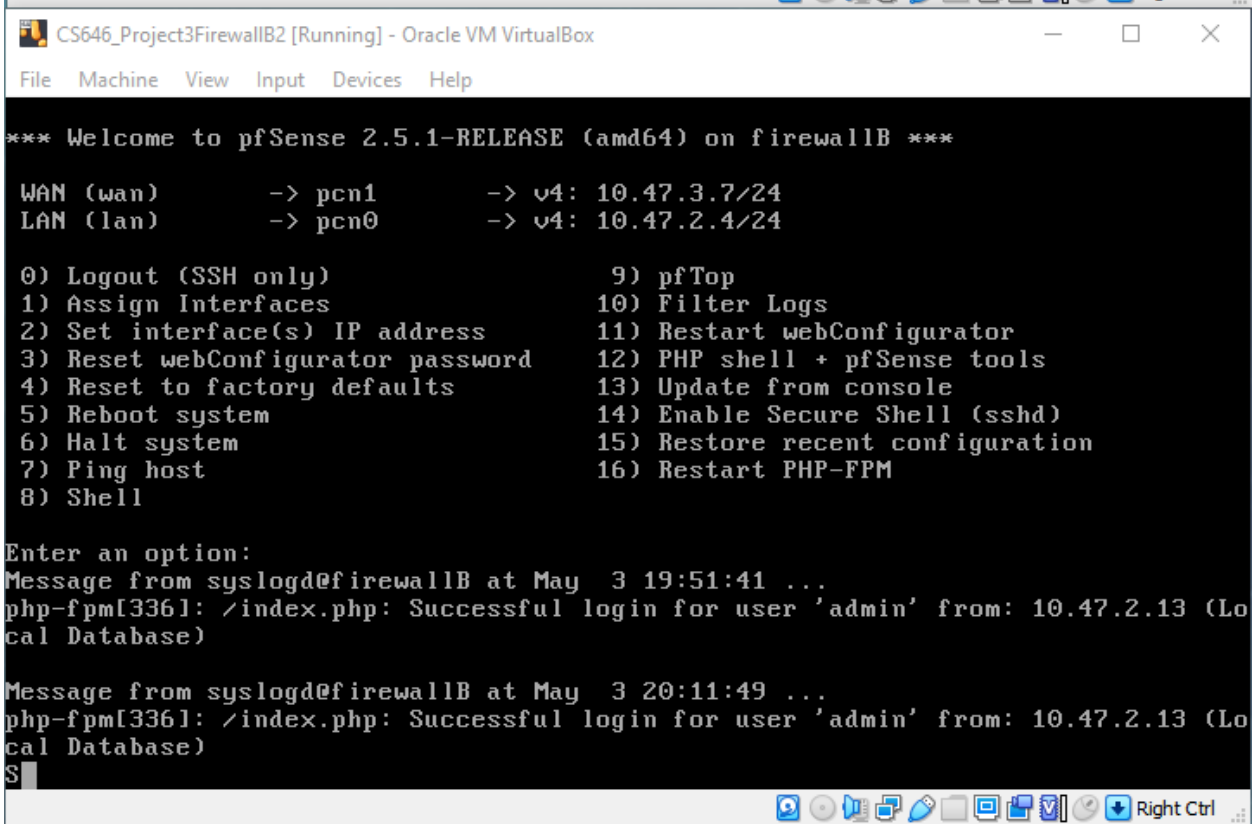
Adapters FirewallA and FirewallB are connected to.



Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

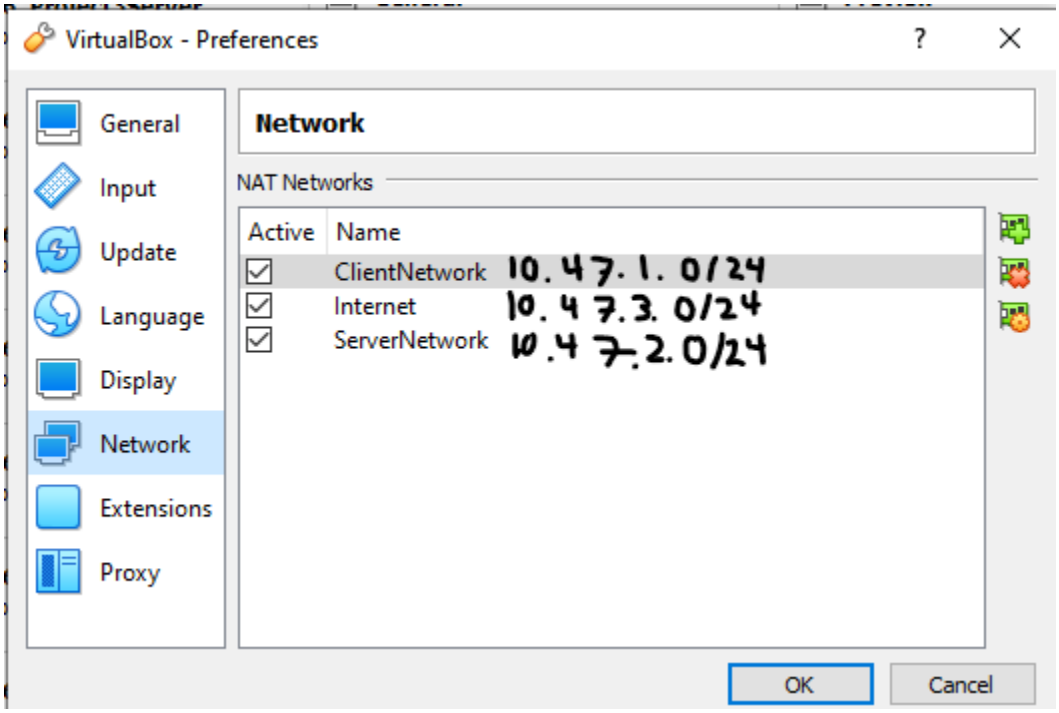
IP addresses of FirewallA and FirewallB



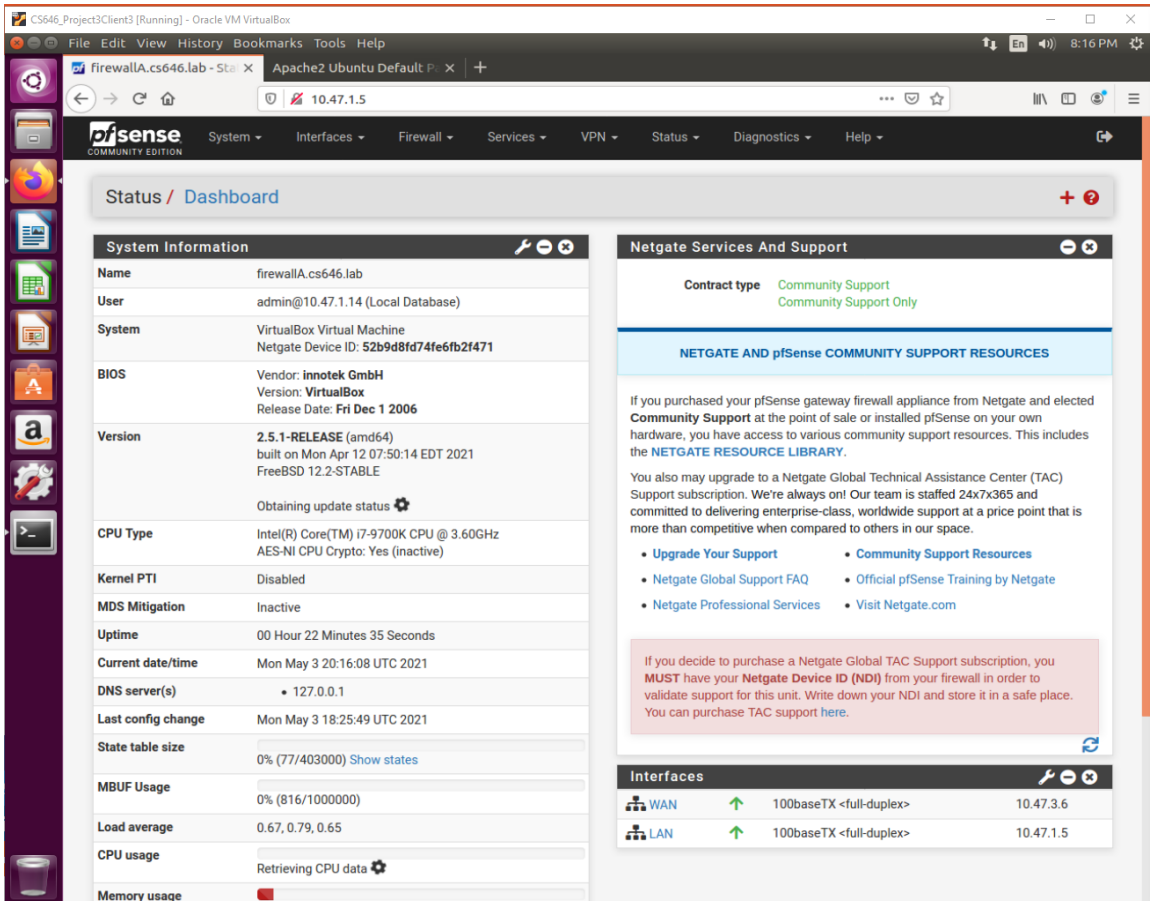


Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

Virtual Networks for the project.

