ATTACKS ON TCP/IP PROTOCOLS-PROJECT 1

Deepanshu Lulla

CS6740 Northeastern University

Contents

Introduction

TCP over IP is the backbone of communication over Internet. However most of the protocols were designed not keeping in mind the possibility of their being attacked. Most of the attacks are based on this blind trust between protocols. IP spoofing combined with MAC spoofing formed the basis of most of the attacks. There was no way for the victim to check whether the spoofed packet came actually from the intended sender. As a result often malicious packets enter the network and the attack becomes prudent if the attacker and the victim are in the same network.

The following attacks were conducted on TCP/IP protocol and most of them were quite successful.

Seq. no	Attacks	Successful?	Type of Attack tested
1	ARP cache poisoning	Yes	Denial of service, Man in The middle
2	ICMP Redirect Attack	Yes	Man in the Middle
3	SYN Flooding Attack	Yes	Denial of service
4	TCP RST Attacks on Telnet and SSH Connections	Yes	Denial of service
5	TCP RST Attacks on Video Streaming Applications	Yes	Denial of service
6	ICMP Blind Connection-Reset and Source-Quench Attacks	No	Denial of service
7	TCP Session Hijacking	Yes	Man in the Middle, Denial of Service

However the basis of these attacks to become successful was that attacker and victim were in the same network. Some of the attacks would not have been successful otherwise. Combination of Netwox (with C) and python (with scapy) was done.

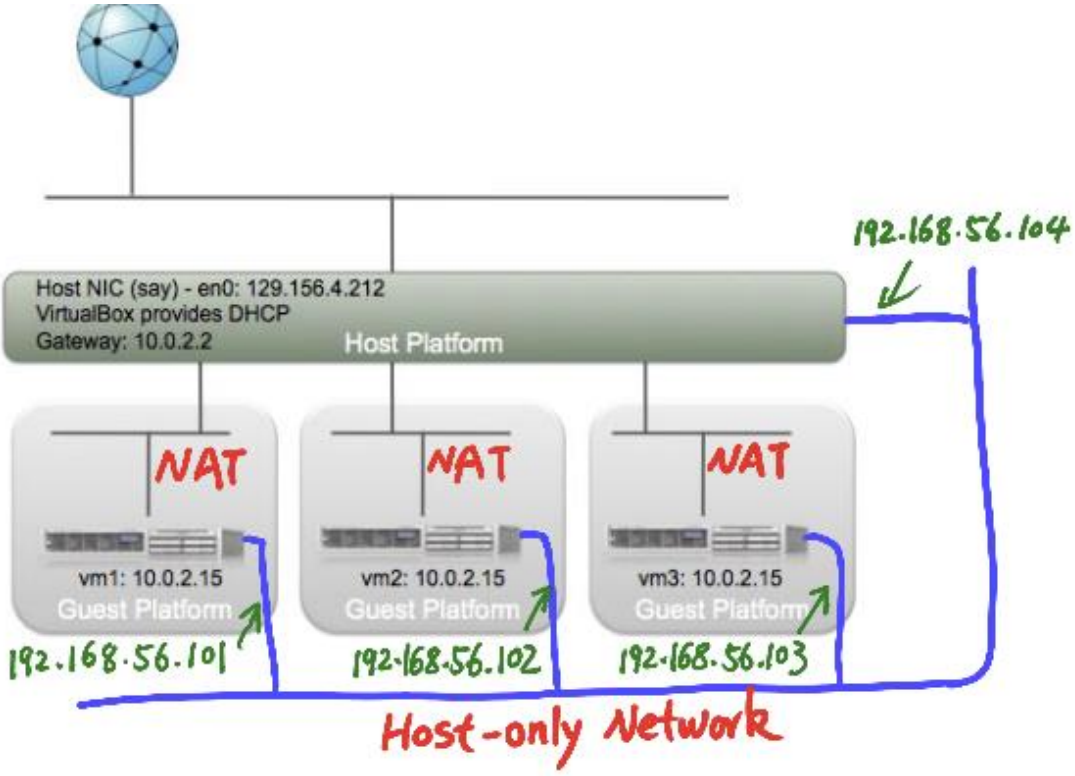
In a nutshell there were three primary reasons for attacks

1) The dire desire for protocols to trust each other
2) The need to maintain state
3) The need to maintain cache table

Most OS versions have the ability to protect attacks and that is one of the reasons some of the attacks were not successful to that extent. However most other attacks were, which means vulnerability still exists to most of the attacks and the system can't be termed as secure.

Configuring the Virtual machines

The virtual machine was setup as per instructions given with the project description.



The VMs have two adapters

-A NAT adapter

-A host only adapter.

All attacks were done on host only network.

System Discovery and Fingerprinting

It is important for the attacker to be first familiar with the IP addresses and MAC addresses of all the hosts within a network.

Name	IP	MAC	Interface
Router	192.168.56.1	0A:00:27:00:00:00	
DHCP Server	192.168.56.100	08:00:27:67:88:BA	
Attacker	192.168.56.101	08:00:27:DF:17:8C	Eth13
Victim	192.168.56.102	08:00:27:1A:C1:21	Eth15
Observer(victim 2 in some cases)	192.168.56.103	08:00:27:36:72:04	Eth15

It was done by active fingerprinting by sending an ARP query to all hosts in a network and then waiting for an ARP reply.

```
[01/30/2016 17:36] root@ubuntu:/home/seed/tcpAttacks# netwox 5 --ips "192.168.56
.0/24"
192.168.56.1    0A:00:27:00:00:00
192.168.56.100  08:00:27:FC:01:6E
192.168.56.101  08:00:27:DF:17:8C
192.168.56.102  08:00:27:1A:C1:21
192.168.56.103  08:00:27:36:72:04
```

Figure 1: System Discovery using netwox

The sequence number and port number prediction was done using ARP cache poisoning as a man in the middle attack. Then using that as a reference most other attacks were done.

Attack 1: ARP Cache Poisoning

Design

ARP poisoning means to cause a victim to associate a wrong IP –address to MAC address mapping. This usually involved sending fake ARP replies indicating an ARP mapping. Since ARP table caches are short lived, they need to be refreshed and they periodically query each other (to find out if the peers in a subnet are still alive). The victim accepts the fake response from the attacker and adds entry to its arp cache table. There are two motives behind the attack:

1) Denial of Service(using flooded requests and causing an incorrect IP-MAC address mapping)
2) Man in The Middle(Assuming two victims, the attacker can send fake ARP replies to both and mapping their respective IP address to its own MAC address.)

The files used are attached and are called *task1_DOS.c and task1_mitm.py.*

Observations

Denial of service

The aim was to deny service for the victim to one of the hosts (Observer). The result was pretty successful with netwox. Tool 33 was used for the attack. As one can see in figure 2 and 3 the victim was not able to ping observer for a pretty good time, after the attack was stopped causing 90% packet loss.

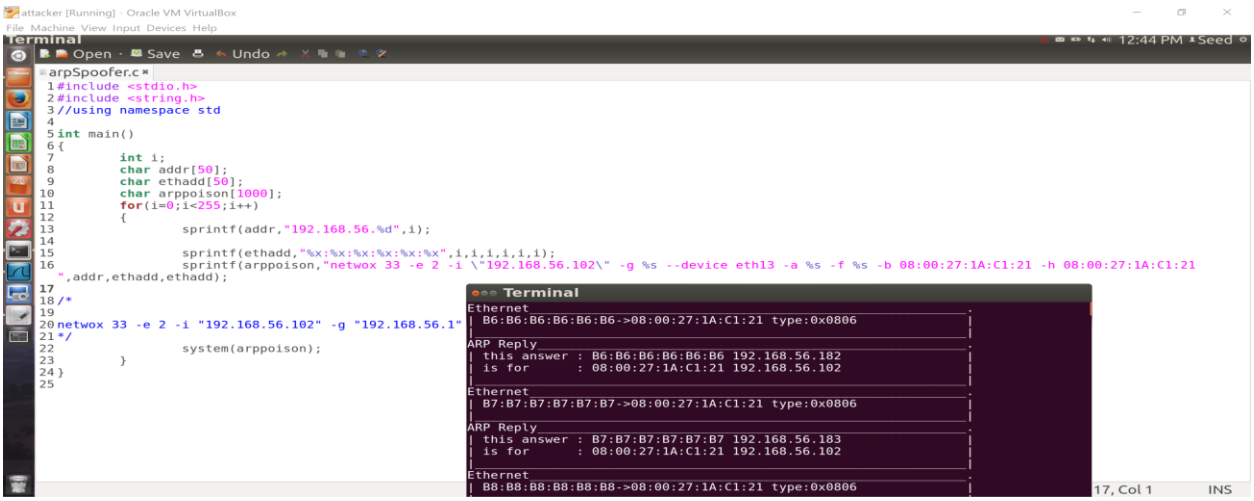


Figure 2: Attacker sending fake replies to the victim

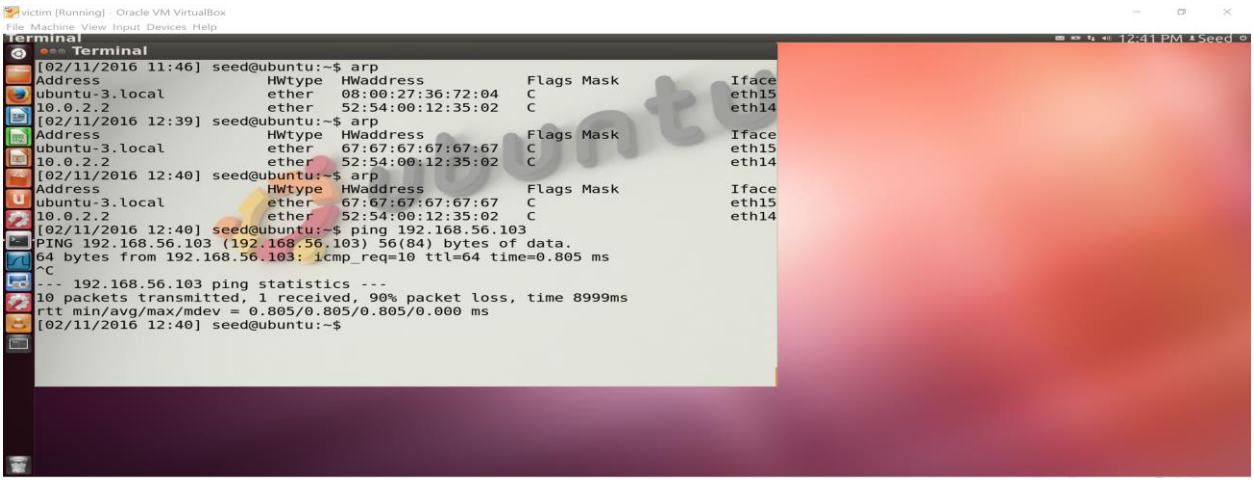


Figure 3: Victim's ARP cache before and after attack. The victim was not able to ping for a significant time, even after the attack was stopped.

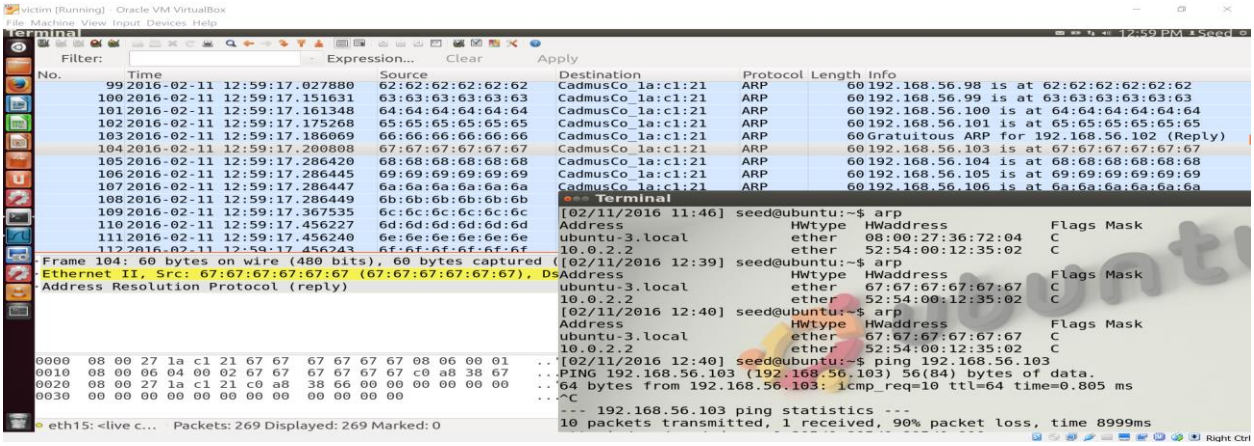


Figure 4: Wireshark snip showing the spoofed ARP reply

Man in the middle

After running the script, the ARP mappings of both observer were changed to attacker's MAC address

