PACKET SNIFFING USING SCAPY

UNIVERSITY OF ENGINEERING

&

MANAGEMENT, JAIPUR

PACKET SNIFFING USING SCAPY

Submitted in the partial fulfillment of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

Under

UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

BY

SHANYA, SHREYA GHOSAL

University Roll no: 12019002001147

12019002001113

University Registration no: 204201900200142

204201900200111

UNDER THE GUIDANCE OF

PROF. HRIDAY BANERJEE

COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF ENGINEERING & MANAGEMENT, JAIPUR

APPROVAL CERTIFICATE

This is to certify that the project report entitled "**Packet Sniffing Using Scapy**" submitted by **Shanya, Shreya Ghosal** (Roll:**12019002001147,12019002001113**) in partial fulfillment of the requirements of the degree of **Bachelor of Technology** in **Computer Science & Engineering** from University **of Engineering and Management, Jaipur** was carried out in a systematic and procedural manner to the best of our knowledge. It is a bona fide work of the candidate and was carried out under our supervision and guidance during the academic session of 2019-2023.

Prof. Hriday Banerjee
Project Guide, Assistant Professor (CSE)
UEM, JAIPUR

Prof. Mrinal Kanti Sarkar
HOD (CSE)
UEM, JAIPUR

Prof. A Mukherjee
Dean
UEM, JAIPUR

ACKNOWLEDGEMENT

The endless thanks goes to Lord Almighty for all the blessings he has showered onto us, which has enabled us to write this last note in our research work. During the period of our research, as in the rest of our life, we have been blessed by Almighty with some extraordinary people who have spun a web of support around us. Words can never be enough in expressing how grateful we are to those incredible people in our life who made this thesis possible. We would like an attempt to thank them for making our time during our research in the Institute a period we will treasure. We are deeply indebted to our research supervisor, Professor Hriday Banerjee for such an interesting thesis topic. Each meeting with him added in valuable aspects to the implementation and broadened our perspective. He has guided us with his invaluable suggestions, lightened up the way in our darkest times and encouraged us a lot in the academic life.

Shanya, Shreya Ghosal.

ABSTRACT

The issue of packet sniffing in a switched environment is the main emphasis of this work, while a brief examination of the impact in a non-switched context is included. ARP (Address Resolution Protocol) spoofing, which enables an attacker to eavesdrop on network traffic in a switched environment, and other methods are described in detail. The process of intercepting each packet as it travels over the network is called packet sniffing. It is a method where an user sniffs data that belongs to other network users. Both switched networks and non-switched networks can benefit from it. Packet sniffers may be used maliciously or as a tool for administration. Depending on the users' intentions. The various techniques for packet sniffing on Layer 2 switched networks are covered in this study. The various sniffing techniques will each be thoroughly discussed. The aim of the study is to demonstrate sniffing on switched networks and to comprehend how it may be avoided. A packet sniffer can be used to gather information that can aid an attacker in breaking into a network, including passwords, IP addresses, protocols being used on the network, and other details. Wiretapping, intrusion detection, network administration, and hacking are the main uses of packet sniffing.

TABLE OF CONTENTS

LIST OF FIGURES

1. CHAPTER INTRODUCTION

1.1 Cyber Attack

People are becoming increasingly aware of the significance of network security as computer network technologies and internet technology evolve more swiftly. The primary problem in computing is network security since there are more and more different sorts of assaults every day. One of the most important challenges is safeguarding network and computer security. The malicious nodes lead to network issues.

Attempts by malicious attackers to obtain access without authorization, steal data, or harm computers, computer networks, or other computing systems are known as cyber attacks. Any site may be used to conduct a cyberattack. A data breach might even be caused by a cyberattack. The data may then be modified, sold, held for ransom, or stolen by attackers. Data backup, penetration testing, bounty training, and fixing security flaws are a few prevention strategies.

1.2 Packet Sniffing

One of the most common attack is packet sniffing. The process of watching and capturing data traffic on a computer network is known as packet sniffing. Data is transmitted between computers over the internet or between any networks in the form of discrete units known as packets, which are then forwarded to their destination and reconstituted into a full message. An application called a packet sniffer runs on a network-attached device and passively captures all data link layer frames that are travelling via the network adapter of the device. It is often referred to as an Ethernet Sniffer, Network Analyzer, or Protocol Analyzer. [1]

A user can review and analyse web traffic that has been recorded using a variety of technologies. TCPDUMP, WIRESHARK, TCPFLOW, and Ethercap are the most well-known tools. This might make it difficult for network monitoring or users to recreate the visually focused web page. Therefore, we must create a software programme to facilitate users' and monitors' viewing, analysing, and decoding of their acquired network data. This can be done using scapy.[2]

With the help of the Python interpreter Scapy, we can produce, spoof, or decode packets over a network, as well as capture and examine them. We can also use it to introduce new packets into the network. It is capable of handling and manipulating wireless communication packets and supports a large variety of network protocols. Scapy can carry out the tasks carried out by several network tools, including nmap, hping, arpscan, and tshark (the command line of wireshark).