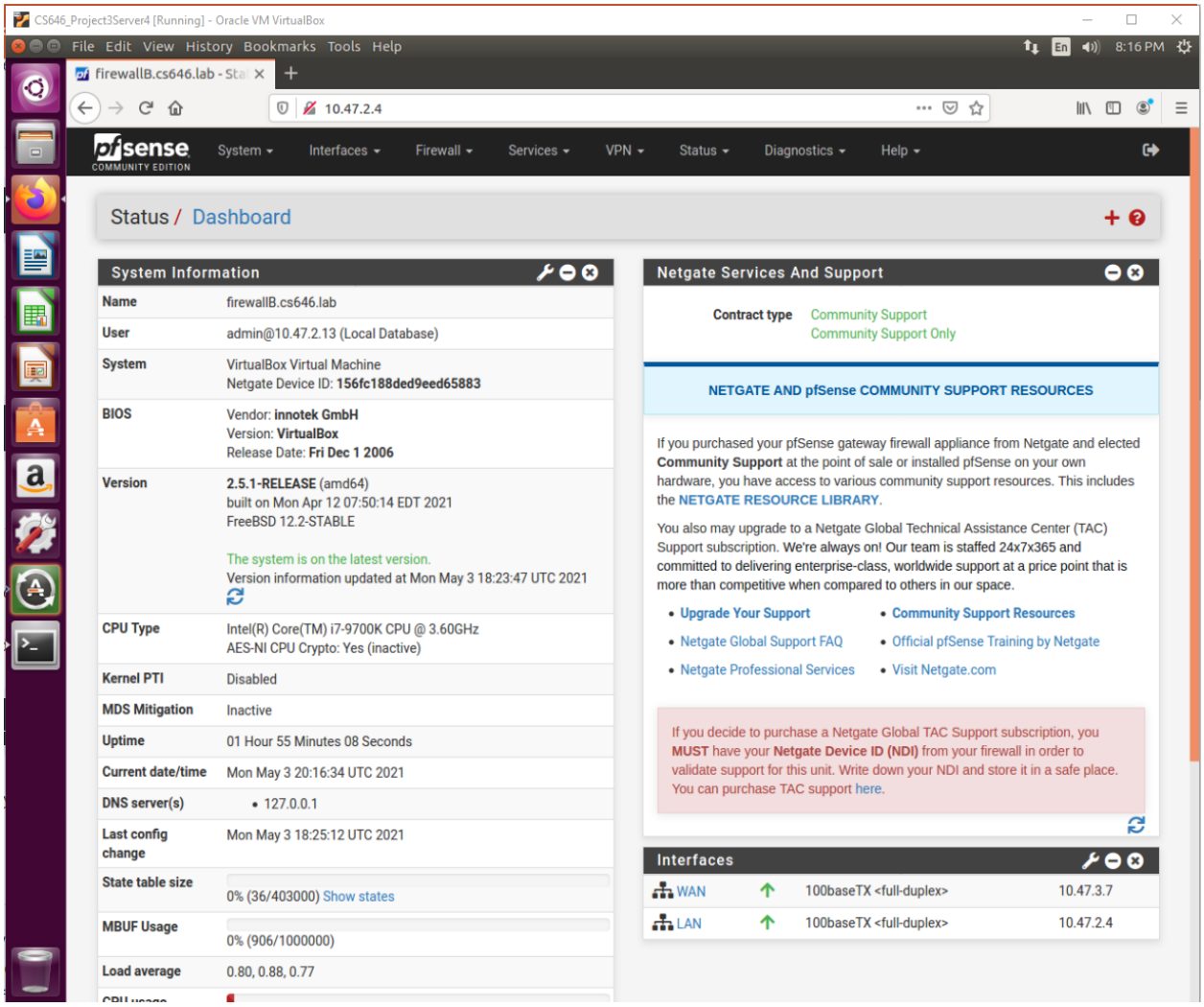


FirewallA's pfsense webpage



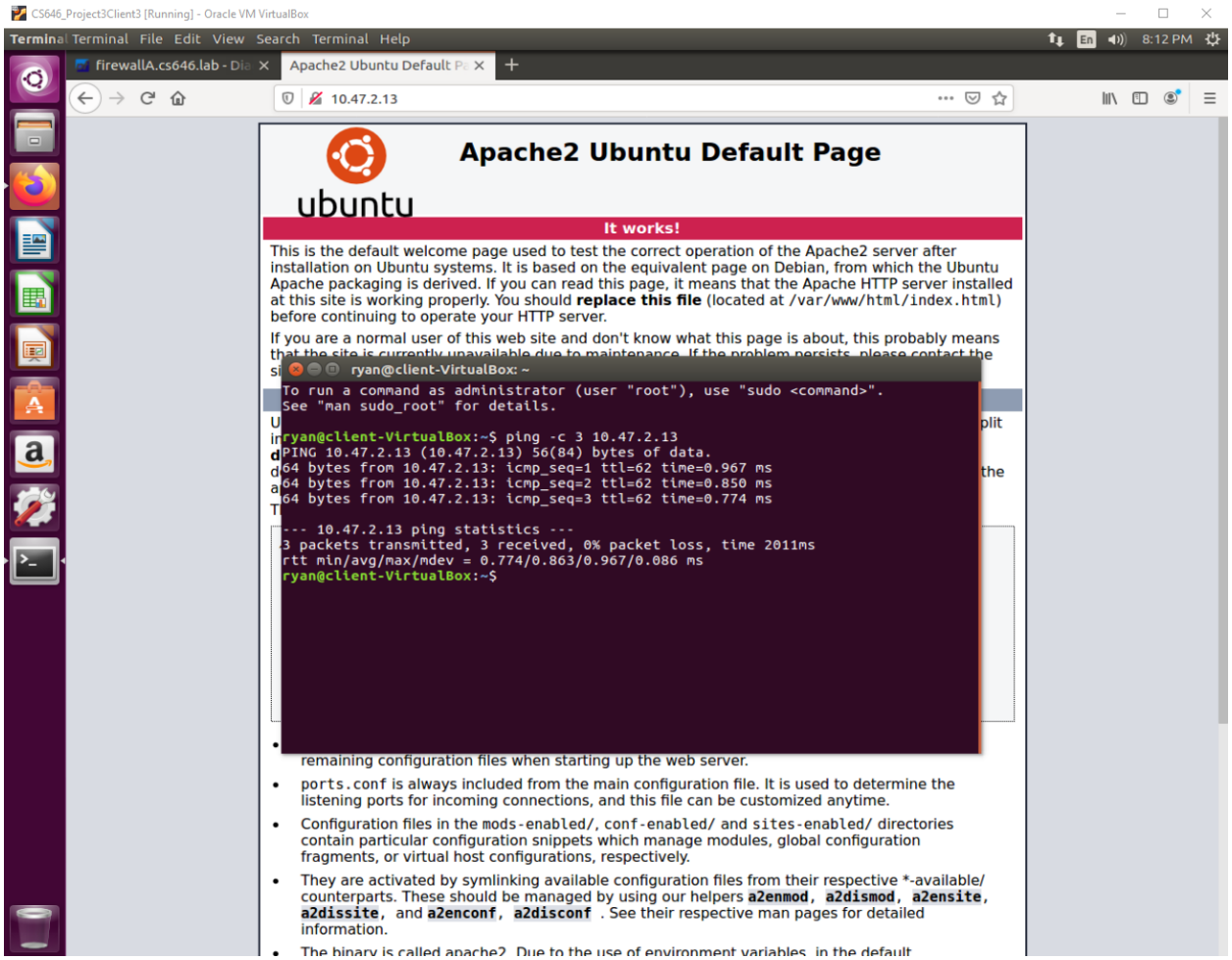
Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

FirewallB's pfsesne webpage

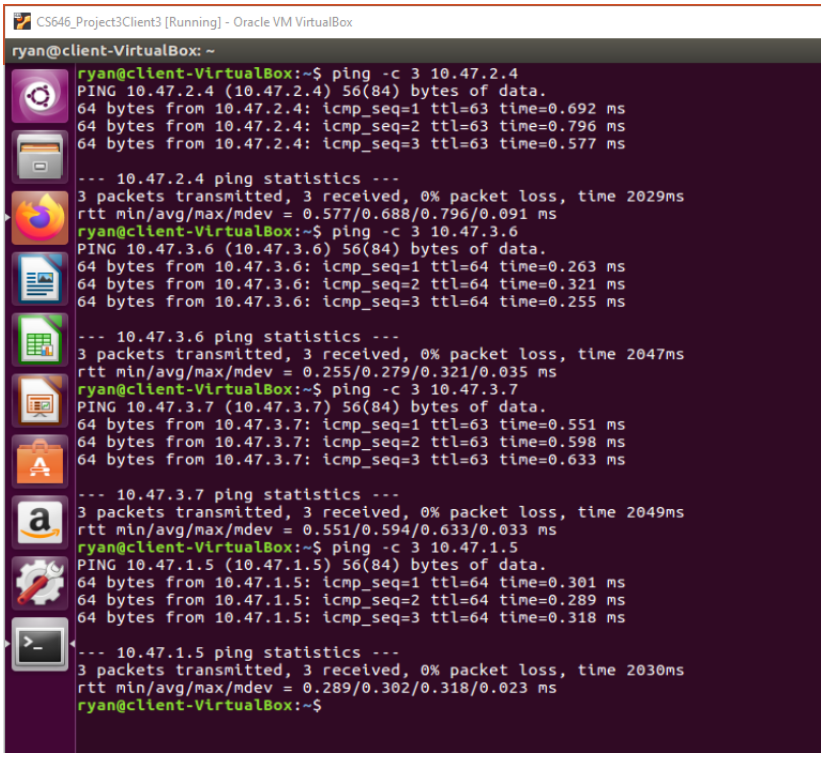


Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

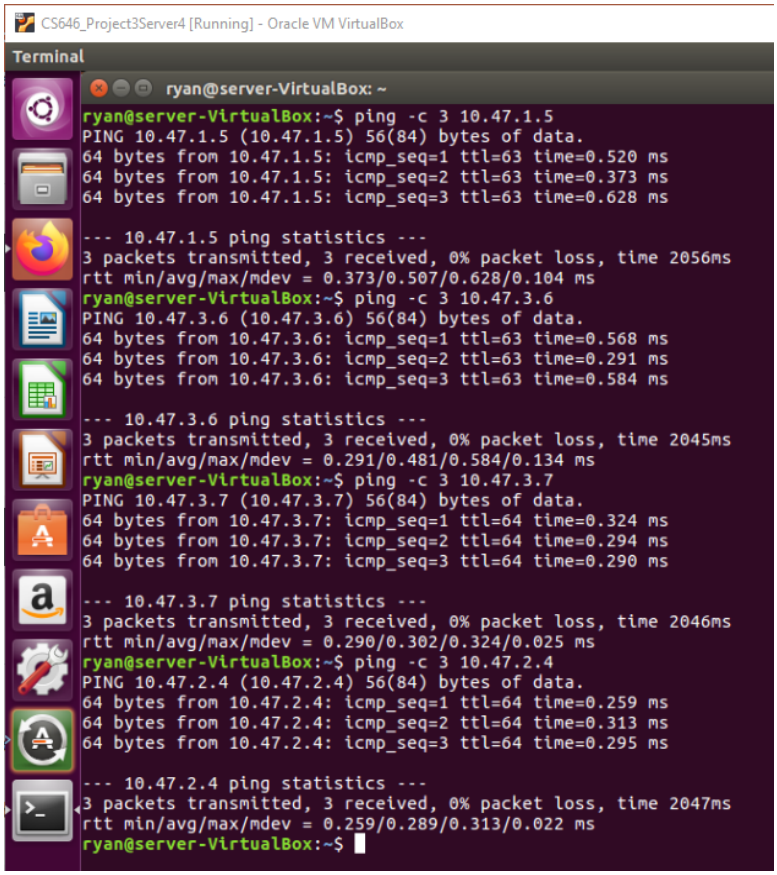
Ping test to web server from client. You can see in the top left corner we are on device CS646_Project3Client3. In the background is the working web page on 10.47.2.13. The terminal shows the ping test to 10.47.2.13.



More ping tests from the Client to all.



Ping test from Server to all

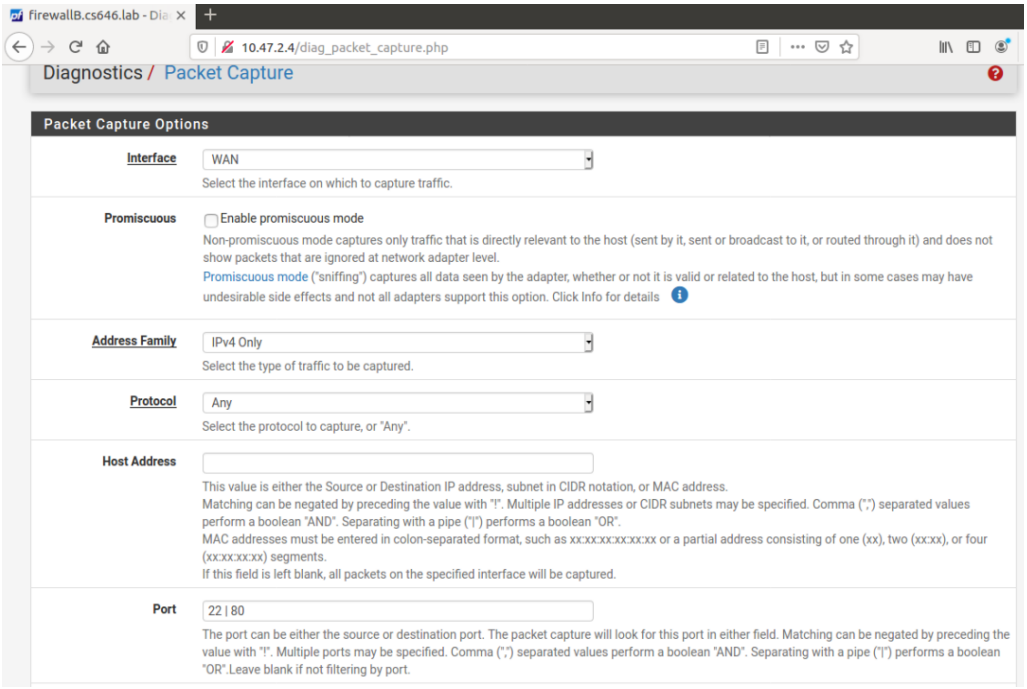


Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

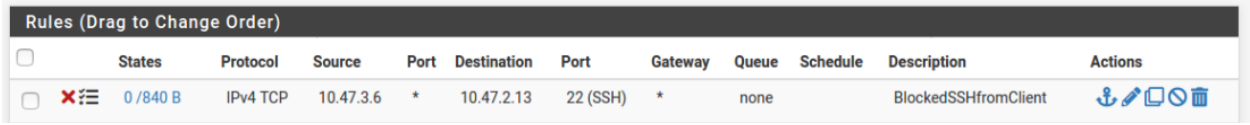
Task2

In order to block SSH, we followed two principes. First, we wanted to block/reject the packets at the door. We don't even want them getting into the firewall if it is bound for the server on port 22. Second, we want to also guarantee that this works on both firewalls. So we start with FirewallB's WAN interface.

Below is how we captured the packets to determine what rule to make. We selected Port 22 and 80 (SSH and HTTP) so that we could make sure that the IP was the same.

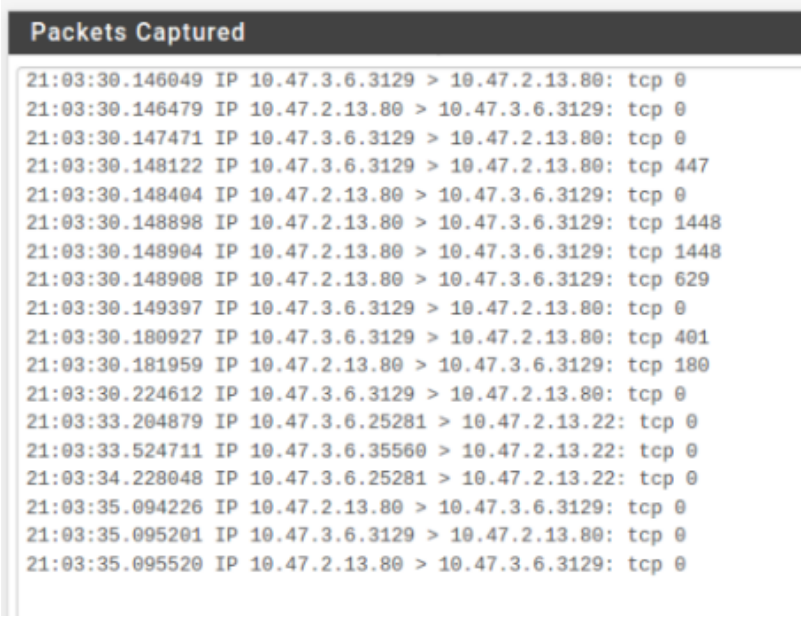


We found that 10.47.3.6 was the source and 10.47.2.13 was the destination. So, we make a firewall rule on the WAN interface, **blocking** traffic with that source and destination on port 22.