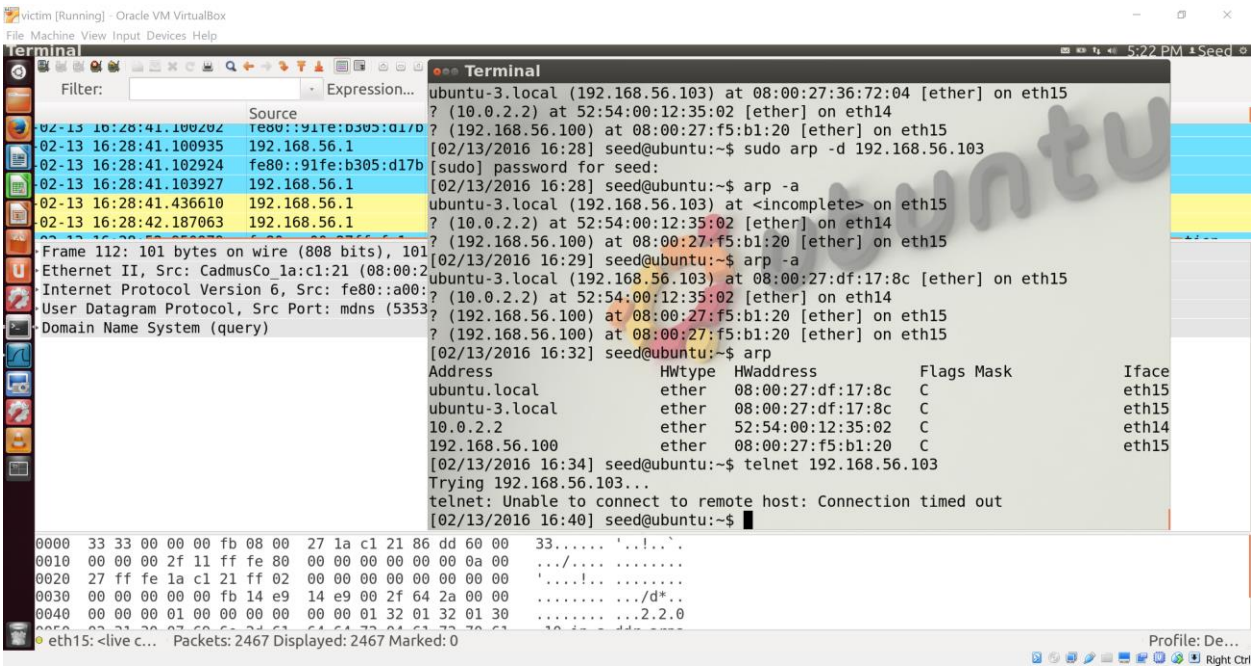


Figure 5: ARP mappings of victim before and after attack

The attacker was able to see packets from both sides

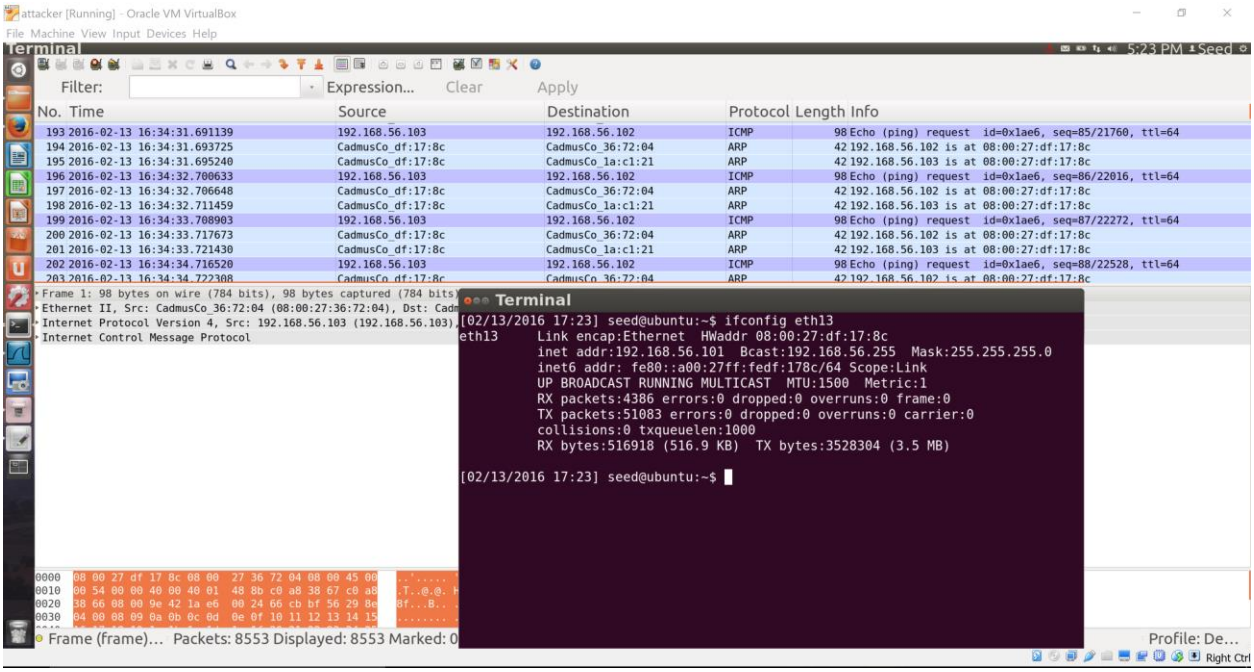


Figure 6: Attacker being able to view packets flowing from both sides

Explanations:

Since ARP is a stateless and connectionless protocol, The ARP reply was easily spoofed and there was an easy Denial of service. The attack could be optimized more and an ARP reply could be sent only when

there is an ARP query which is usually a broadcast reaches the attacker. This can be mitigated by using Port security features in switches like associating a MAC address with a particular port of switch and dropping all packets not from that mac address. Other ways include limiting one MAC address per port.

Attack 2: ICMP Redirect attack

Design

ICMP redirects are an error message which is used by routers to convey to hosts that they need to forward packets from a different gateway.

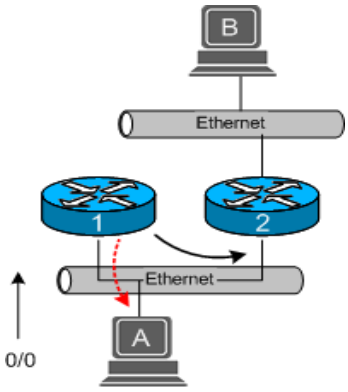


Figure 7: Figure adapted from [1]

However ICMP redirects can be bad, because it can be used to create a Man in the Middle attack. The attacker can forge ICMP redirect packets with IP spoofing and MAC spoofing and ask them to be forwarded through its own IP address, advertising itself as the new gateway.

Observations

The Ubuntu 12.04 has inherent protection for the ICMP redirects and hence the attack doesn't work out of box. However the settings can be disabled just to prove the attack is working. Open /etc/sysctl.conf

$ sudo gedit /etc/sysctl.conf

And edit the following variables as shown in figure 5. Change these parameters in all host machines

After editing, reload the file using

$sudo sysctl –p /etc/sysctl.conf

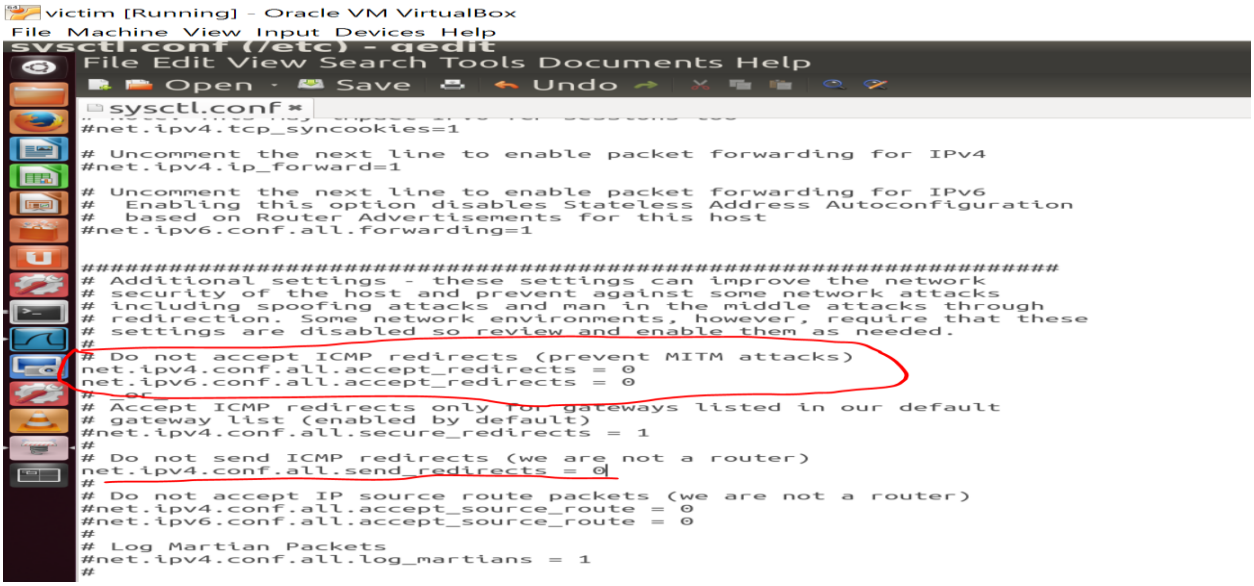


Figure 8: sysctl.conf file

Retrying the attack after that started working and created a successful Man in the Middle attack.

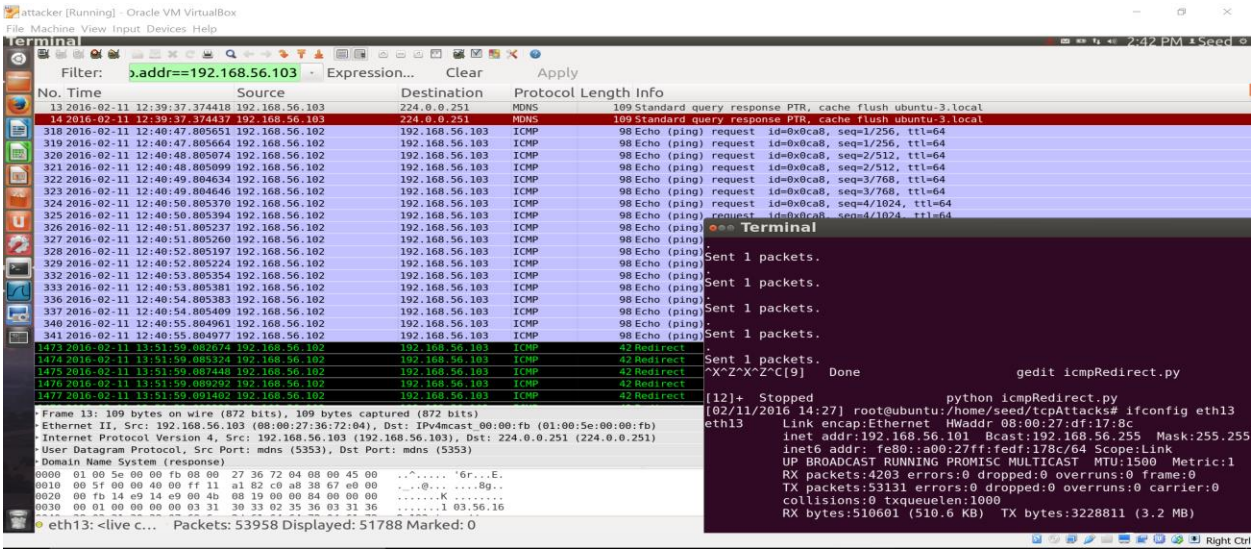


Figure 9: VM of attacker (192.168.56.101) which can successfully see the ICMP packets flowing from 192.168.56.102 to 192.168.56.103

As one can see, from the VM of attacker (as shown in figure 6) can intercept packets flowing from 192.168.56.102(victim) to 192.168.56.103(observer).

The worst part is unless the victim opens wireshark, there was no way for him or her to detect a possible anomaly in the connection.

Pings to other hosts work fine, pings to external network works fine and there is not even a change in the routing table.

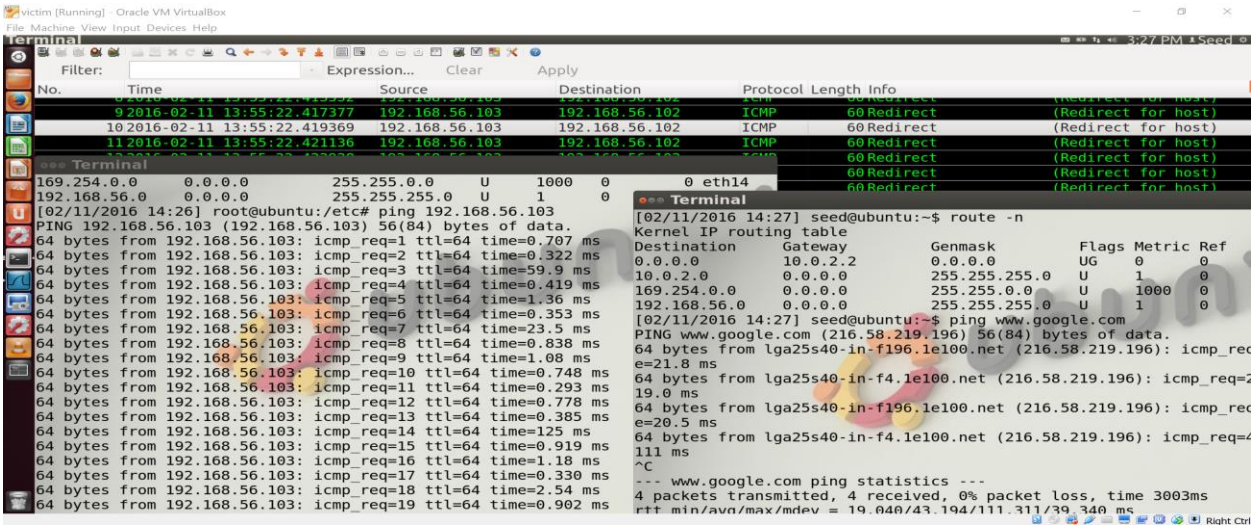


Figure 10: The victim is not able to detect any anomaly in connection unless s/he opens wireshark and deduces an anomaly