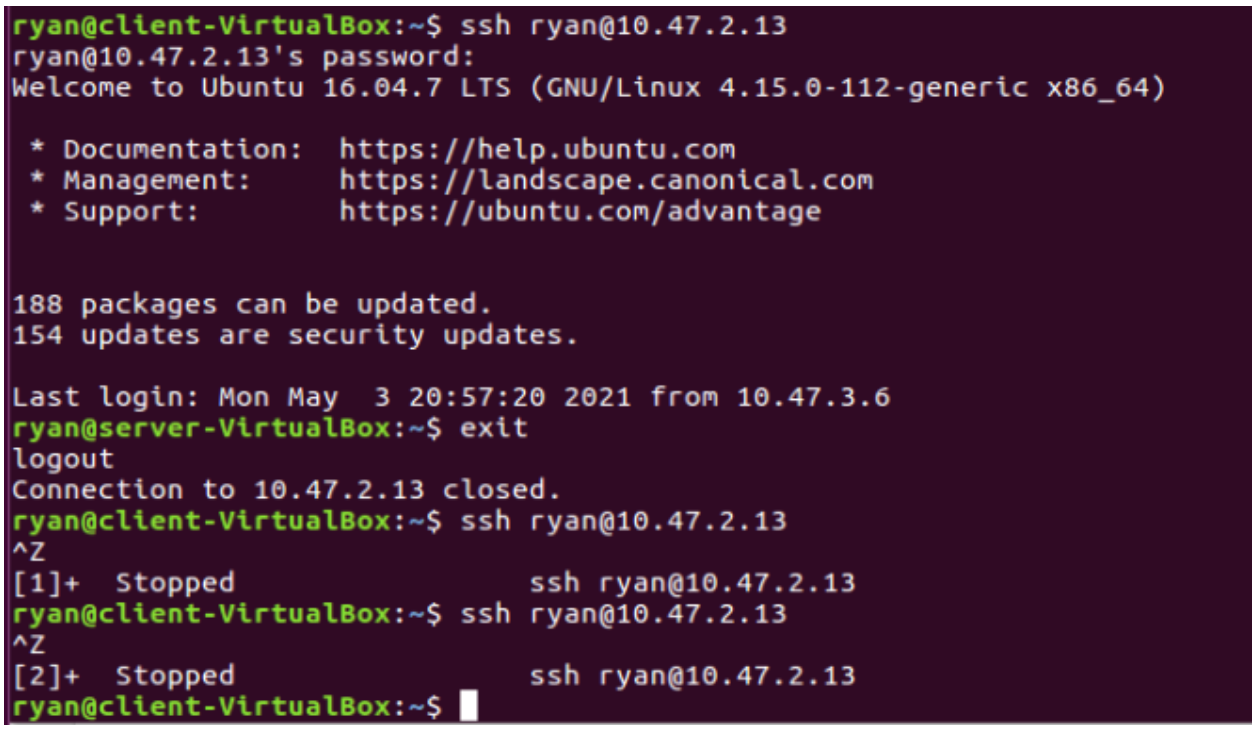


We decided to specifically block SSH to only the server. We don't know what else this network will be used for in the future, so no need to block everything.

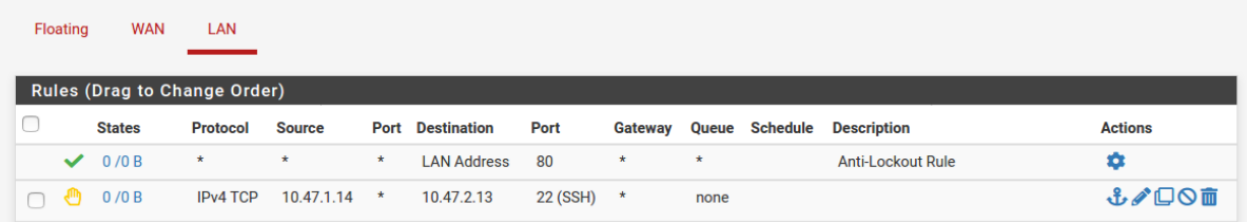
In the next screenshot, you can see the packet capture from this. We tried to SSH a few times and refreshed the default apache webpage to prove it worked.



And here is a screenshot of the terminal on the Client. We first show that it was possible to connect before the firewall rule is added. We also explicitly chose to block here. On the WAN, we don't even want a reject, only block.



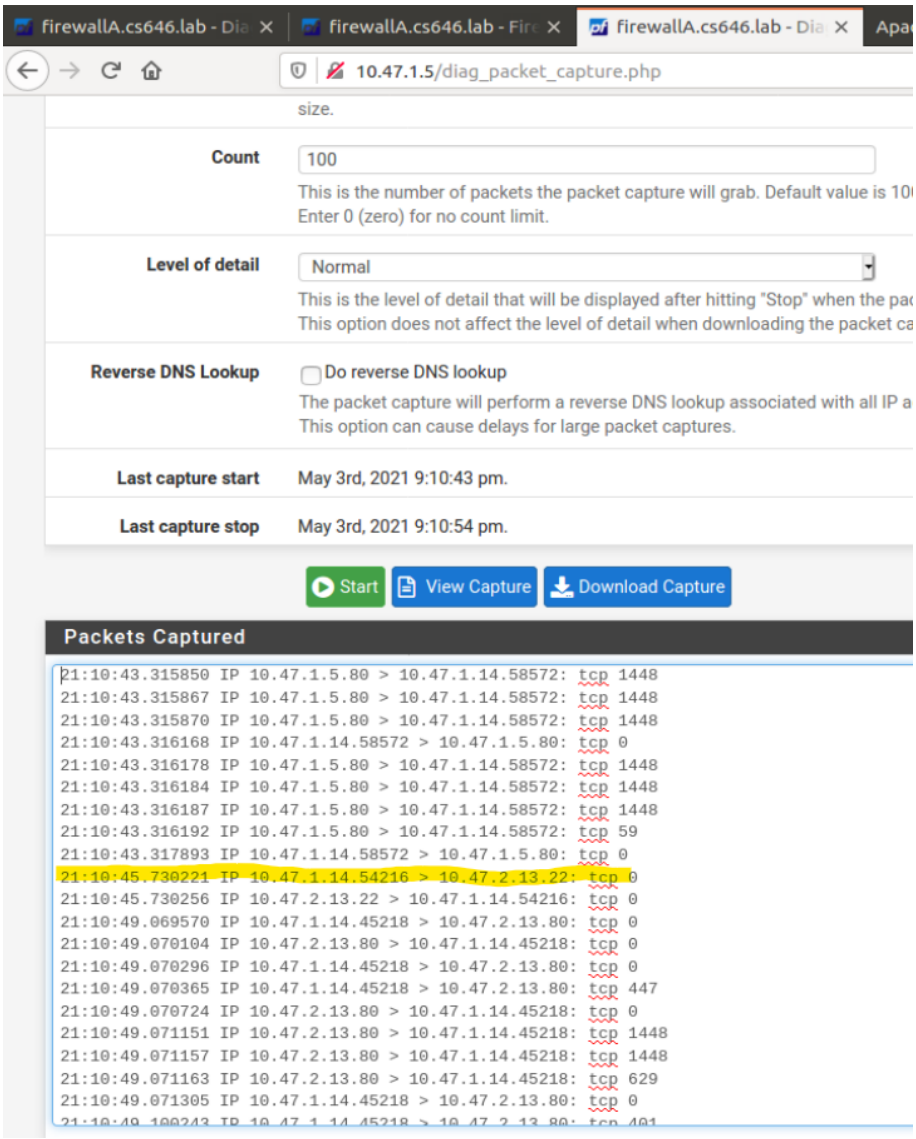
Next, we configure a Firewall Rule on FirewallA's LAN interface. This time, the source is 10.47.1.14 and destination is 10.47.2.13. This is because we have not gone through the firewall yet, we are still in the LAN. In the previous example, the packets were coming through the "Internet" (10.47.3.0/24 subnet).



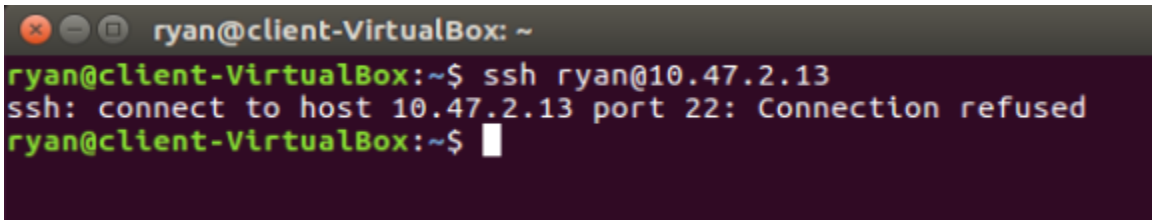
Instead of blocking here, we politely reject. Clients within the LAN shouldn't be sitting around waiting for a response. We let them know so that we don't waste our clients time.

We then test it out. Below we have our packet capture. We only tried to SSH once, to which we saw the connection was refused. In our packet capture, we highlighted the line where SSH was blocked. The 0 signifies nothing passed through in this case.

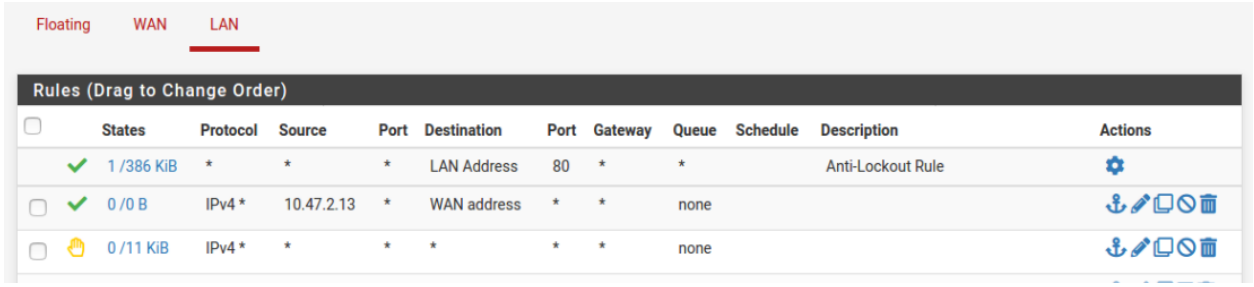
We also have some communication between our firewallA pfsense webpage and the client itself, which is signified by any packet that contains a 10.47.1.5. The packets that contain 10.47.2.13.80 are our packets send to the apache web server across http.



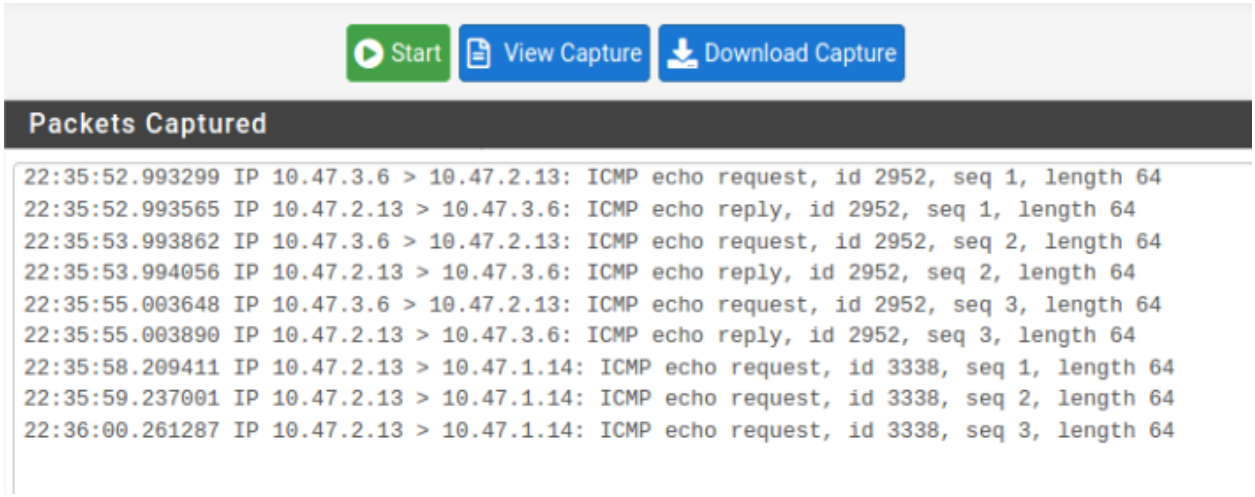
In this picture we see the Connection refused message. This important distinction in our rules proves to us that this is getting caught at the LAN interface on FirewallA.



Next, we want to stop unsolicited messages from the server to the client. Our thinking on this was to block it at the LAN interface of FirewallB. So we added a Firewall Rule on the LAN Interface that rejects all traffic. Just above that, we add another rule that states to allow any IPv4 packets from the network 10.47.2.0



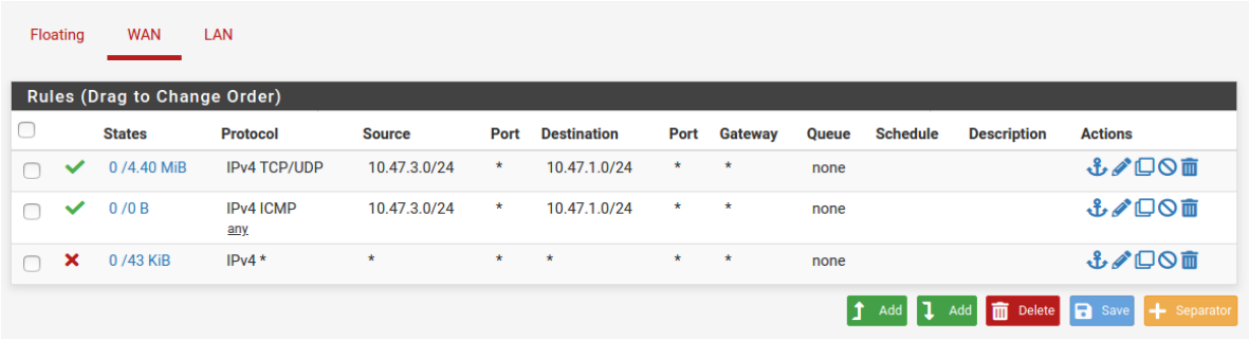
To test this, we went to the packet capture tab and only looked for ping requests. In theory, ICMP echo requests from the client to the server should pass through the LAN, but ICMP echo requests from the server to anyone will be blocked. Below are the results of our packet capture with this rule.



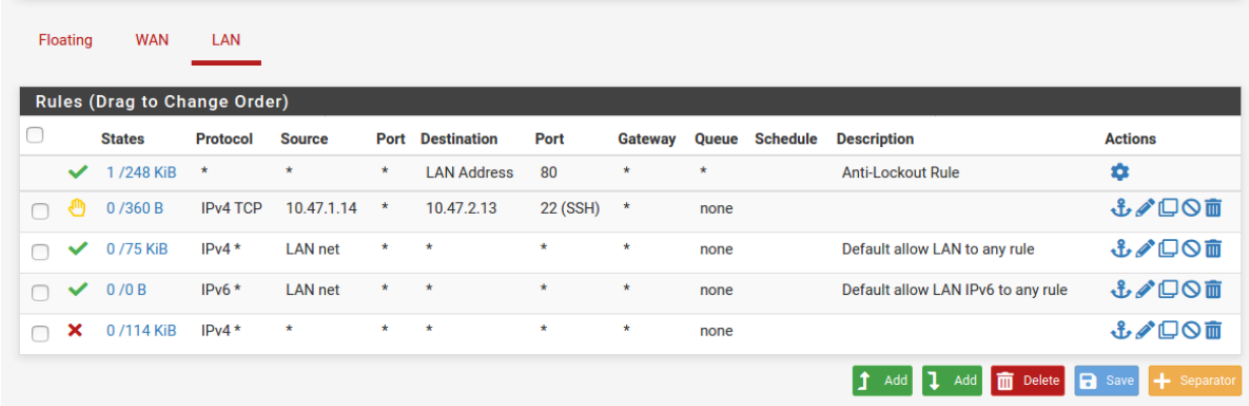
As you can see, the ICMP packets from 10.47.3.6 are allowed to pass through. These pings are actually from the client, 10.47.1.14. However, when we try to ping from our server to the client, we only see ICMP echo requests. This is due to our firewall rule only allowing packets out that use a WAN address, or came in from the WAN first amd therefore were solicited.

As a last little detail, we just wanted to clean up our firewall rules a bit. The final firewall rulesets for Task 2 is contained in the screenshot below

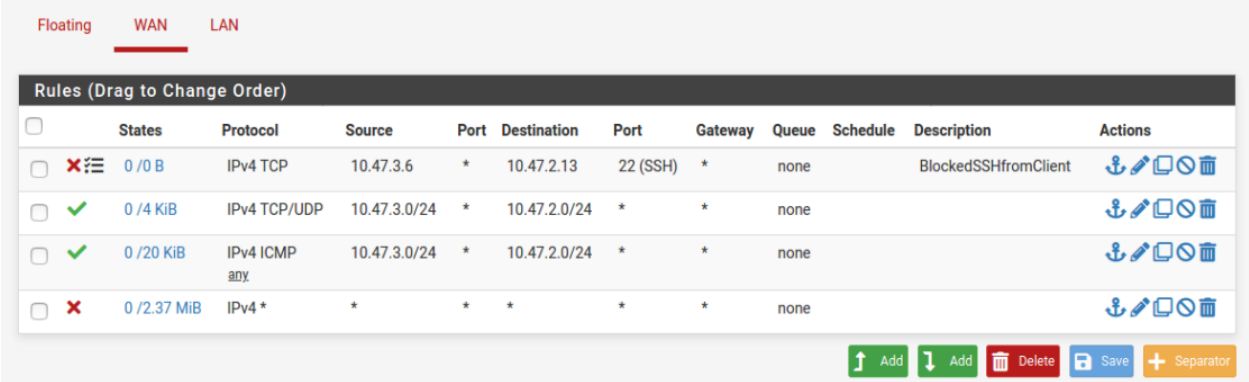
FirewallA WAN final ruleset. We decided to only allow TCP, UDP and ICMP traffic. Seems safe for the time being.



Firewall LAN final ruleset. Here, we are preventing SSH first. That stops the client from using SSH to connect to the server. The final rule is to not allow just anything to pass through the LAN interface. The other two were default rules added by pfsense before any modification.

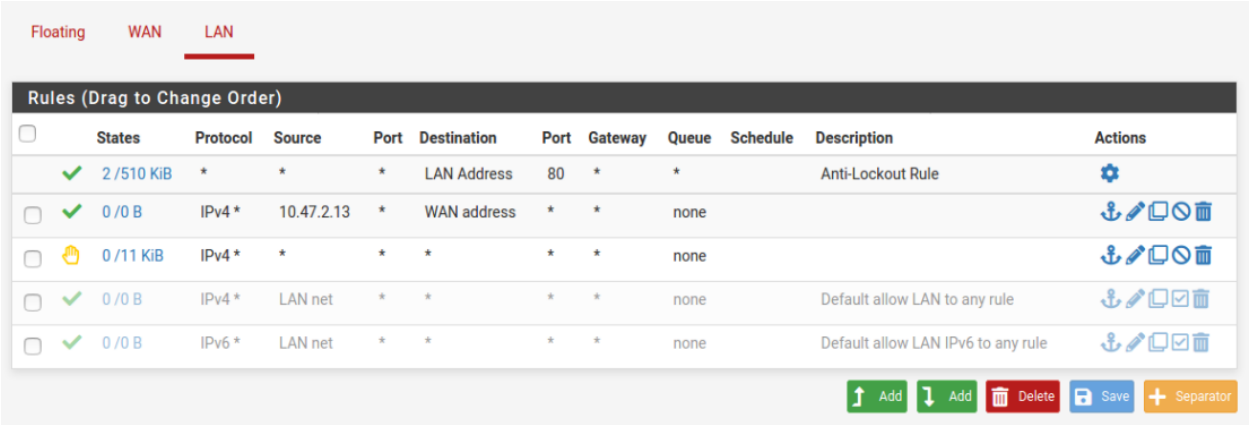


FirewallB WAN final ruleset. This one is similar to the FirewallA WAN ruleset in that we decided to only allow TCP, UDP and ICMP traffic. The only difference is the first rule, which is blocking any SSHrequest from 10.47.3.6, where our clients traffic will come from, to 10.47.2.13, our server.



Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

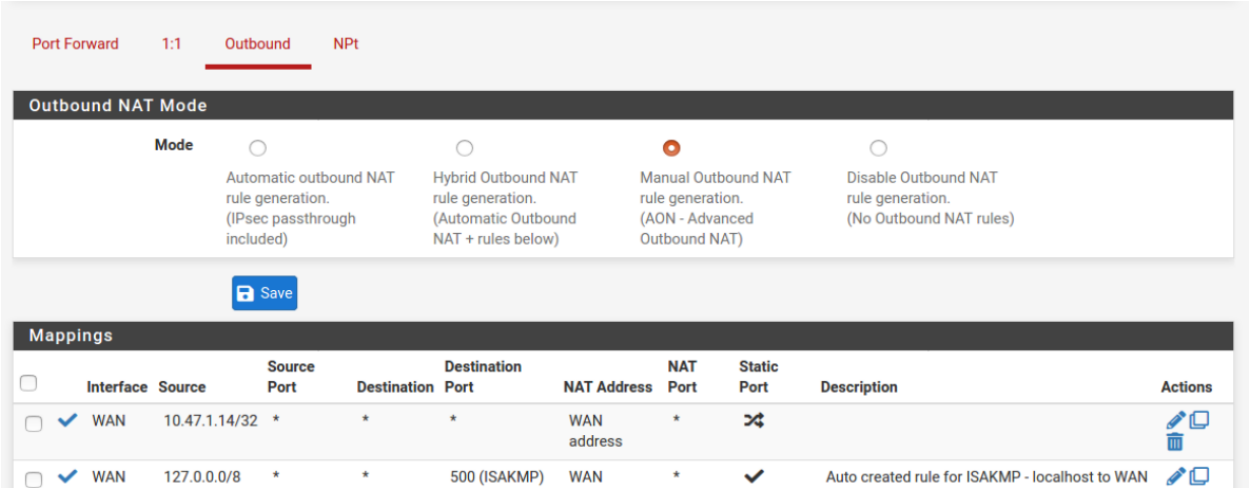
FirewallB LAN final ruleset. We started by turning off the default rules. We don't want the server going out and doing anything without being solicited. The first rule is required by pfsense. The second rule is our "only respond to solicited traffic" rule. Third is to reject any other traffic. Currently this is ok because this subnet only contains the server.



Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

Task3

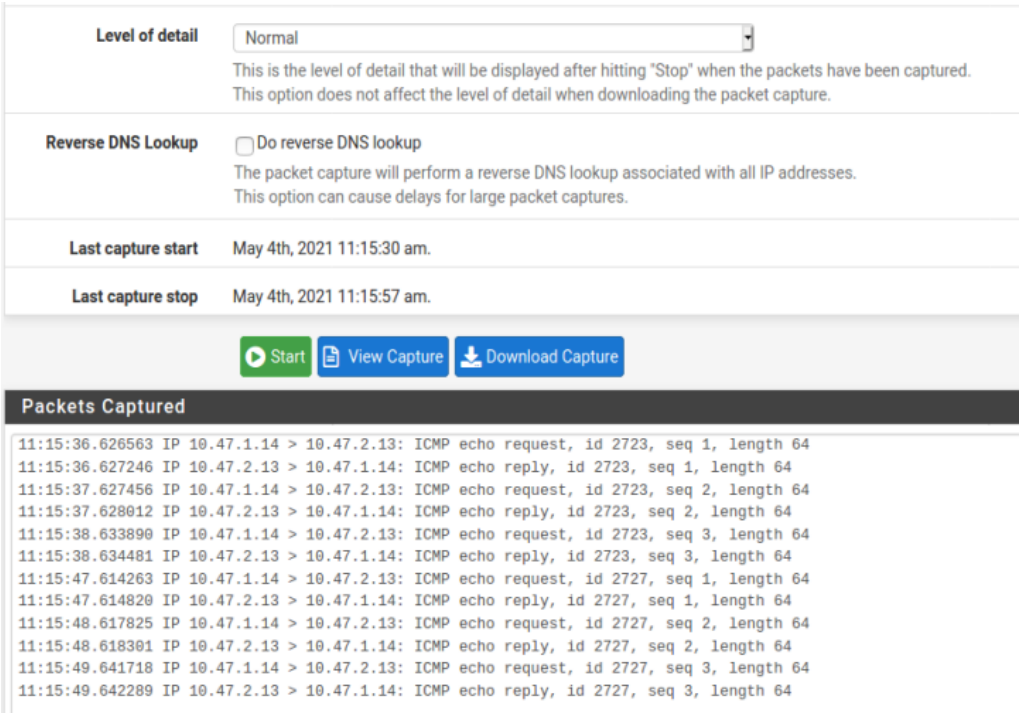
To configure our client to use the external IP of firewallA, we create the following NAT rule.