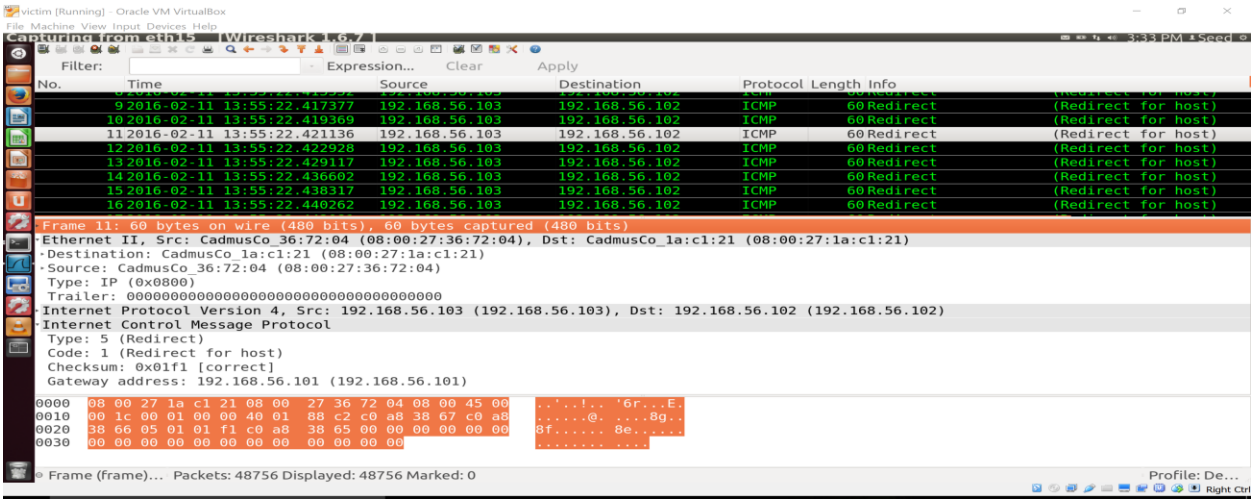


Figure 11: Wireshark analysis of the attack

The file used to attack the victim is attached with the report and is stored as task2.py

Explanation:

As ICMP redirects are meant to be sent by routers, the hosts accept the new gateway information by default and start forwarding packets to them. The protocol was designed with trust and the creators never thought that it could be so easily manipulated. The attack can be mitigated by disabling ICMP redirects for hosts and enabling them for only routers.

Attack 3: SYN flooding attack

Design

SYN flooding attacks exploit the inherent nature of TCP session initiation. The basis of the attack is asymmetric allocation of resources between server and client. The server associates state with the connection while the client doesn't. As a result a particular client can repeatedly send the server lot of SYN requests for which the servers replies with a SYN-ACK and expects an ACK in return to establish connection. However the attacker can exploit this by only sending SYN requests and cause a Denial of Service (DOS) by causing a valid request to get turned down.

I created a webserver on the victim using apache2 server. And the observer here is the honest client who tries to request a webpage while the attacker tries to create a DOS attack.

Observations

This was the result in the absence of an attacker.

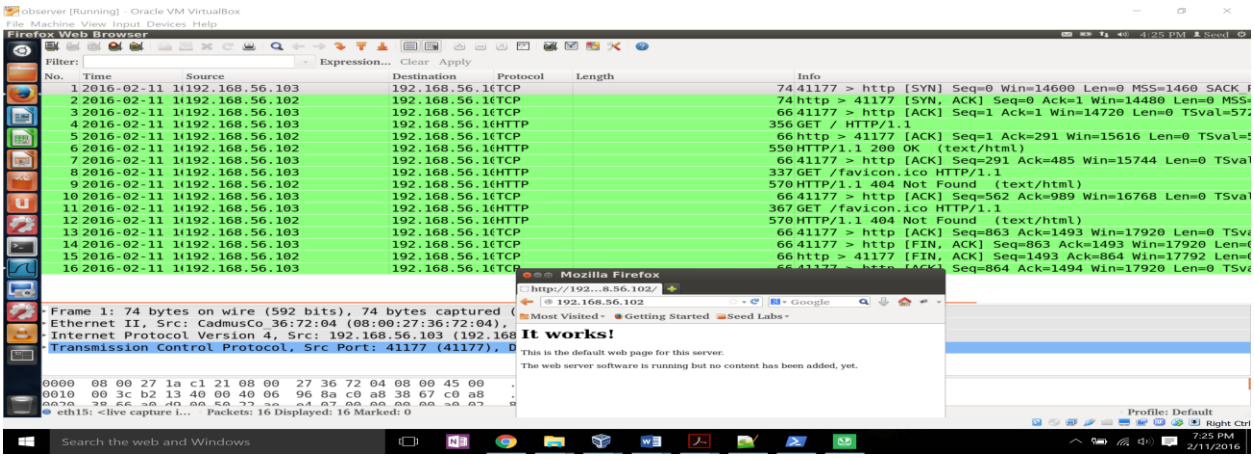


Figure 12: The observer requests for web page and the client successfully delivers the request

Another thing to remember during the attack, is that the web cache (stored in History under Mozilla) needs to be cleared else the server sends a HTTP Not Modified response which asks the client to load from cache. The cache needs to be cleared every time the request is sent.

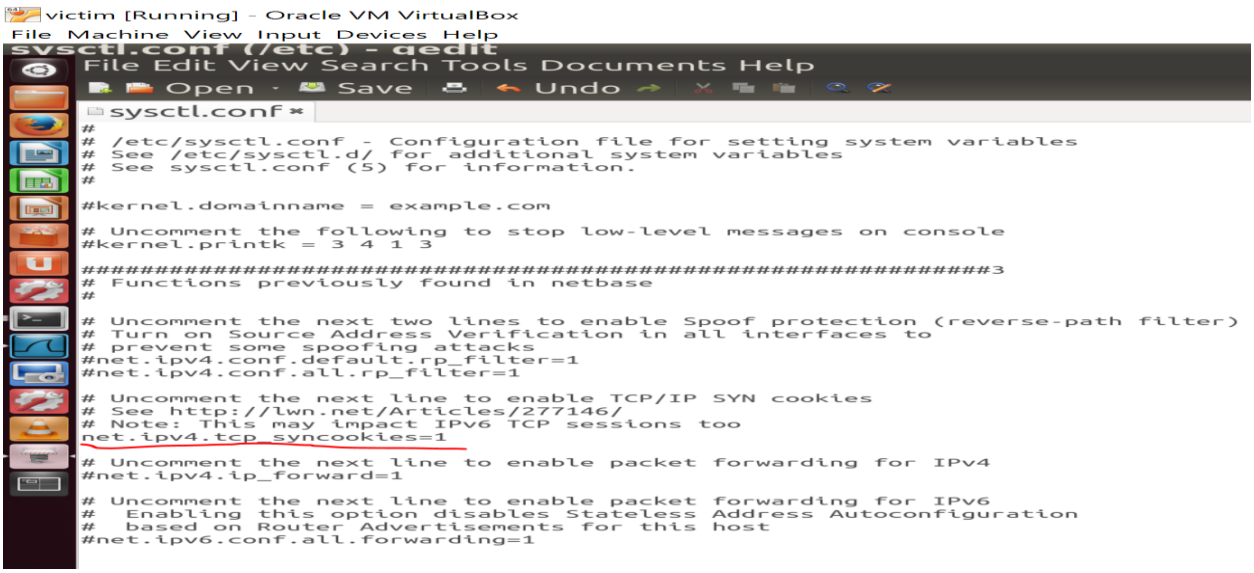


Figure 13: SYN cookies enabled in sysctl.conf

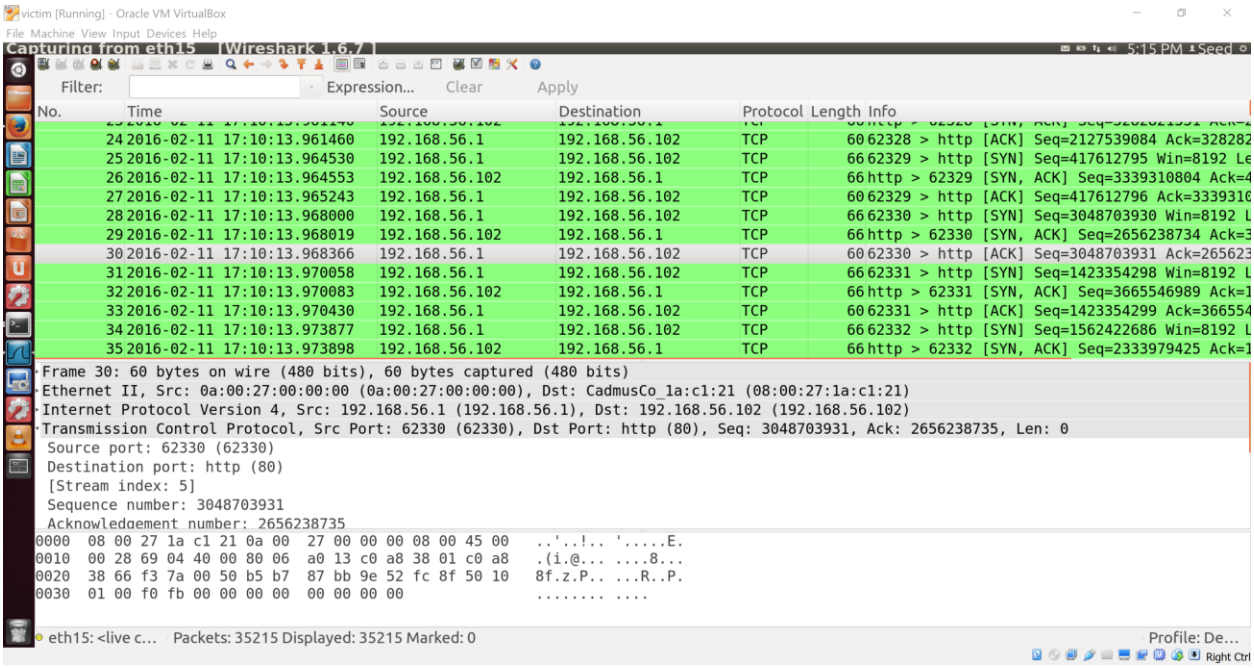


Figure 14: WIreshark snip showing attack being executed. There was no change in the pattern irrespective of syn cookies being enabled or not

Disabling the cookies, trying the same experiment

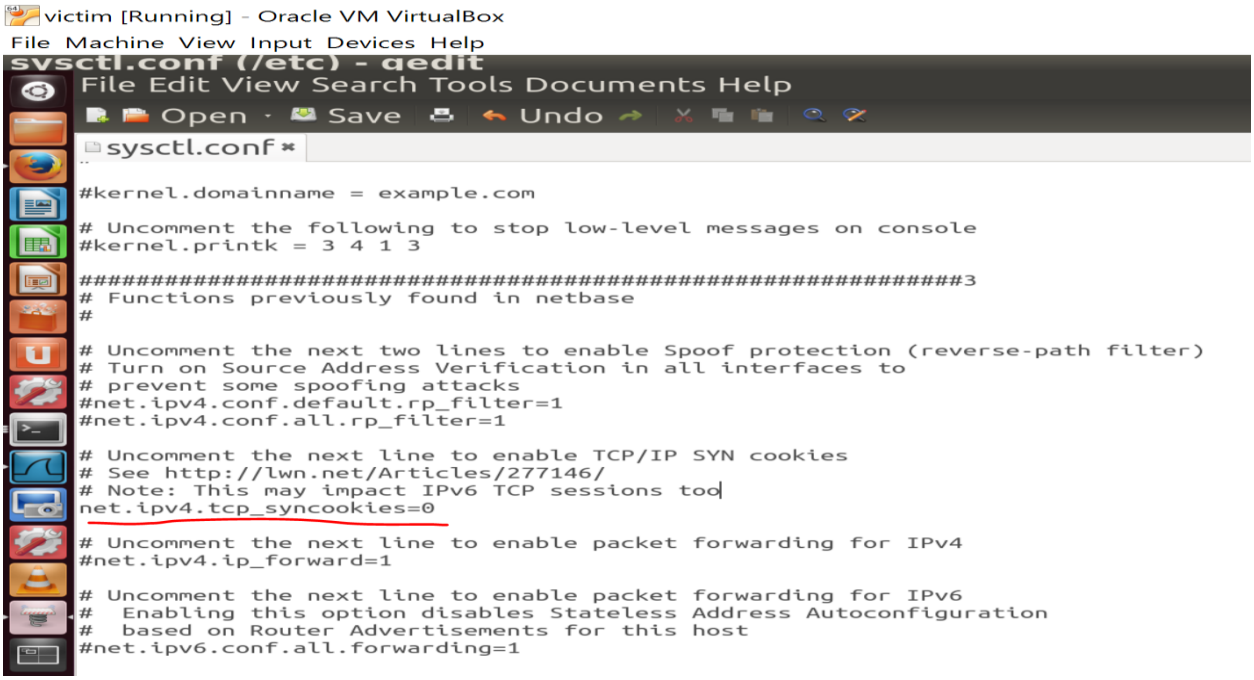


Figure 15: Disabling syn cookies in synctl.conf

We may also need to block RST packets originated from victim

$sudo iptables -I INPUT -p tcp --tcp-flags ALL RST -j DROP

$sudo iptables -I OUTPUT -p tcp --tcp-flags ALL RST -j DROP

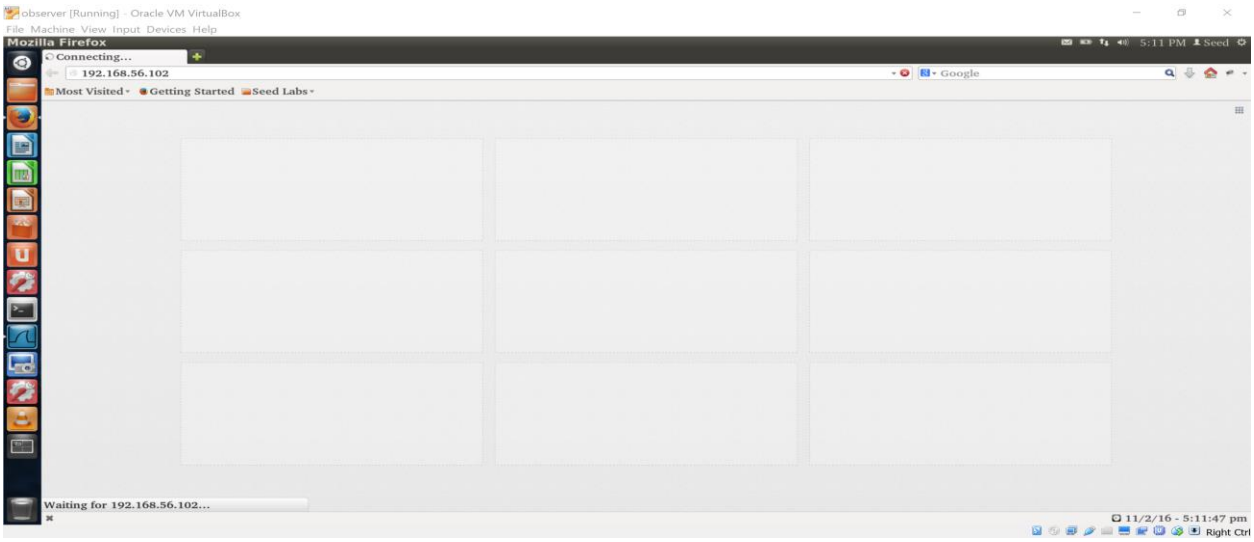


Figure 16: Denial of service due to SYN cookies being disable

Due to SYN flood, the observer faces a delay in requesting for the webpage but eventually the page is received.