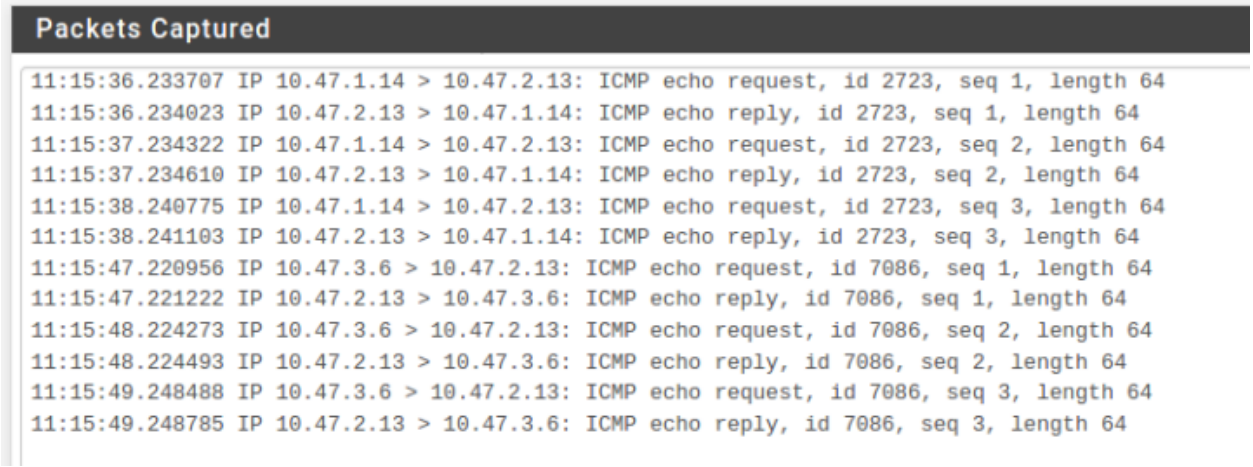
Any Outbound connections will be passed through this list of NAT rules. We decided to force any packets from source 10.47.1.14 to be translated to the WAN interface IP address, 10.47.3.6.

To test this, we do 4 packet traces (one at each interface). During the ICMP-only packet trace, we first do a 3 ping test from our client (10.47.1.14) to our web server (10.47.2.13) with Outbound NAT rules disabled. We then turn on Manual Outbound NAT rules, which uses our previously configured NAT rule, and do another 3 ping test. Here are the results of the LAN and WAN interface on firewallA. The LAN and WAN are identical since they

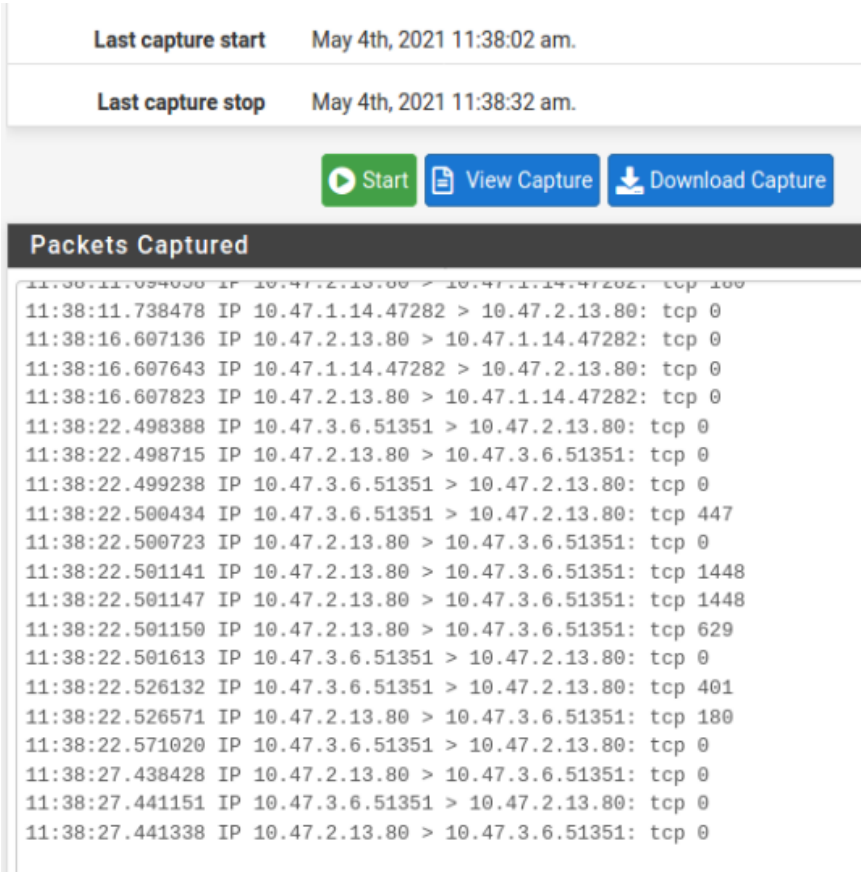
As packets pass out of the WAN interface, the WAN IP address will take the place of 10.47.1.14.



On both interfaces for FirewallB, we can see the difference. Our second set of Pings has been modified to show 10.47.3.6 instead of 10.47.1.14 for our source IP. The first 6 ICMp echo messages are with no Outbound NAT rules and the second three are the pings just after using the manual Outbound NAT rules

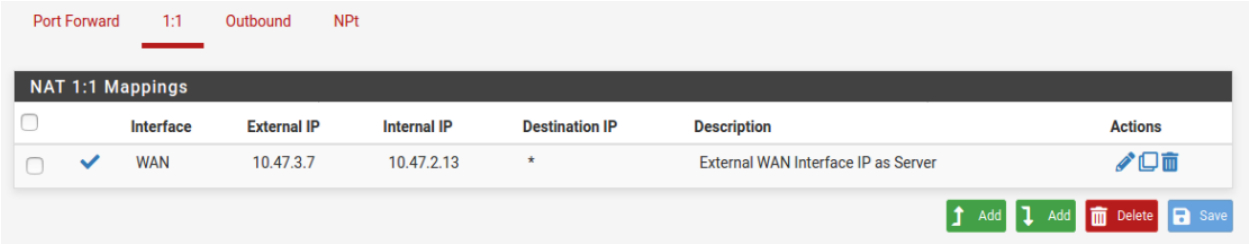


We decided to test this again by connecting to the web server. We ran a packet trace looking for 10.47.2.13 and port 80. We turned off the Outbound NAT rules for the first hard refresh, and then turned on Manual Outbound NAT rules for the second refresh. (*By hard refresh we mean we hit CTRL+R. Without that we might get that cached page*).

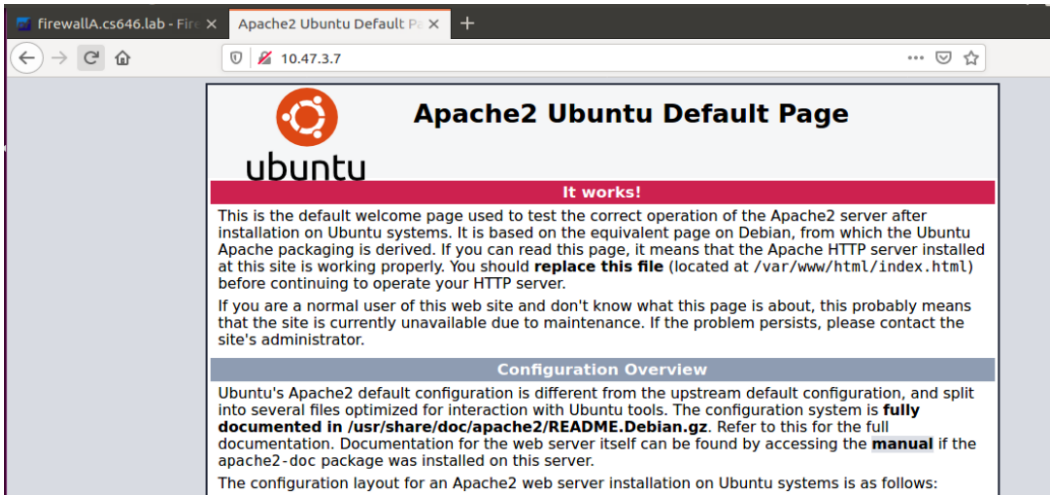


Our packet capture perfectly shows us our Outbound Rules in action. The first refresh before the 11:38:20 used 10.47.1.14 as our source IP address. After turning on the Outbound NAT Rules, we then see the all of our connection after 11:38:20 are using 10.47.3.6 instead of 10.47.1.14.

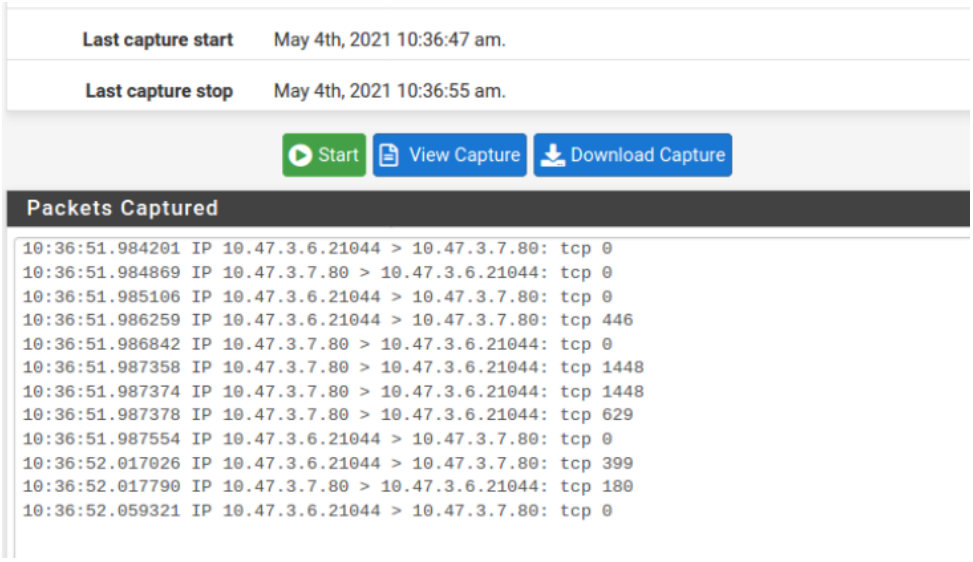
Now that we completed the outbound NAT rules for our client, we need to publish the server to an outside interface. To achieve this, ee configured a 1:1 NAT rule. All this NAT rule does is tell the firewall to push packets to 10.47.3.7 to 10.47.2.13.



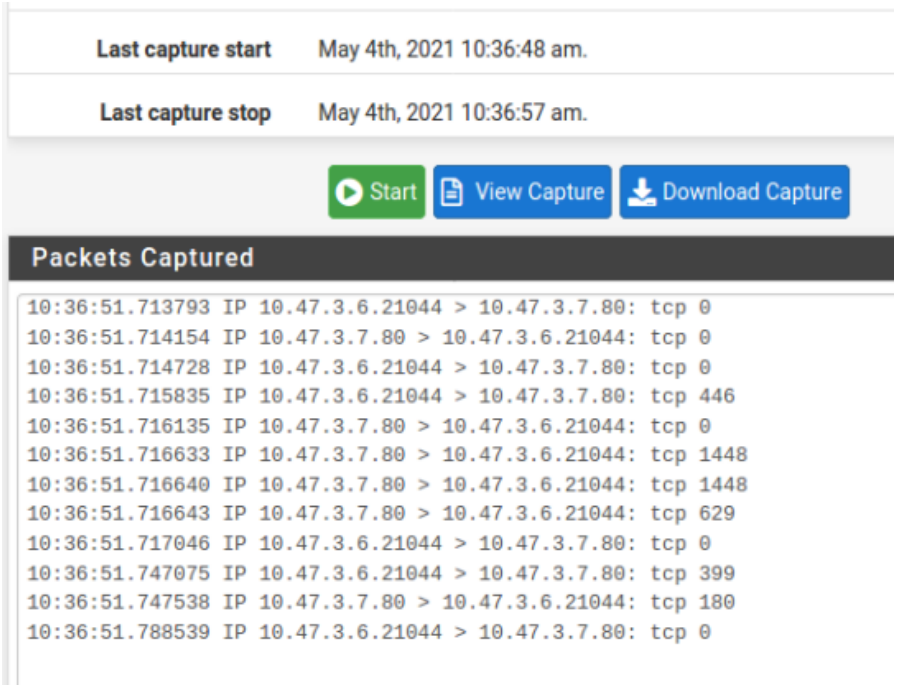
After typing in 10.47.3.7 into the browser address bar on the client, we see the apache webpage, which is what we expected. The IP address has been properly translated to the internal IP after reaching the firewall



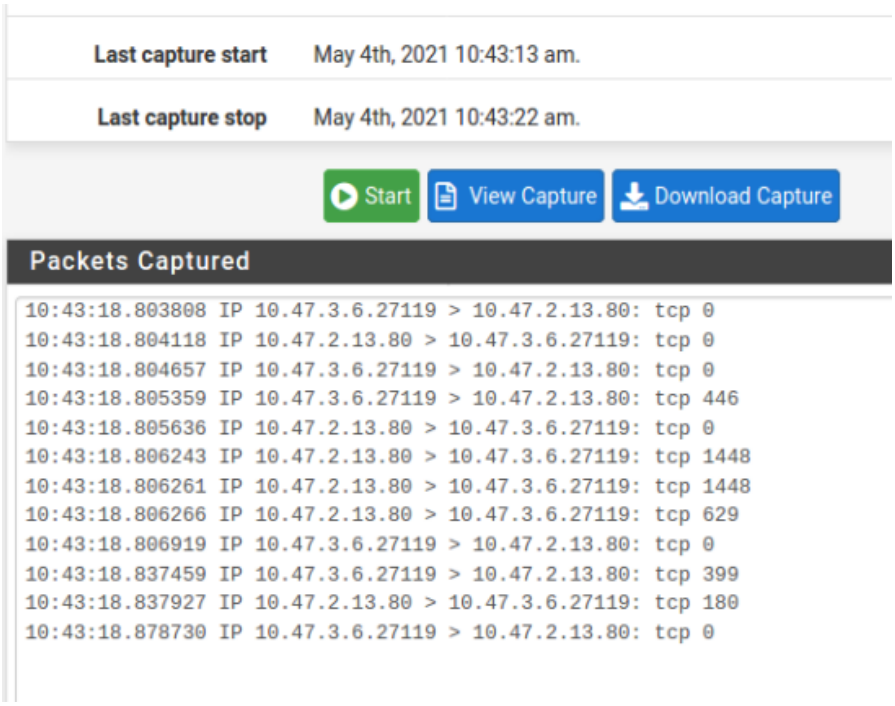
We prove this by doing two packet captures, one at each WAN interface. Unsurprisingly, the packet captures are identical. We performed a single hard refresh of the webpage on 10.47.3.7 after starting the packet capture. This is the packet capture on firewallA's WAN interface.



As you can see, our 10.47.3.6 firewall is communicating directly with 10.47.3.7 on port 80. It is using the firewallB's IP with port 80 to communicate with the web server. In earlier tasks, we had seen that pings to 10.47.2.13 were going straight to the internal IP directly, instead of stopping at 10.47.3.6 like we see here.Below is our identical packet trace on firewallB's WAN.



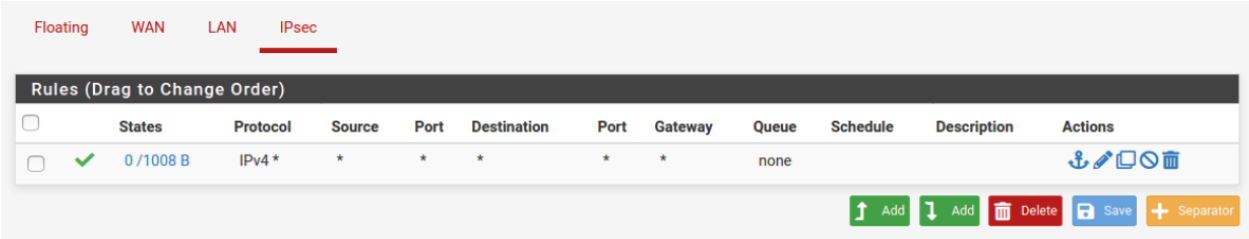
And now we did one final packet trace, running the same test again. This time on the LAN interface on firewallB. We expect to see 10.47.3.6 communicating with 10.47.2.13. This is because Network Address Translation has already occurred. The 10.47.3.7 has been converted to 10.47.2.13 and is communicating with the web server.


Last capture start
May 4th, 2021 10:43:13 am.
Last capture stop
May 4th, 2021 10:43:22 am.
Start
View Capture
Download Capture
Packets Captured
10:43:18.803808 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 0
10:43:18.804118 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 0
10:43:18.804657 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 0
10:43:18.805359 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 446
10:43:18.805636 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 0
10:43:18.806243 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 1448
10:43:18.806261 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 1448
10:43:18.806266 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 629
10:43:18.806919 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 0
10:43:18.837459 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 399
10:43:18.837927 IP 10.47.2.13.80 > 10.47.3.6.27119: tcp 180
10:43:18.878730 IP 10.47.3.6.27119 > 10.47.2.13.80: tcp 0

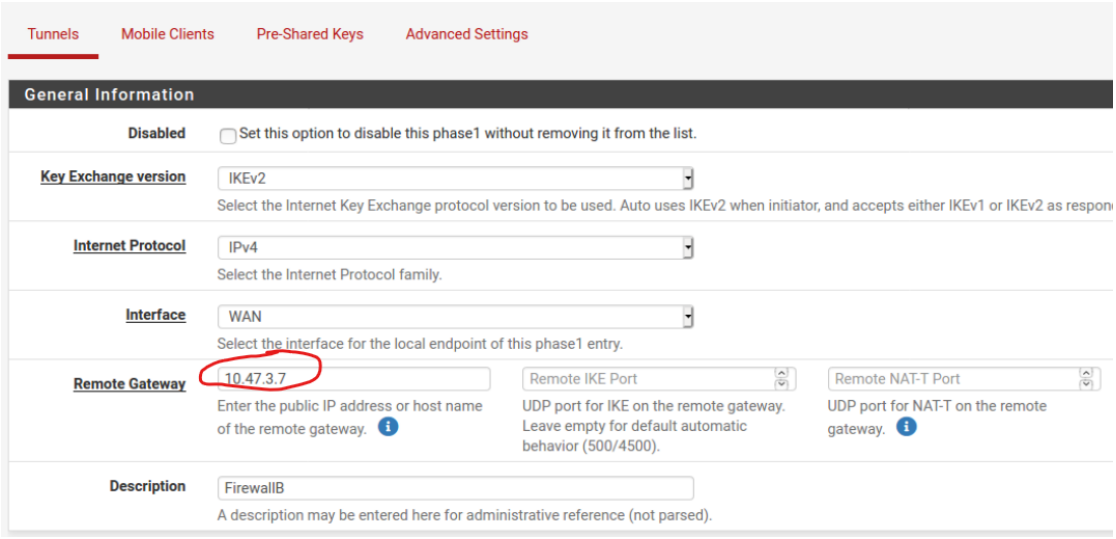
Raphael Archer, Ryan Rozanitis
CS 646 Project 3 Deliverable

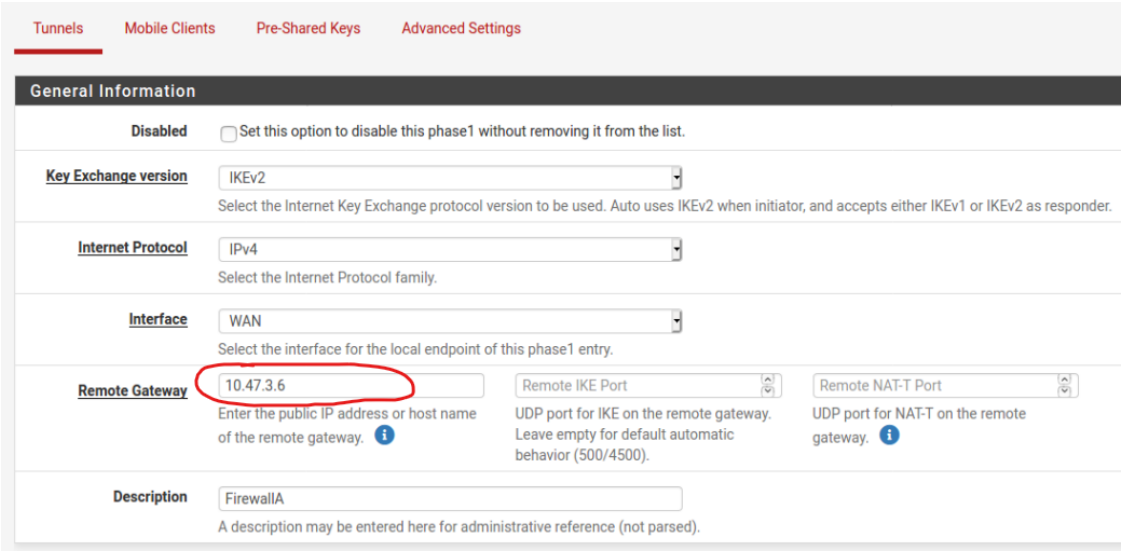
Task4

In order to create the VPN tunnel with IPSEC, we first had to make sure that our firewall rules were set properly. We cleared most of the firewall rules from previous tasks so that all traffic was allowed to pass through. We then also added a new firewall rule to the newly turned on IPSEC firewall rule area that also allowed any IPV4 traffic to pass through.



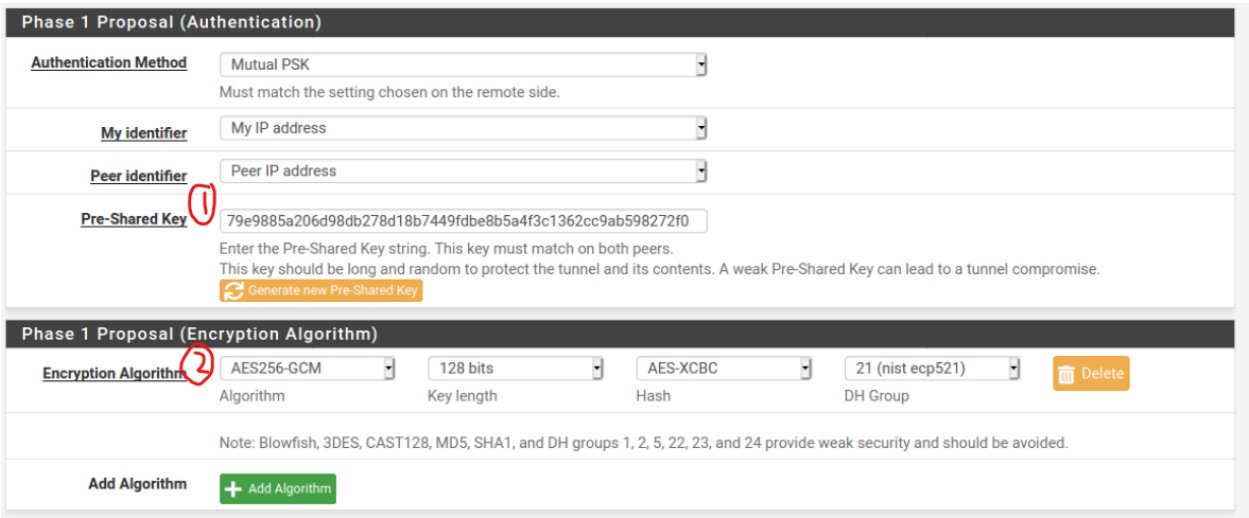
Next we began what is called phase 1. Here we set up the remote gateway, description, internet protocol, encryption algorithm and pre-shared key. For firewallA , our remote gateway as 10.47.3.7. This is telling IPSEC that the other endpoint of our tunnel is 10.47.3.7. We similarly set up the reverse remote gateway for firewallB, 10.47.3.6.



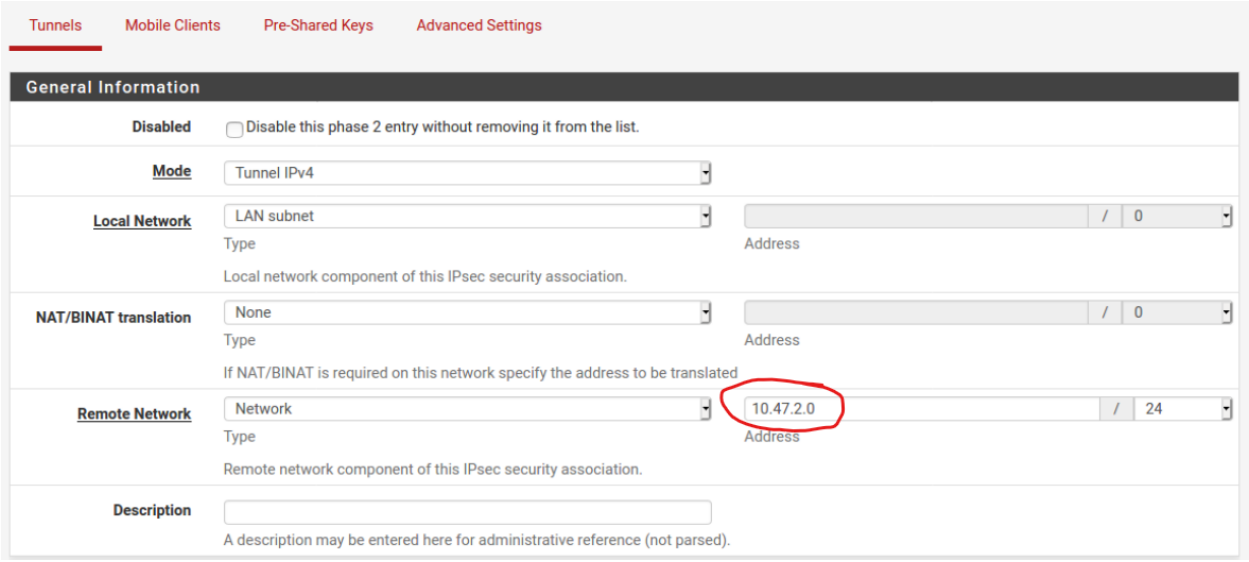


We then configured our pre-shared key and encryption algorithm for phase 1. Near bubble 1 in the picture below, we generated a pre-shared key and copied it over to the other firewall as well. We wanted this to be as secure as possible as per our task 4 requirements, so generating a random key is best practice.