The file for this attack is attached with the report and is called task3.py

Explanation

Although the three way handshake is essential for initiating new connections, it can be misused easily due to asymmetric allocation of resources in session initiation. The SYN flood causes a significant Denial of Service attack and causes the page to load after 5-6 seconds which was a little slower load as compared to when cookies were enabled. The attack could be obviously mitigated by using SYN cookies, but using firewall and active monitoring can be more effective to prevent the attack.

Attack 4: TCP RST attacks

Design

TCP RST packets were another of the ways to urgently end connection. The attacker can forge these packets using IP and MAC spoofing and can pretend to be one of the communicators. The experiment was done on telnet and ssh connections.

The observer was trying to remote login into victim whereas the attacker spoofed as victim was sending forged packets to observer.

Observations

Telnet

When the server is not under attack, the remote login works perfectly fine.

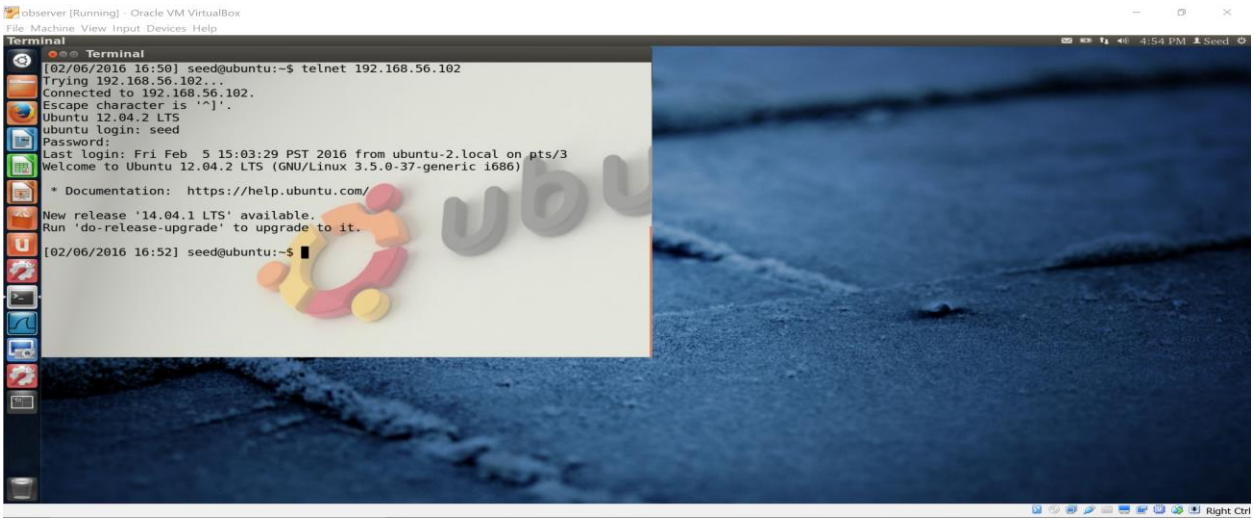


Figure 17: Normal telnet login before attack

After attack, the session gets closed due to TCP RST packet.