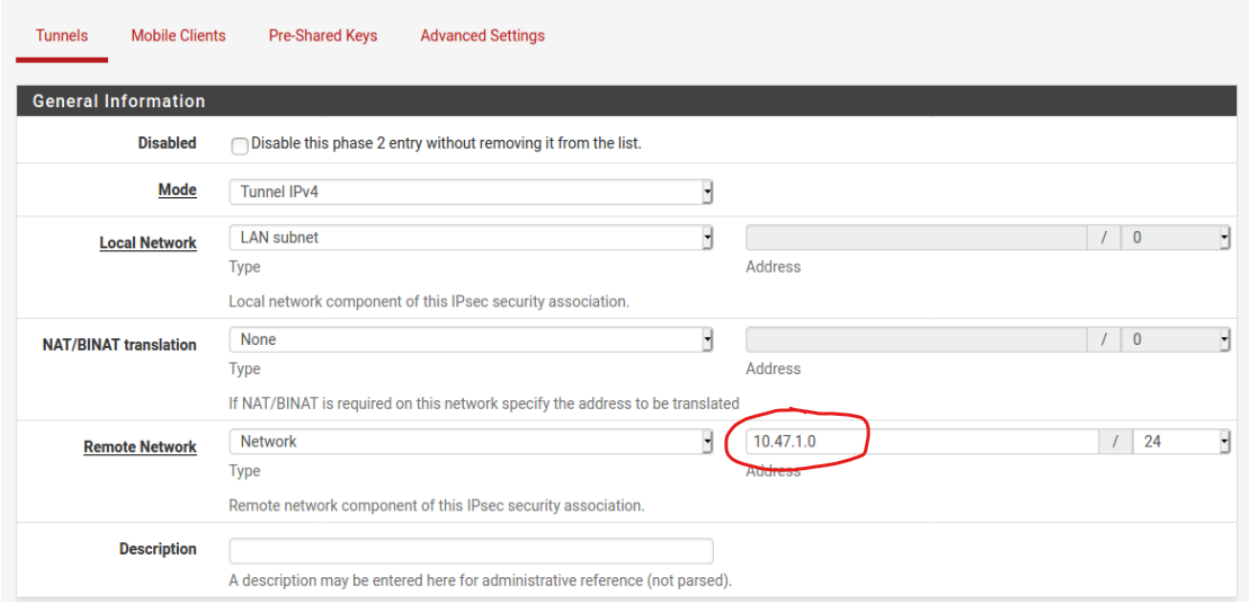
For the Encryption Algorithm, we had to do a bit of research. Pfsense's configuration documentation told us to use at least AES128-GCM with AES-XCBC and DH group 14 or higher. We ended up deciding the AES256-GCMis more secure and stuck with AES-XCBC as our hash. Our key length remained 128 bits, which is the maximum size. We then did a bit more independent research to determine what the best DH group was. Originally we thought DH group 32 would be the best, or ed25519, but enough articles pushed us to use DH group 21 as the standard to use with AES256-GCM while using a 128 bit key.



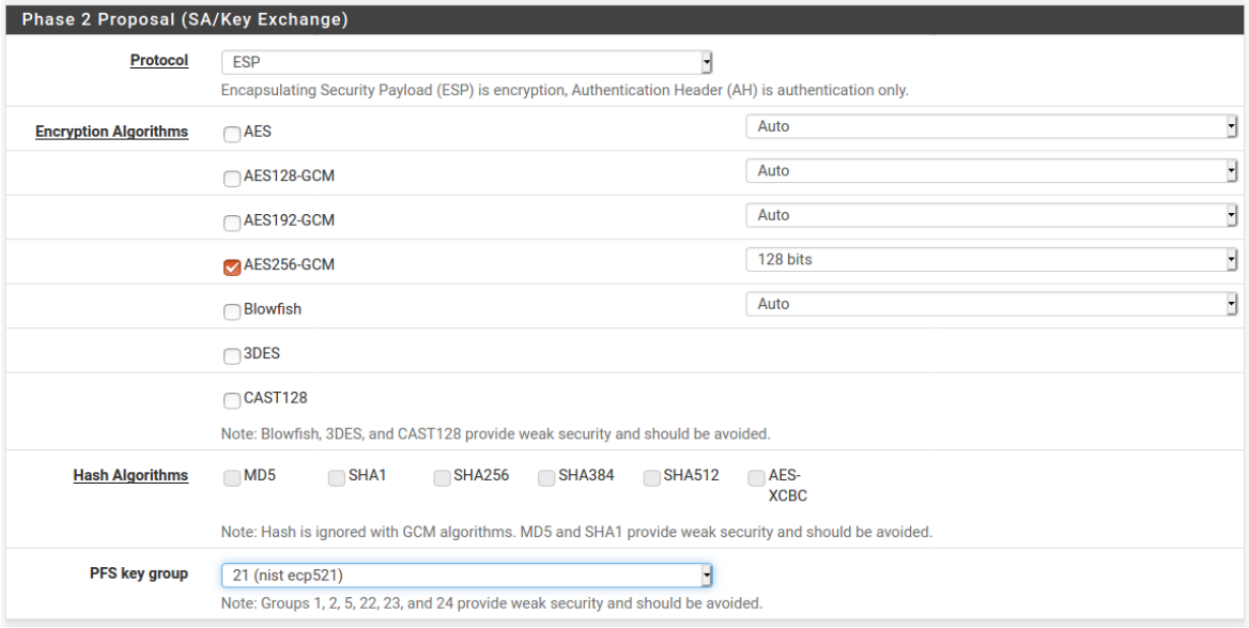
Next we had to add our phase 2 for IPSEC. The important distinction between the two firewalls at this stage is the remote network address. For FirewallA, the remote network is 10.47.2.0/24



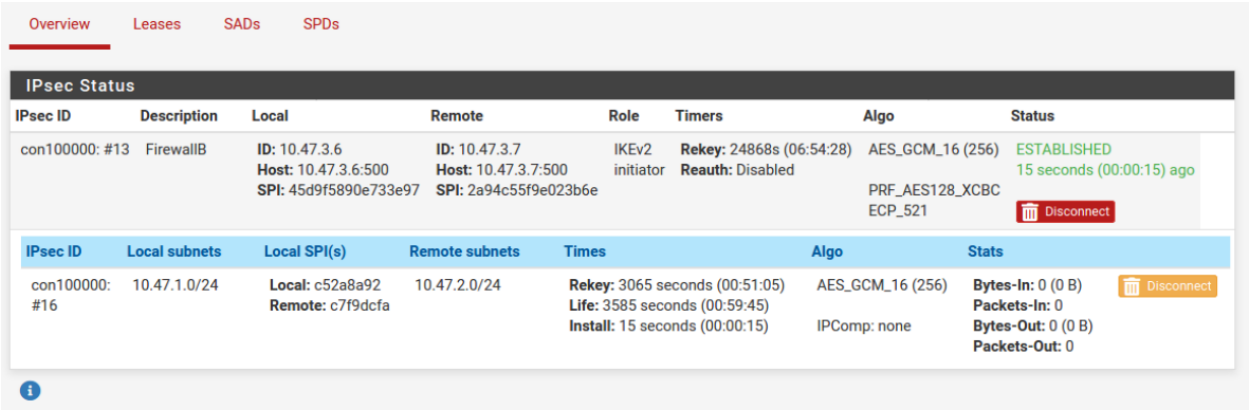
For FirewallB, the remote network is 10.47.1.0/24.

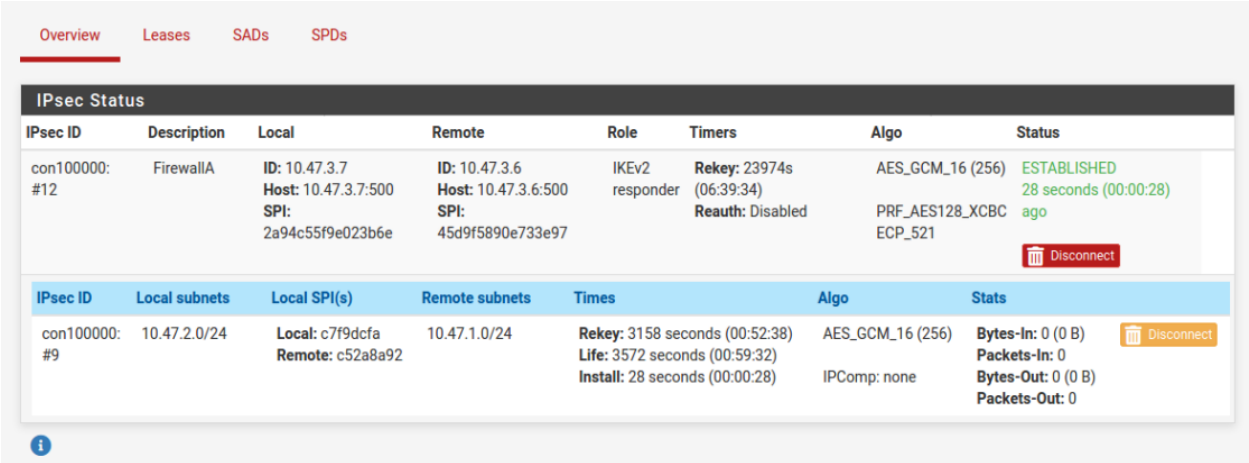


Like phase 1, the encryption at phase 2 is the same for both firewalls. We use AES256-GCM with 128 bits and Key group 21.

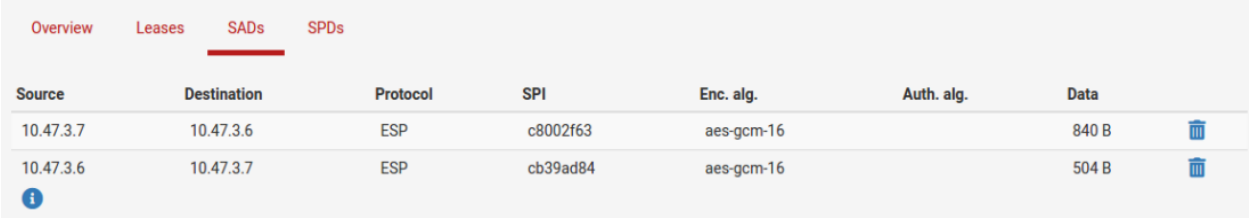


After this, we save both the phase 2 in the firewalls. We can then test and see if they connect by clicking the connect button.





We quickly look at SAD and SPD. SADs are tables that contain the information for a Security Association. SPDs tell what packets IPSEC should apply to. In the SPDs picture, you can see that it specifies the direction. This is telling IPSEC the direction the packets are flowing when IPSEC should be keeping track of them. We can also see the tunnel endpoints are flip flopped in the pictures, which makes sense. Our Firewalls are mirroring each other as endpoints in the VPN tunnel.



Overview Leases SADs SPDs

Source	Destination	Direction	Protocol	Tunnel endpoints
10.47.2.0/24	10.47.1.0/24	◄ Inbound	ESP	10.47.3.7 -> 10.47.3.6
10.47.1.0/24	10.47.2.0/24	► Outbound	ESP	10.47.3.6 -> 10.47.3.7

Overview Leases SADs SPDs

Source	Destination	Direction	Protocol	Tunnel endpoints
10.47.1.0/24	10.47.2.0/24	◄ Inbound	ESP	10.47.3.6 -> 10.47.3.7
10.47.2.0/24	10.47.1.0/24	► Outbound	ESP	10.47.3.7 -> 10.47.3.6

After completing a quick SSH test, we see the packets in and out change. This proves to us that the packets are moving through the tunnel.