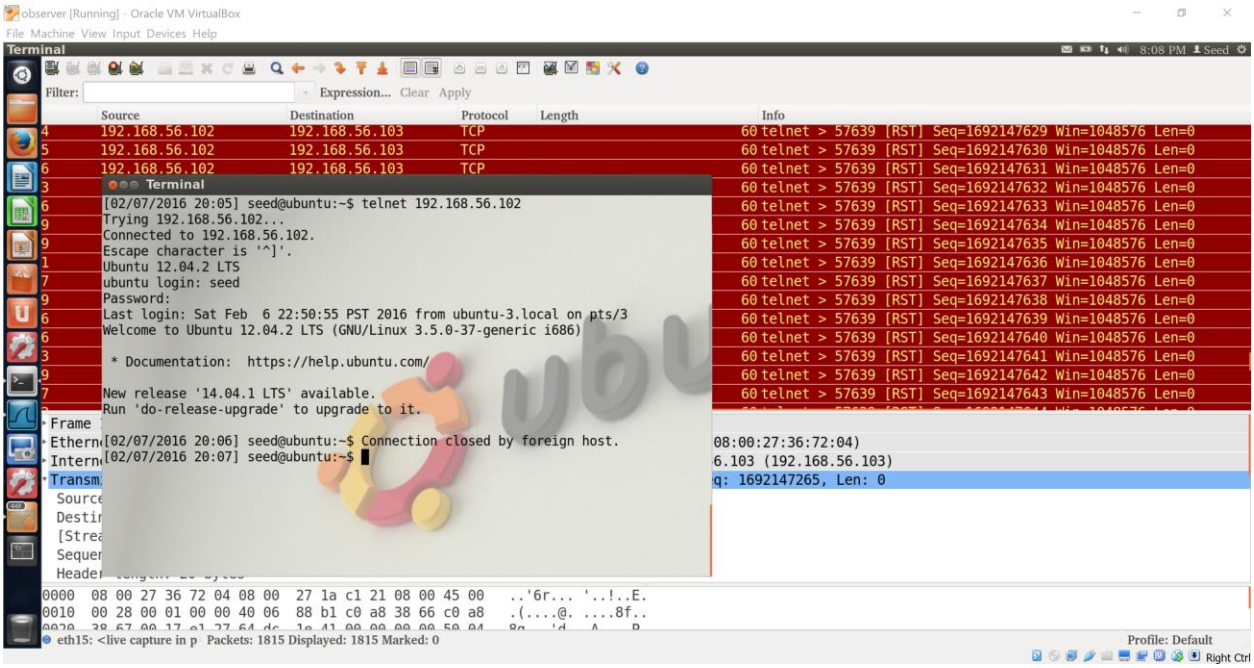


Figure 18: After attack the session got reset

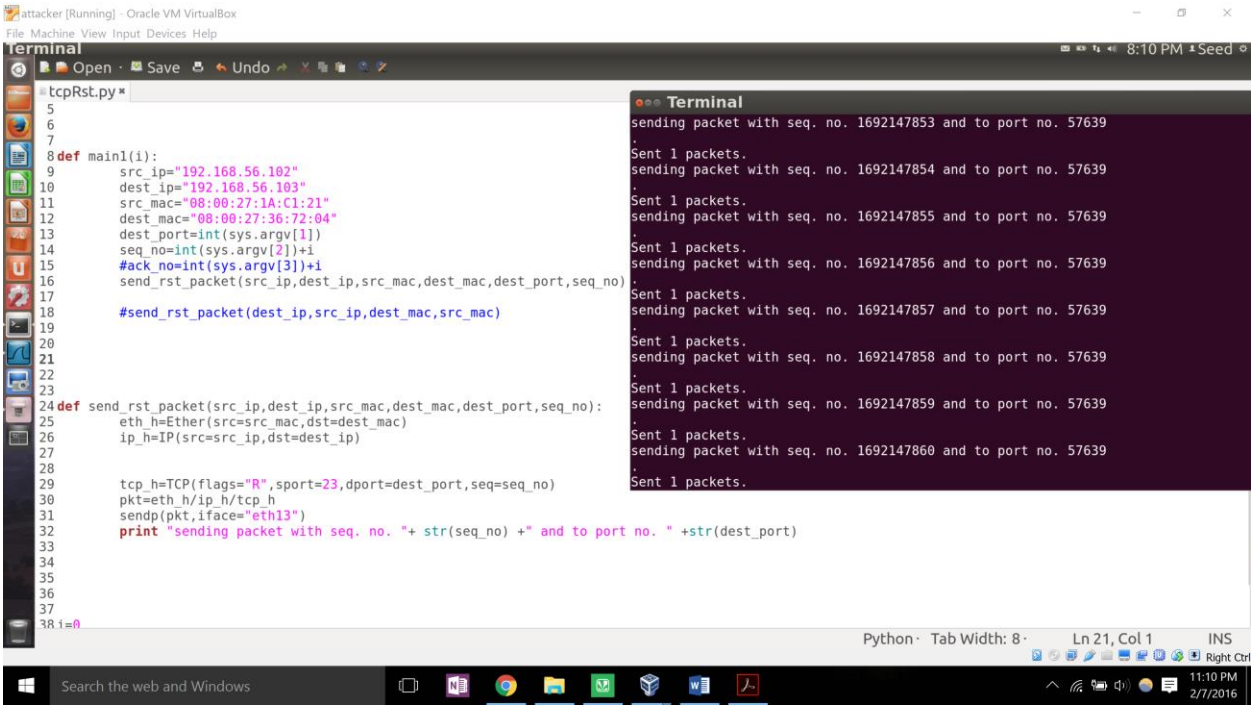


Figure 19: In the frame of attacker

SSH

Before attack

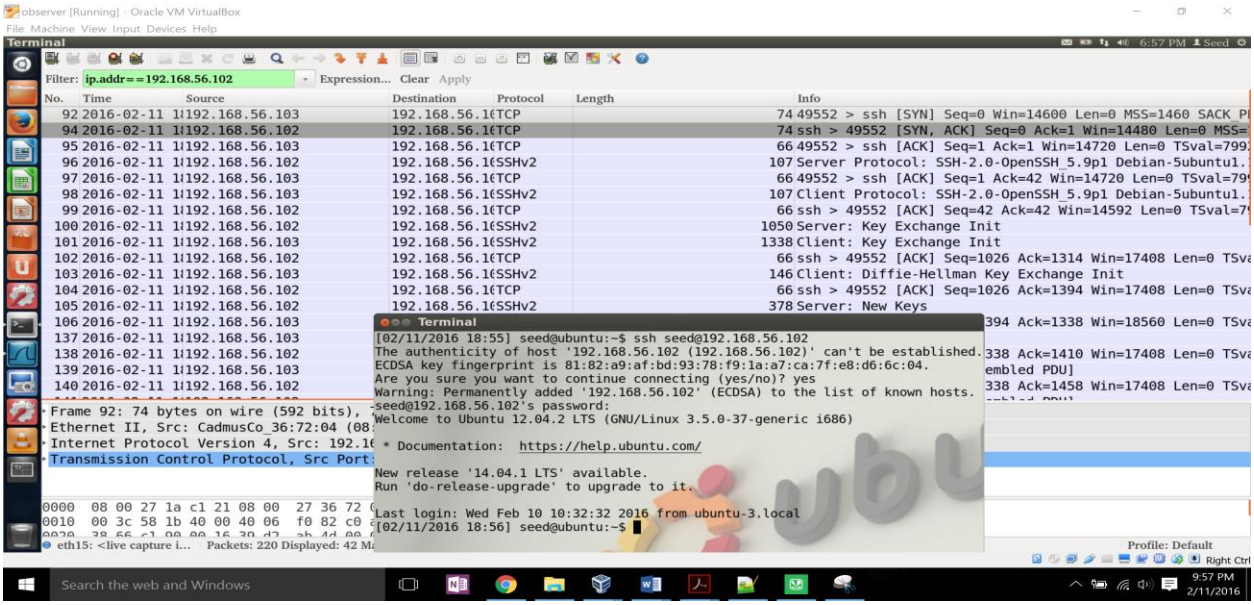


Figure 20: Normal SSH login before attack

After attack

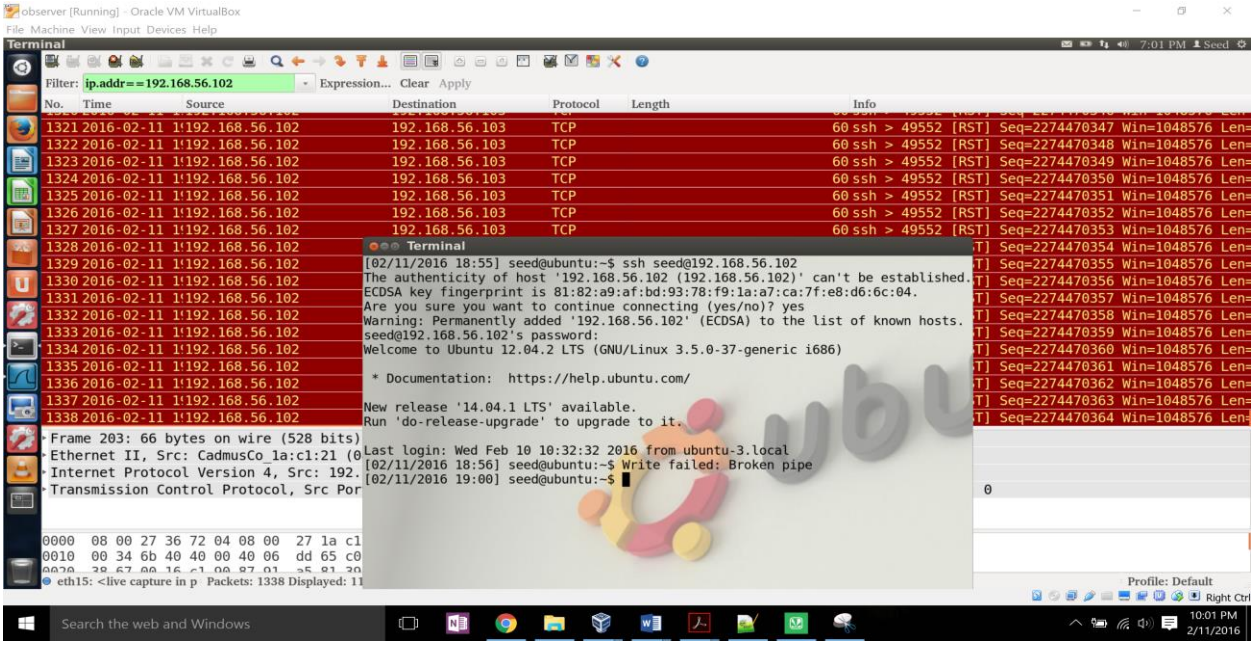


Figure 21: SSH login connection broken after attack

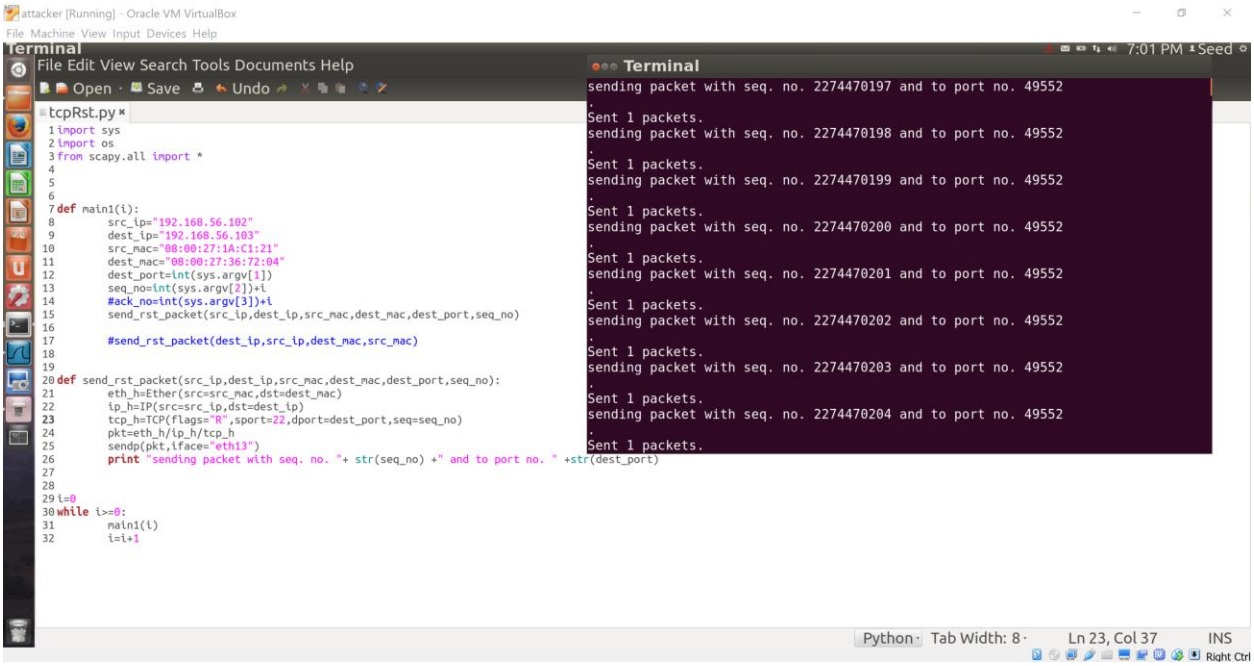


Figure 22: In the frame of attacker

There are two files attached with this report for the attack

Task41.py→ Telnet connections

Task42.py→ SSH connections

Explanation

This is a clear misuse of trust of the protocol as TCP RST packets can be easily attacked and attacker can misuse this trust to close SSH or Telnet connections. This however was tested for telnet and ssh, however it shall work on any connection involving TCP. The caveat in the attack is however for the attacker to guess the correct port number and sequence number.

Attack 5: TCP RST Attacks on Video Streaming Applications

Design

The TCP RST attack can work on any connection involving TCP. In this case the observer was acting as a content for video whereas the victim was trying to stream the video. The attacker spoofed as victim was sending forged RST packets to observer. The attack was extremely similar to task 4. The observer was delivering the video on port 8164.

Observations

Under normal conditions in the absence of attacker the video played smoothly and the Observer shows connection being established.

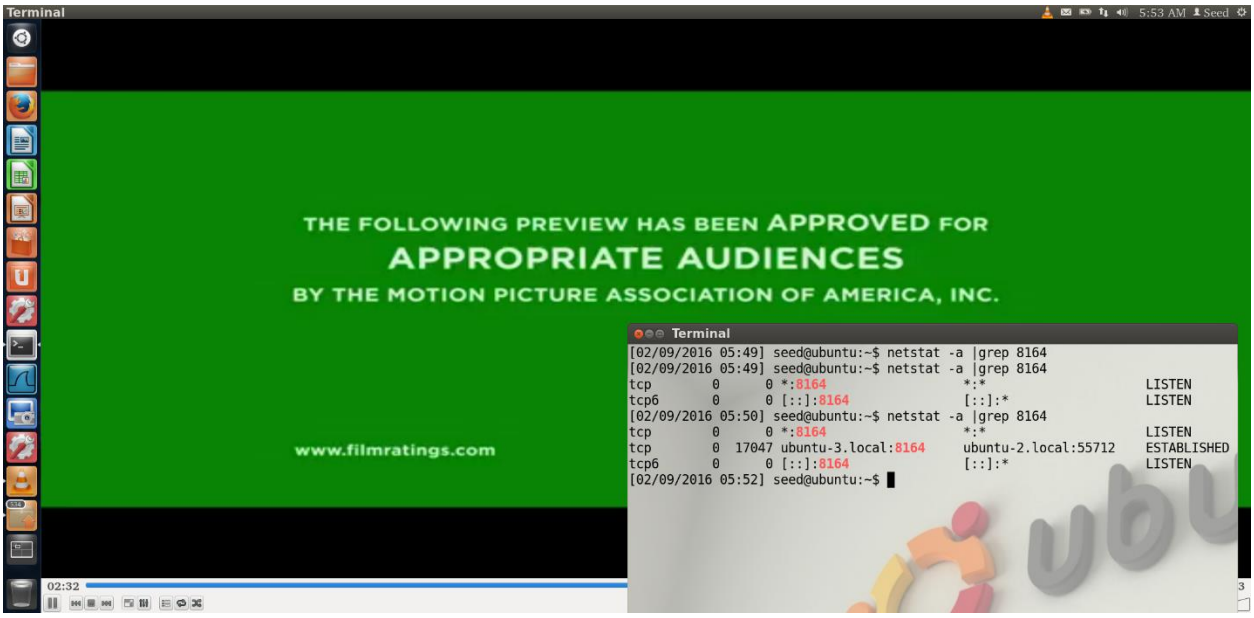


Figure 23: Session establishment and streaming before attack in the frame of observer

The victim was able to see the packets on wireshark and also able to play the video.

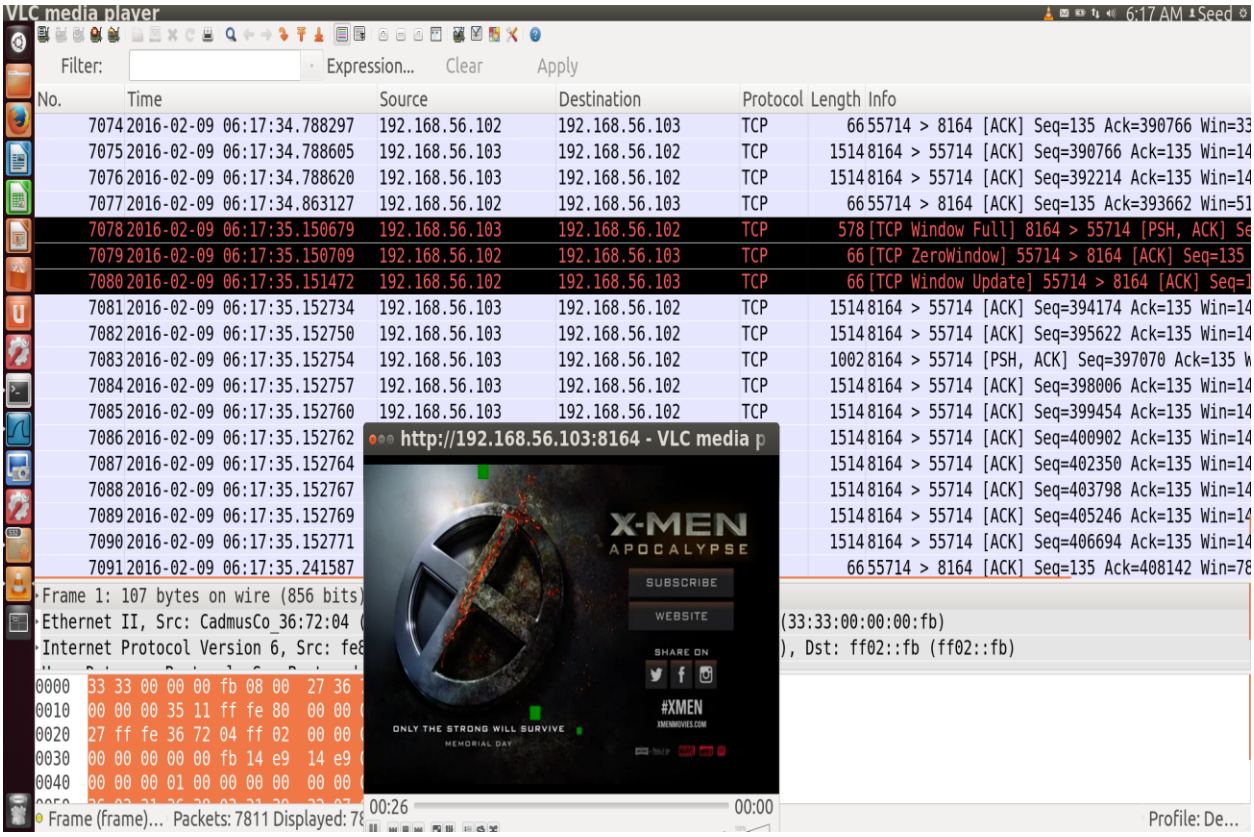


Figure 24:Video being streamed in frame of victim

After attack the connection broke and the video stopped streaming. The connection on the observer seem to be closed.

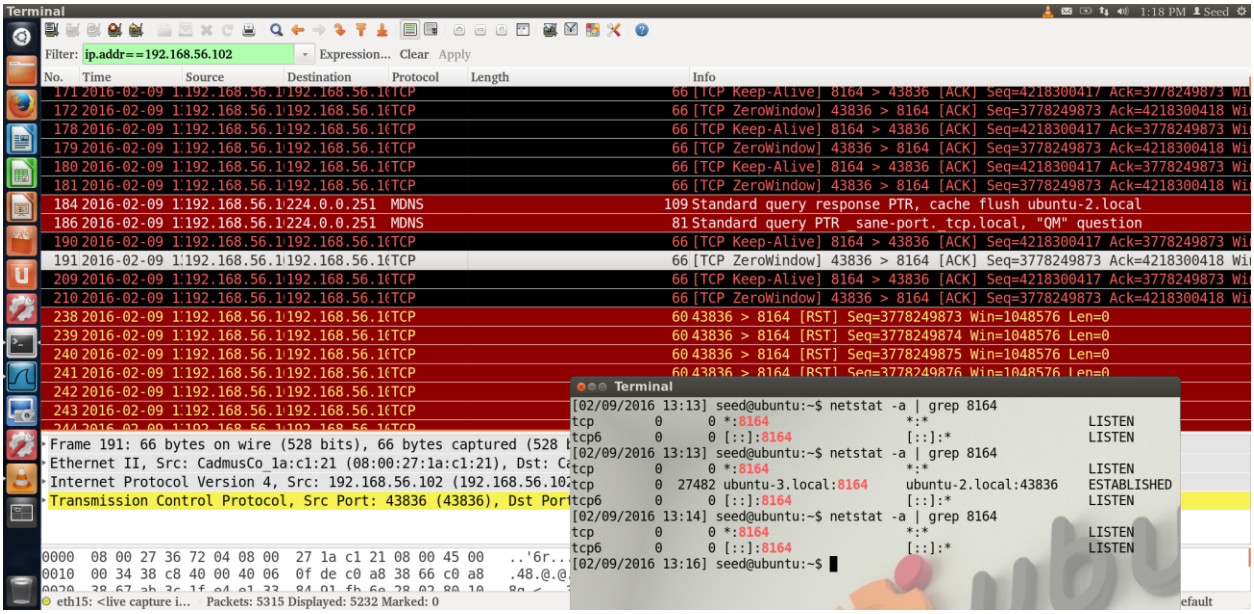


Figure 25: Connection closed due to attack using RST packets sent in frame of observer