Overview Leases SADs SPDs

IPsec Status

IPsec ID	Description	Local	Remote	Role	Timers	Algo	Status
con100000: #13	FirewallB	**ID:** 10.47.3.6 **Host:** 10.47.3.6:500 **SPI:** 45d9f5890e733e97	**ID:** 10.47.3.7 **Host:** 10.47.3.7:500 **SPI:** 2a94c55f9e023b6e	IKEv2 initiator	**Rekey:** 24587s (06:49:47) **Reauth:** Disabled	AES_GCM_16 (256) PRF_AES128_XCBC ECP_521	ESTABLISHED 296 seconds (00:04:56) ago Disconnect

IPsec ID	Local subnets	Local SPI(s)	Remote subnets	Times	Algo	Stats	
con100000: #16	10.47.1.0/24	**Local:** c52a8a92 **Remote:** c7f9dcfa	10.47.2.0/24	**Rekey:** 2784 seconds (00:46:24) **Life:** 3304 seconds (00:55:04) **Install:** 296 seconds (00:04:56)	AES_GCM_16 (256) IPComp: none	**Bytes-In:** 4,038 (4 KiB) **Packets-In:** 18 **Bytes-Out:** 4,424 (4 KiB) **Packets-Out:** 20	Disconnect

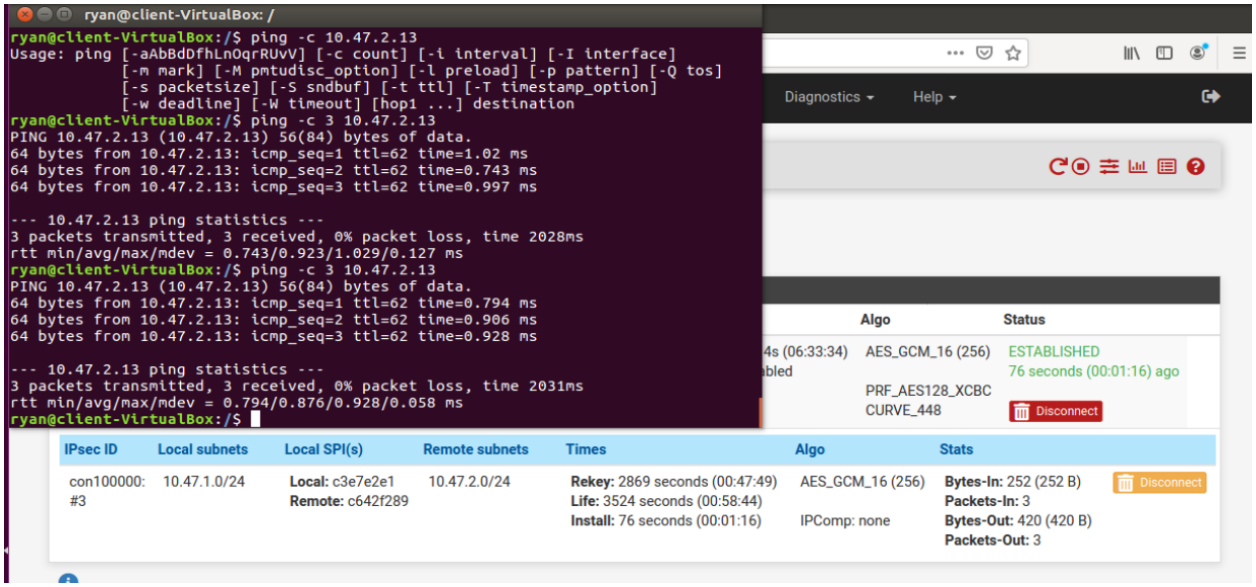
We also confirm this by looking at the logs. On firewallA, we see a request generated, packet sent, and then packet received and response parsed. This is our ssh test we just performed a moment ago.

```
May 4 01:46:25  charon  67921  10[ENC] <con100000|13> generating INFORMATIONAL request 25 [ ]
May 4 01:46:25  charon  67921  10[NET] <con100000|13> sending packet: from 10.47.3.6[500] to 10.47.3.7[500] (57 bytes)
May 4 01:46:25  charon  67921  10[NET] <con100000|13> received packet: from 10.47.3.7[500] to 10.47.3.6[500] (57 bytes)
May 4 01:46:25  charon  67921  10[ENC] <con100000|13> parsed INFORMATIONAL response 25 [ ]
```

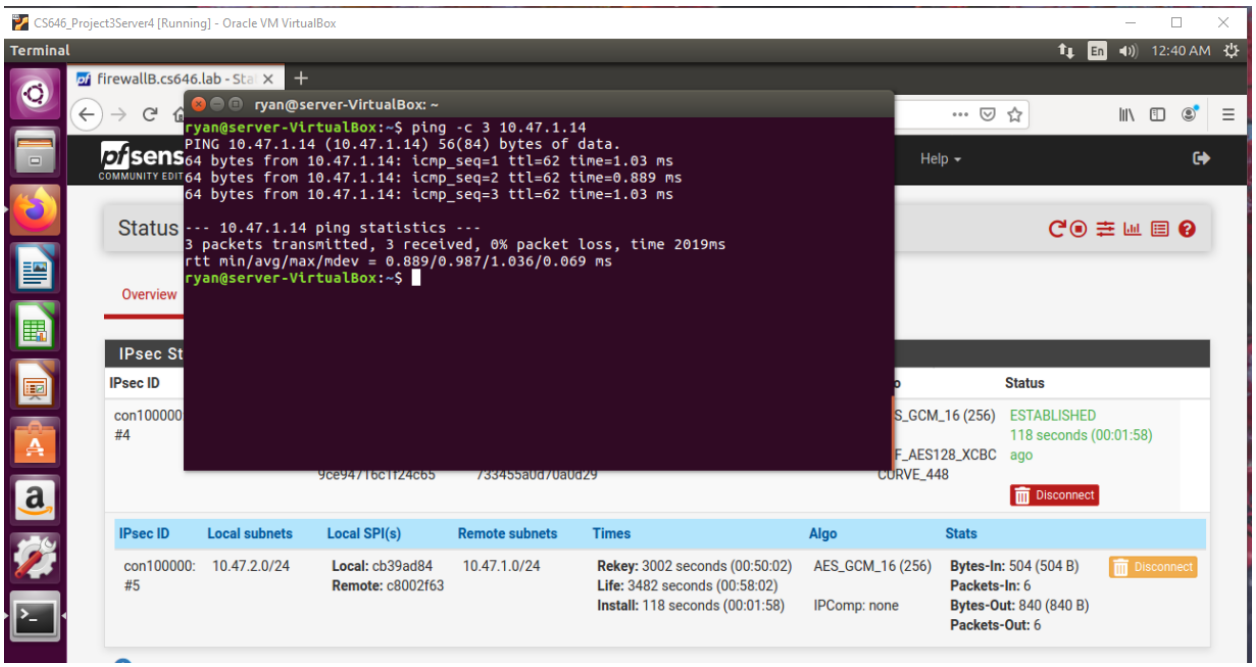
We can also see the logs from firewallB showing a mirror image of the above. First we receive a packet and parse it, then generate a response and send it out.

```
May 4 01:46:34  charon  45056  07[NET] <con100000|12> received packet: from 10.47.3.6[500] to 10.47.3.7[500] (57 bytes)
May 4 01:46:34  charon  45056  07[ENC] <con100000|12> parsed INFORMATIONAL request 26 [ ]
May 4 01:46:34  charon  45056  07[ENC] <con100000|12> generating INFORMATIONAL response 26 [ ]
May 4 01:46:34  charon  45056  07[NET] <con100000|12> sending packet: from 10.47.3.7[500] to 10.47.3.6[500] (57 bytes)
```

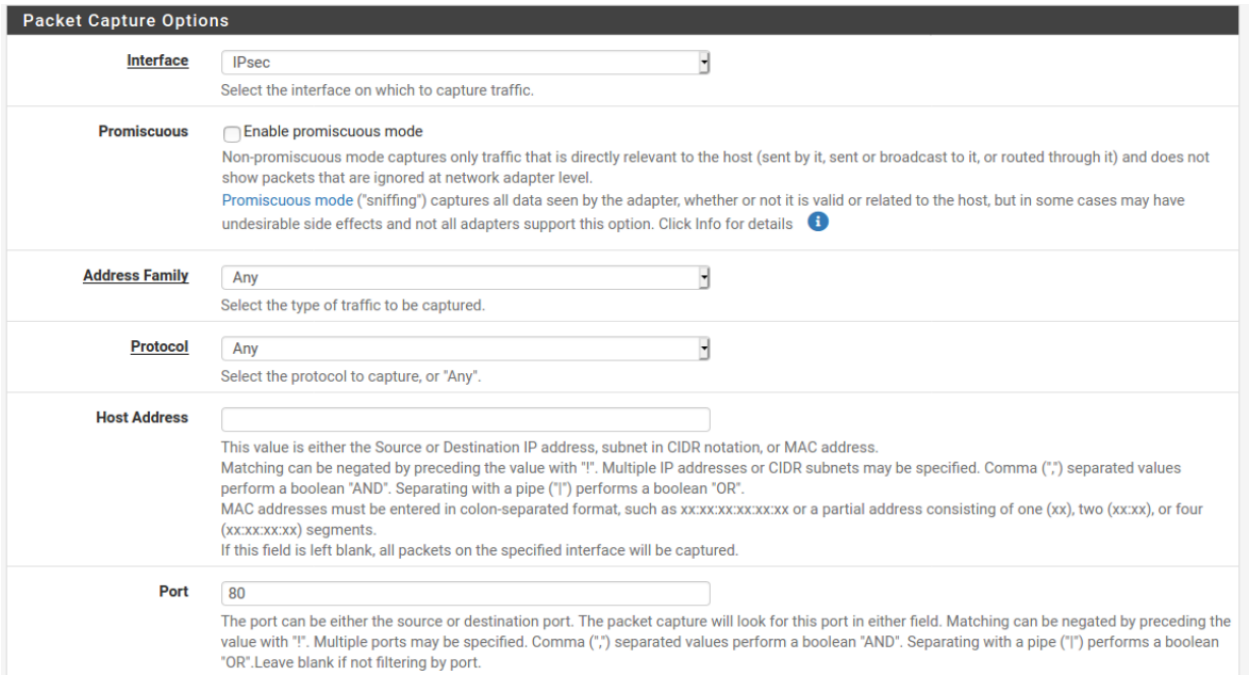
We restart the connection, to get 0 packets in and our to make sure that our connection is working. We then sent 3 pings from 10.47.1.14 to 10.47.2.13. In response, we see 3 packets in and 3 packets out.



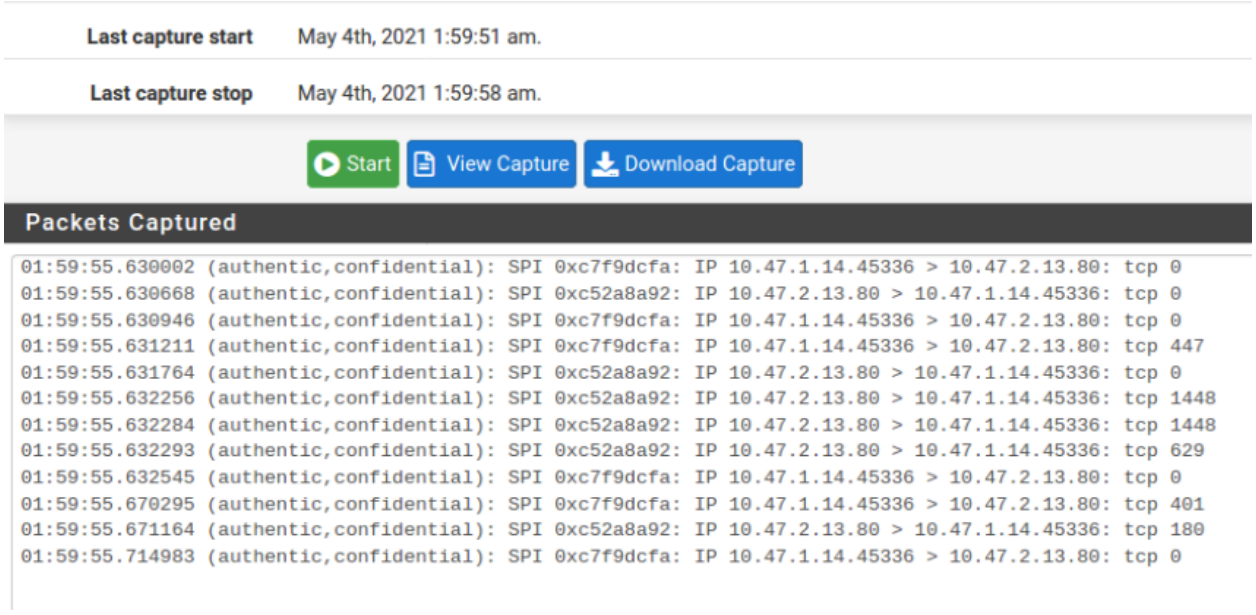
Similarly, we do the reverse on the server. We send 3 pings from 10.47.2.13 to 10.47.1.14. Now we see 6 packets in and out of our VPN tunnel.



Lastly, we do one final packet capture using the apache web server as our test. We set up the packet capture on firewallA and firewallB to only look for packets looking for port 80. We also specify to look on our IPSEC interface.



The results below prove our IPSEC VPN tunnel is active and functional. First we have the packet capture on firewallA. We can see the client and the web server communicating over HTTP. We also see the client initiated the connection, since the first packet captured is 10.47.1.14.45336 > 10.47.2.13.80. The (authentic, confidential) also reaffirms that this is a VPN connection, since we had to authenticate using the pre-shared key and encrypt using AES156-GCM with all of our other parameters.



We stopped firewallB's packet capture a second later to show the difference, but other than that last capture stop time everything else is identical, which makes sense since both firewalls are communicating through the same VPN tunnel.