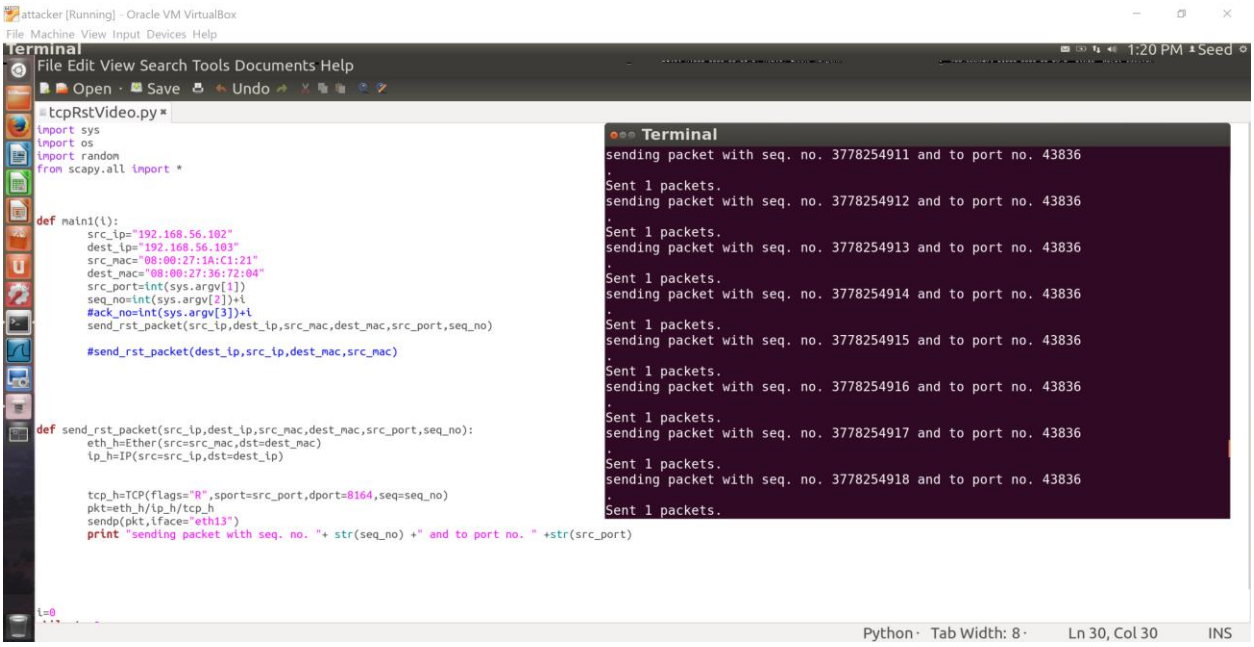


Figure 26: In frame of attacker during attack

The files attached are task5.py

Explanation

The attack was similar to task 4 attack, however a little more tougher since the video being streamed was not on a well -known port. Hence the attacker needs to guess both the port numbers as well as the sequence number in this case.

Attack 6: ICMP Blind Connection-Reset and Source-Quench Attacks

Design

ICMP error messages are used to indicate the network's failure to deliver packets. The ICMP error messages are divided into 2 categories: hard and soft errors. ICMP blind connection reset and source quench are both types of hard errors. These errors are useful for Path Maximum Transmit Unit (PMTU) discovery but can be attacked. However the current linux implementations no longer considers these hard errors as soft erros, which made it difficult to attack.

Observation

My experience with these attacks was that they didn't work out.

Different types of communication were tried

1) Streaming video
2) Chat Box between client and server
3) Unprompted data communication

None of the tools netwox or scapy worked out.

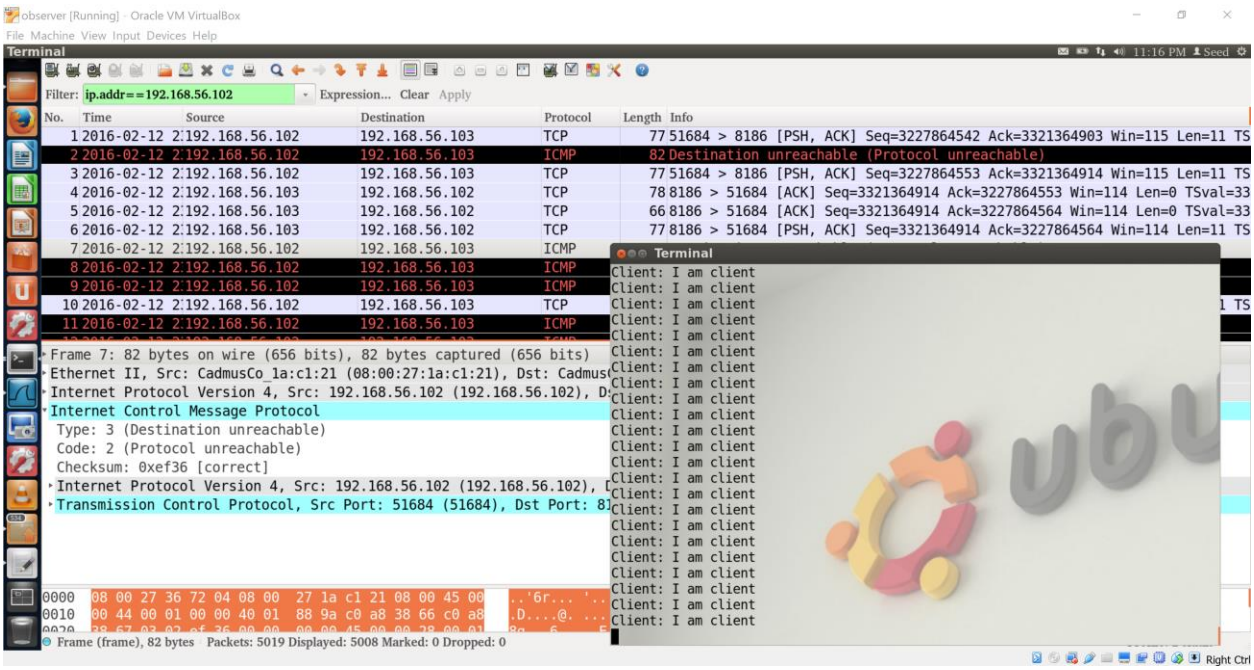


Figure 27: Observer sending "I am client" unprompted and ICMP reset has no effect on it

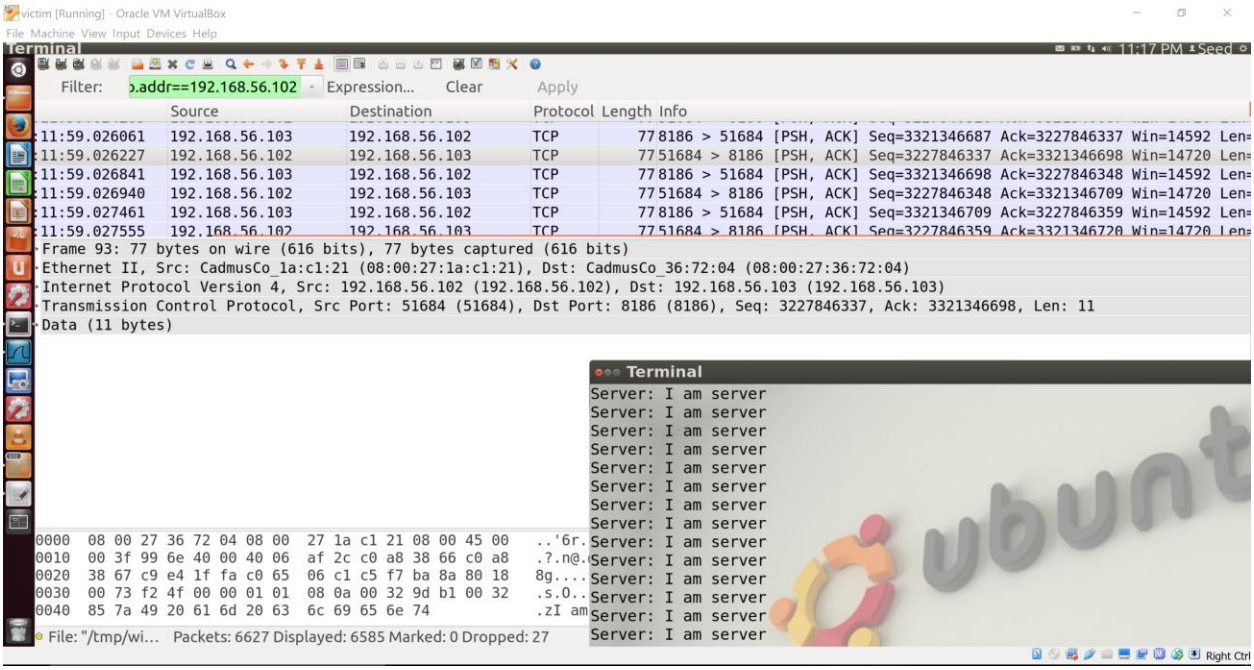


Figure 28: Observer sending "I am client" unprompted and ICMP reset has no effect on it

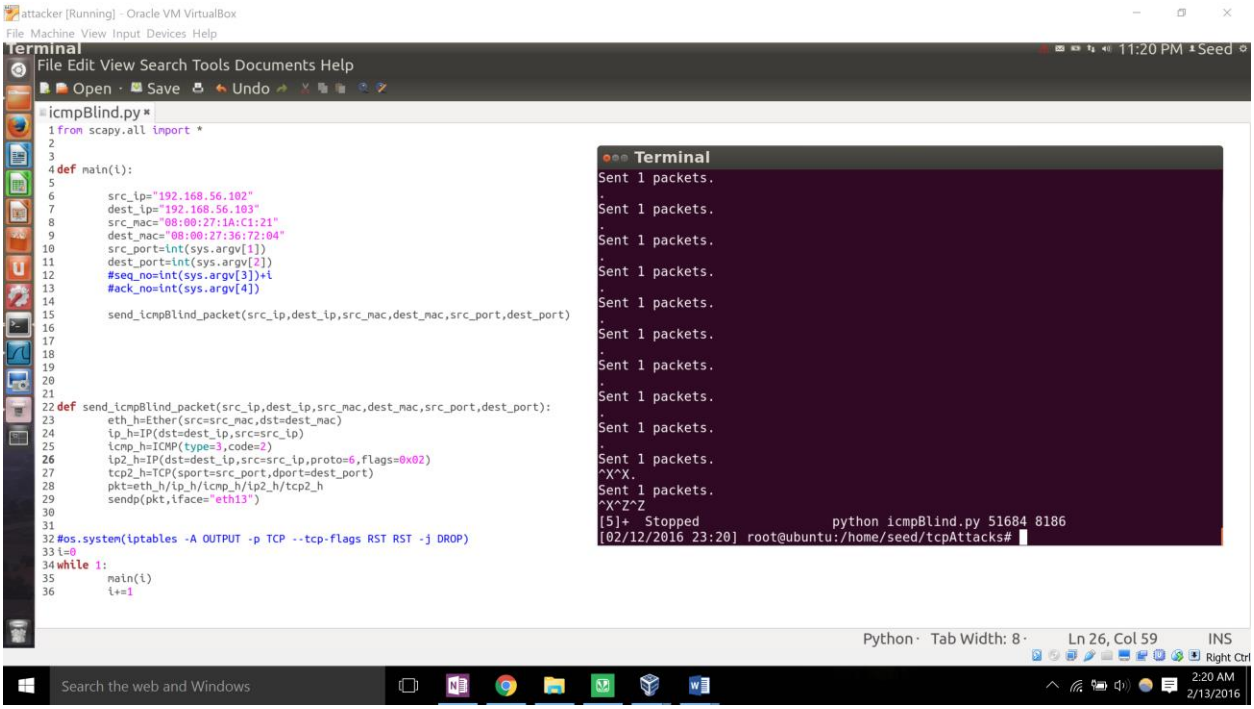


Figure 29: in frame of attacker during attack(ICMP blind reset)

The communication didn't get reset and continued despite error messages. The same was the case for ICMP source quench messages.

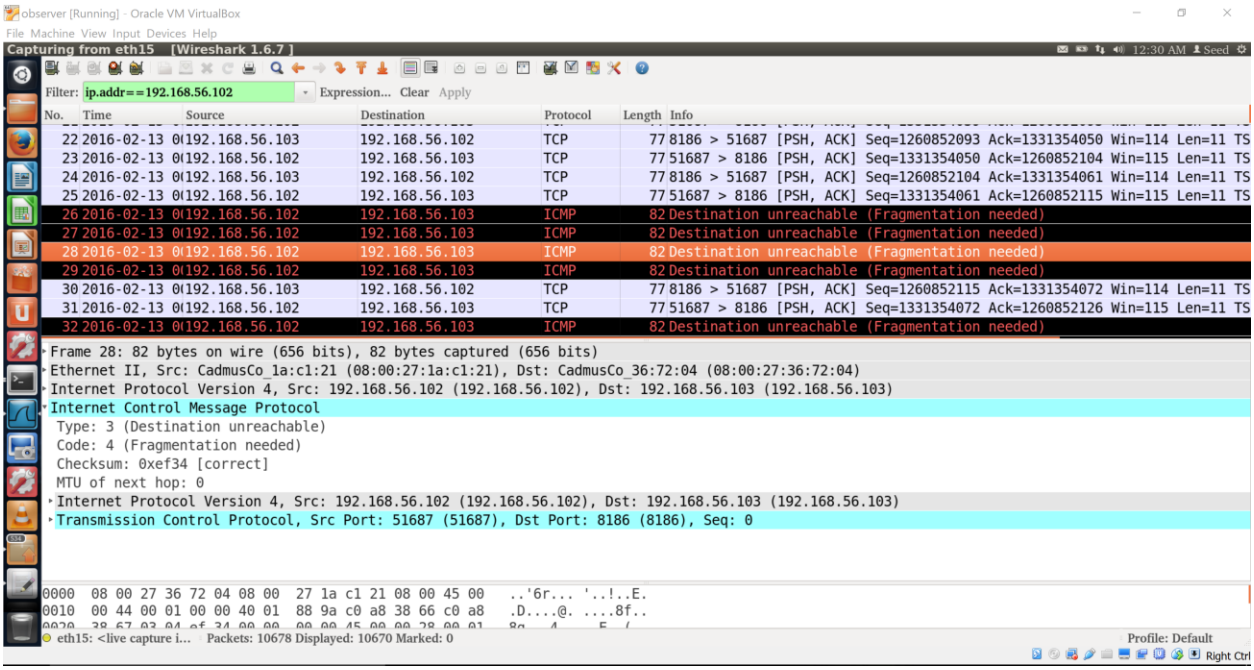


Figure 30: The sesson continues despite ICMP source quench messages

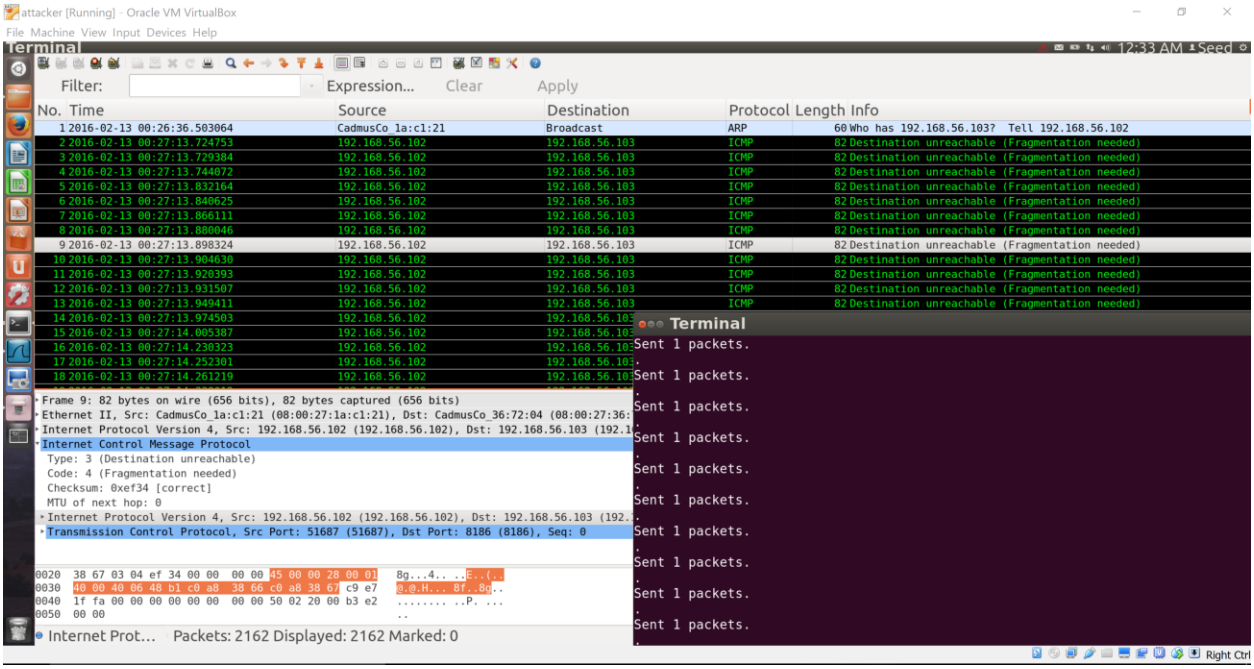


Figure 31: ICMP source quench messages in frame of attacker

Explanation

The reason for the attack to fail can be attributed to the changes made on Operating systems to handle ICMP errors.

The reading [2] has 3 important sections

1) The current Operating systems treat ICMP hard errors as soft errors and hence don't reset connection.

"Based on this analysis, most popular TCP implementations treat all ICMP "hard errors" received for connections in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, or TIME-WAIT) as "soft errors". That is, they do not abort the corresponding connection upon receipt of them."[2]

2) The reason is attributed to the fact that ICMP messages are sent unreliably and should not be used to reset TCP connections.

"Additionally, they do not extrapolate ICMP errors across TCP connections. This policy is based on the premise that TCP should be as robust as possible. Aborting the connection would be to ignore the valuable feature of the Internet -- that for many internal failures, it reconstructs its function without any disruption of the endpoints [RFC0816]."[2]

3) The functionality is given to upper layers to decide whether on receiving ICMP error messages, the connection should be reset or not.

"That is, communication should survive if there is still a working path to the destination system [DClark]. Additionally, applications may still be notified asynchronously about the error condition, and thus may still abort their connections on their own if they consider it appropriate. This counter-measure has been implemented in BSD-derived TCP/IP implementations (e.g., [FreeBSD], [NetBSD], and [OpenBSD]) for more than ten years [Wright][McKusick]. The Linux kernel has also implemented this policy for more than ten years [Linux]."[2]

Attack 7: TCP Session Hijacking

Design

TCP session hijacking is a way to harm the existing TCP connections by injecting malicious packets in between the connection. TCP RST attacks was one of the ways to hijack connection. However in this attack there is a chat system going on between Observer and Victim.

The attack was tries in two ways

MITM: The attacker tries to inject its own packets in between the communication. It attacks both observer and victim and the result is the message sent appears to be sent from observer to the victim and vice versa.

DOS: The attacker tries to inject malicious commands like exit to cause session logout

Observation

Man in the Middle

Before the attack

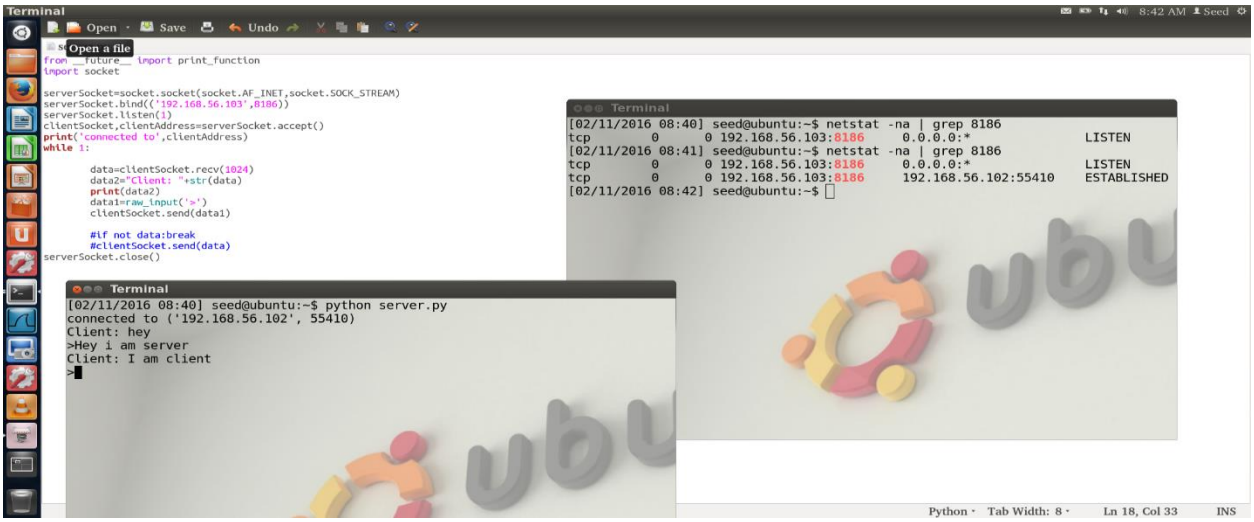


Figure 32: Normal chat session before attack in frame of Observer

After the attack

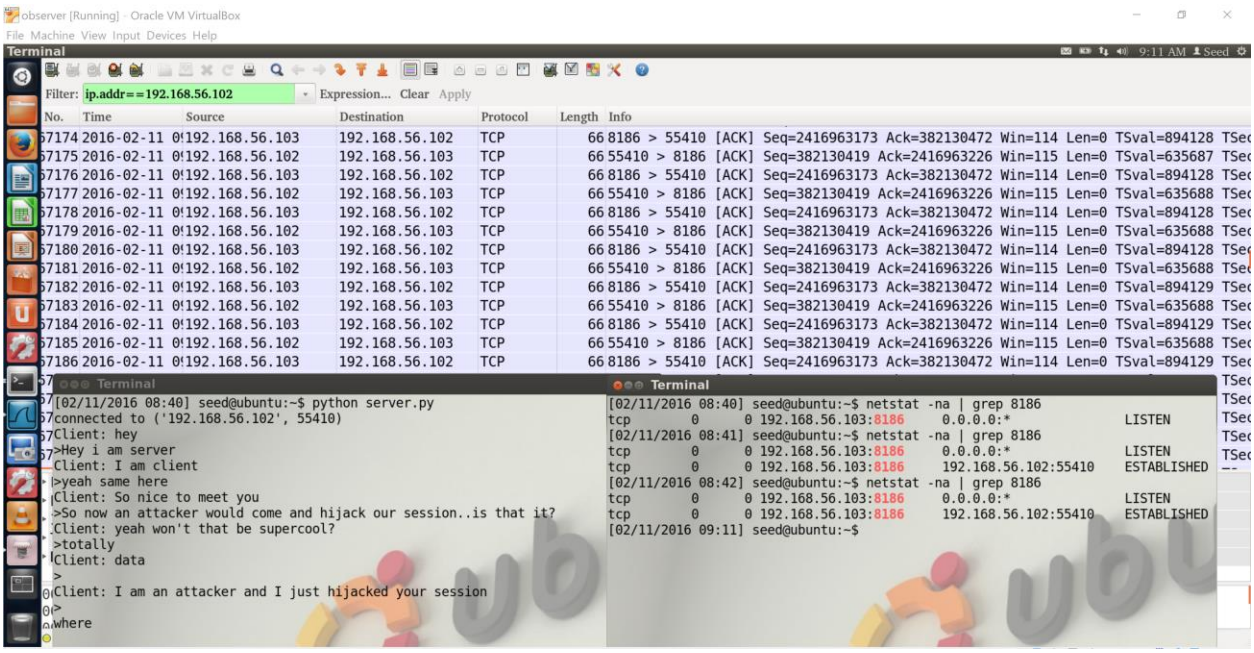


Figure 33: MITM attack in frame of observer. the attacker inserts his message "I am an attacker..."

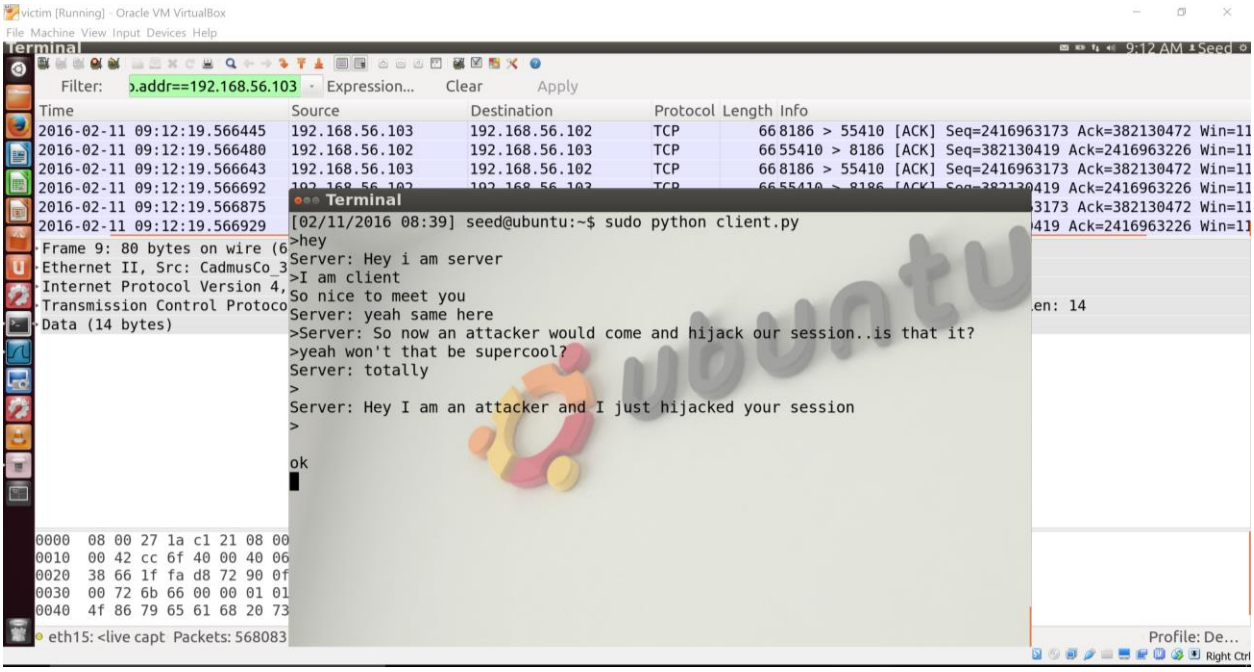


Figure 34: Frame of victim showing packet injected inside conversation

This however did cause the session to get completely sabotaged the packet injection flood messed up badly with the sequence numbers and valid packets can't be sent further.

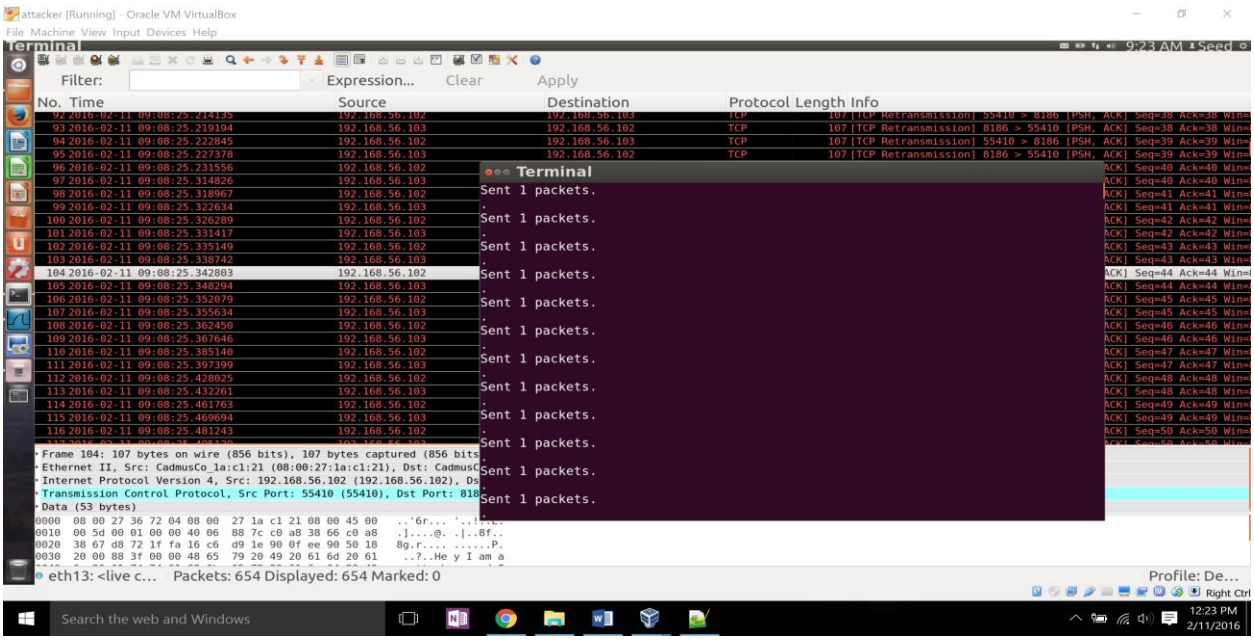


Figure 35: In frame of attacker

Denial of Service

The observer tries to telnet into victim and the connection is established.

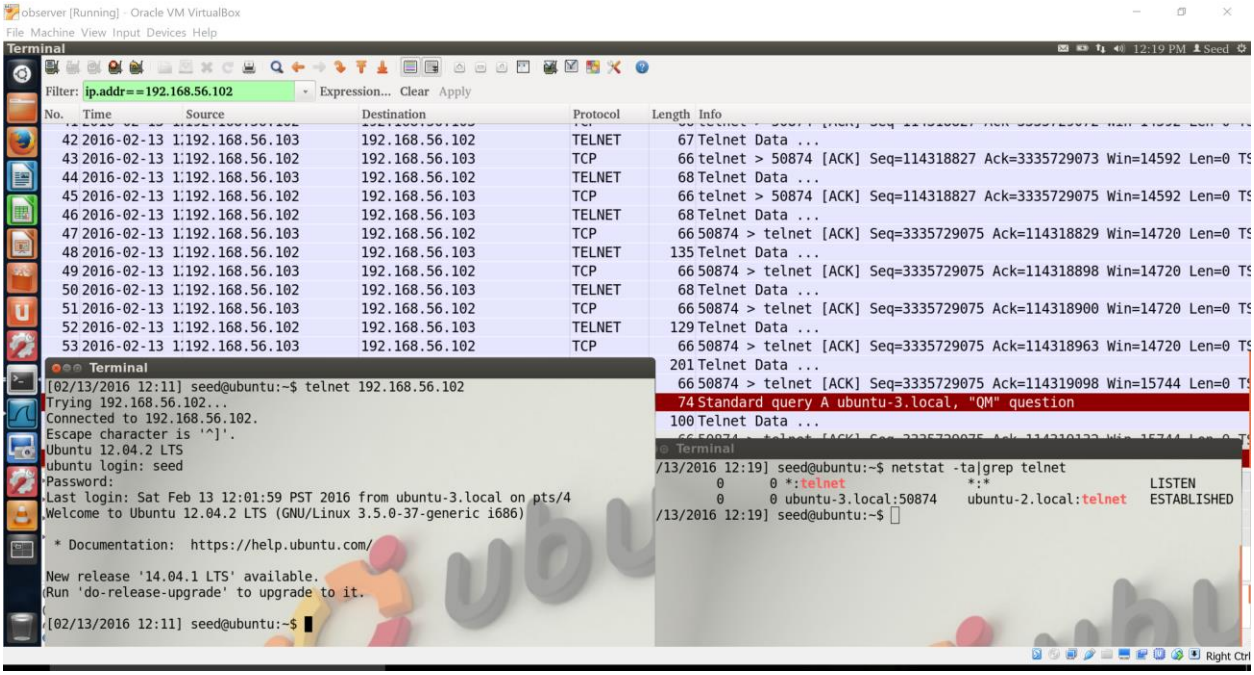


Figure 36: In frame of observer before attack; Observer logged in as victim

After attack, the connection is closed. The attacker spoofs IP and MC addresses and sends “exit” command to victim from observer. This generates an RST packet from victim to observer.

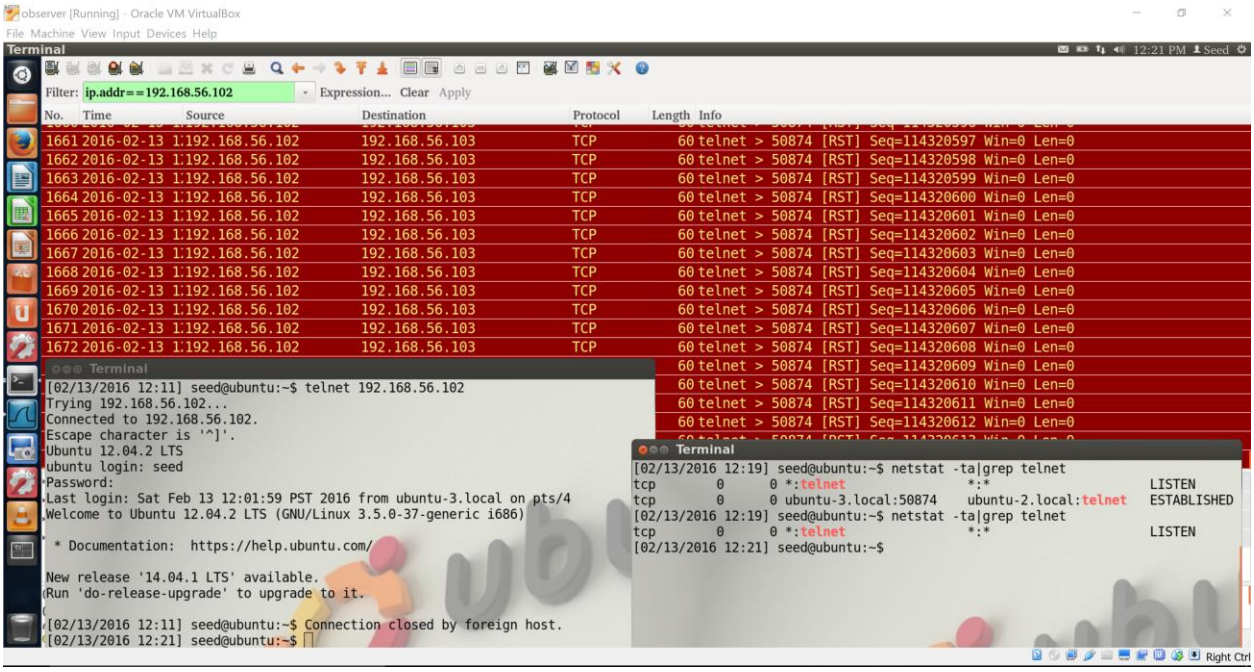


Figure 37: In frame of observer connection closed by victim

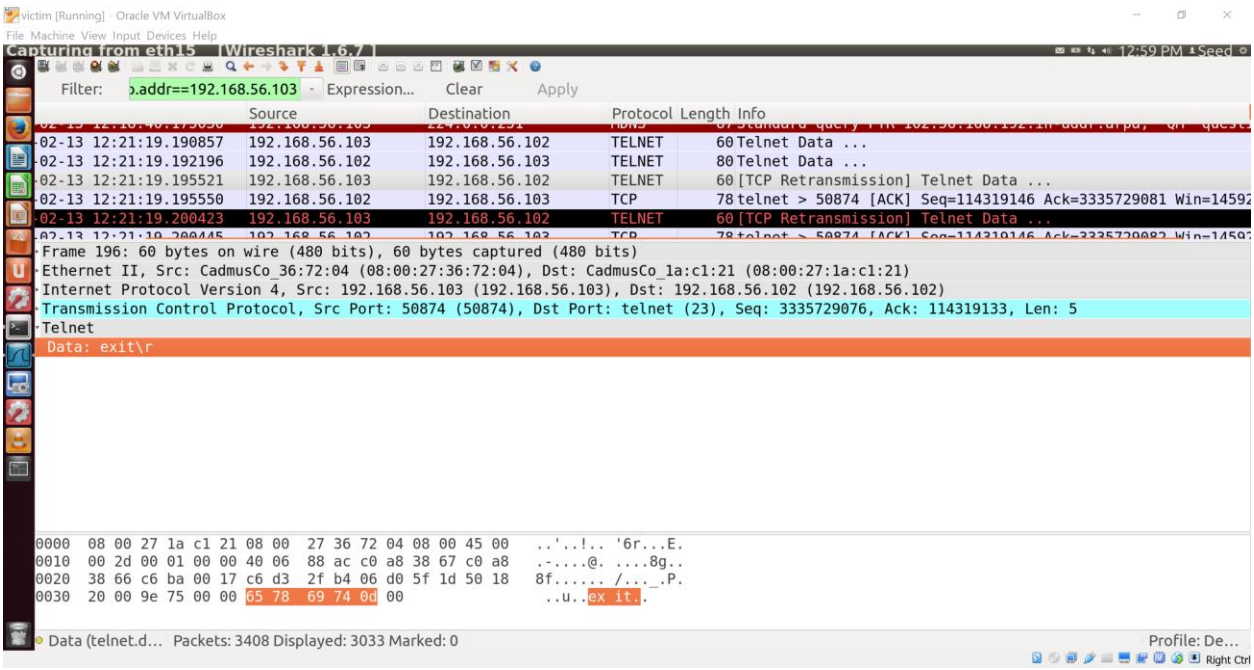


Figure 38:Spoofed packet sent by attacker to victim pretending as observer

The files used were

1) client.py running on victim
2) server.py running on observer
3) task7_MITM.py
4) task7_DOS.py

Explanation

The attacker spoofed both the observer and victim with the help of IP and MAC spoofing and injected its packet inside the stream. This attack would not have been successful without the attacker correctly guessing the correct port numbers of both observer and victim apart from the sequence and acknowledgement numbers.

Task 8

The task was to analyze

1) Initial sequence numbers
2) Selection of source port number

Though all three of them may appear random to the naïve eye, but there is no such thing as a random selection of a number in the computing word. Every one of these selection is based on algorithms.

Source Port numbers

There are algorithms suggested for port number selection in this rfc [3].An excerpt from [3]

"IANA has reserved the following use of the 16-bit port range of TCP and UDP:

- The Well-Known Ports, 0 through 1023.
- The Registered Ports, 1024 through 49151
- The Dynamic and/or Private Ports, 49152 through 65535

The dynamic port range defined by IANA consists of the 49152-65535 range, and is meant for the selection of ephemeral ports."

There are five algorithms defined in the RFC for ephemeral port selection. An attacker can reverse engineer the algorithm makes a good guess about the ports.

Algorithm 1: Simple Port Randomization Algorithm

Algorithm 2: Another Simple Port Randomization

Algorithm 3: Simple Hash-Based Port Selection

Algorithm 4: Double-Hash Port Selection Algorithm

Algorithm 5: Random-Increments Port Selection

For simplicity we shall try to reverse engineer the first one. With statistical analysis and some knowledge of Operating system of victim. The attacker can hugely minimize his guesses.

```
/* Initialization at system boot time. Could be random */
        next_ephemeral = min_ephemeral;
// next_ephemeral=49152


        /* Ephemeral port selection function */
        count = max_ephemeral - min_ephemeral + 1;
//count=65535-49152+1=16384
        do {
            port = next_ephemeral;
//port=49152
            if (next_ephemeral == max_ephemeral) {
                next_ephemeral = min_ephemeral;
            } else {
                next_ephemeral++;
//next_ephemeral=49153
            }



            if (check_suitable_port(port))
                return port;
//check if the port is available or busy, if available return it else try port 49153(the next_ephemeral
//port
            count--;


        } while (count > 0);


        return ERROR;
//return error if all ports are filled
```

Initial Sequence number (ISN) prediction

Predicting sequence number is tough but as per RFC 1948 we use the current 4 microsecond timer M and set

ISN = M + F(localhost, localport, remotehost, remoteport)

The rfc[4] adds
"It is vital that F not be computable from the outside, or an attacker could still guess at sequence numbers from the initial sequence number used for some other connection. We therefore suggest that F be a cryptographic hash function of the connection-id and some secret data. MD5 is a good choice, since the code is widely available. The secret data can either be a true random number, or it can be the combination of some per-host secret and the boot time of the machine."

References

[1] (http://www.cymru.com/gillsr/documents/icmp-redirects-are-bad.htm, n.d.)

[2] (http://www.faqs.org/rfcs/rfc5927.html)

[3] https://tools.ietf.org/html/rfc6056#page-11

[4] http://tools.ietf.org/html/rfc1948