A packet sniffer can be used to gather information that can aid an attacker in breaking into a network, including passwords, IP addresses, protocols being used on the network, and other details. Wiretapping, intrusion detection, network administration, and hacking are the main uses of packet sniffing.

1.3 Sniffing packets in non-switched environment:

Sniffing packets is simple to achieve in a non-switched network environment. This is due to the fact that network traffic is broadcast to everyone via a hub. Every port that is linked to a hub receives traffic that is routed via it. So, to examine a computer linked to a hub, just connect a packet sniffer to an open port on the hub, which will enable seeing all traffic to and from all computers connected to that hub. It is proposed that a computer's network card be switched into a specific "promiscuous" mode in order to listen in on network traffic. Once in this mode, any application can access any network traffic (regardless of its destination) that reaches the network card (such as a packet sniffing program).

1.4 Sniffing packets in switched environment:

The most frequent sort of network is a switched environment. Data may be efficiently transported using switches for broadcast, unicast, and multicast traffic. Switches, however, bring a whole new degree of complexity to the task of a packet analyzer. According to their destination MAC addresses, packets in a switched network environment are only routed to the port they are intended for. In a switched environment, devices only receive packets that are intended for them, preventing promiscuous devices from sniffing extra packets. Only broadcast traffic and traffic sent and received by that computer are visible when sniffer is connected to a switch port. On a switched network, there are three main techniques for intercepting data from a target device: port mirroring, hubbing out, and ARP cache poisoning.

2. CHAPTER

LITERATURE REVIEW

Some literature reviews have been done based on this initial purpose.

[3] The main purpose of this research paper is to provide a secure atmosphere for our fellow classmates and peers in this internet-driven society. Unlike a standard tool that produces a large amount of data and takes a long time to filter out the desired results, the primary goal of this research is to enhance the sniffer private certificates. The primary motivation behind this project is to offer a highly recommended solution that individuals can use to simply keep track of the duties involved in monitoring their home network. Many things are uncommon because we lack network monitoring equipment, like TCP dump and wireshark. Recent study has also shown that

this project may be applied successfully in the field of education in addition to being valuable for network monitoring.

[4] The objective of the research is to demonstrate sniffing on switched networks and to comprehend how it may be avoided. Computers can connect with other local computers more effectively thanks to switched networks. The way that switches are utilised now as security features was never intended. They are intended to improve the utilisation of network capacity. By restricting traffic to the desired destination, this is achieved. Sniffing is therefore more difficult on switched networks. Modern networking protocol flaws have given rise to a number of sniffer techniques. Fortunately, there are various ways to avoid problems.

[5] The purpose of this article is to inform network administrators and specialists of the benefits of implementing Wireshark, a free and open source packet sniffer, to monitor the network. Since all packet sniffing tools function similarly to Wireshark, it was chosen for this paper's experimental goals. There are other packet-sniffing tools available. Additionally, the article provides real-world examples of frequent LAN assaults and shows how Wireshark may be used to spot them. Additionally, this article is separated into parts that illustrate several actual assaults on local networks, including port monitoring, DHCP flooding, DNS spoofing, ARP spoofing, and DDoS attacks.

[6] The topic of packet sniffing and methods for enhancing network security are covered in this article. On a switched network, sniffing is possible with third-party software. The outcome of using a variety of these tools on a switched, isolated network is shown, and it amply supports the assertion that perhaps the threat they present is substantial and genuine. The last part discusses defences against network eavesdropping in switched and non-switched contexts. The main argument of this essay is that encryption is the only real protection against the threat of sniffing.

3. CHAPTER

OBJECTIVE

- This project's major goal is to show that sniffing over switched networks is feasible. We'll use Scapy to see how to create and transmit packets, as well as how to examine and capture packets using packet sniffing software like tcpdump and wireshark.
- With the help of packet sniffer we can figure out why computer A and computer B can't interact, for instance (the communication may be impossible due to a variety of factors, such as an issue with the system or the transmission media). And also, recover the user names and passwords of network users.
- In addition to scanning, fundamental tasks like network tracing and port identification may also be performed, as well as packet sniffing and traffic monitoring. The scapy module will enable us to create a programme that will steal user names and passwords from unprotected websites.

4. CHAPTER

PROPOSED MODEL

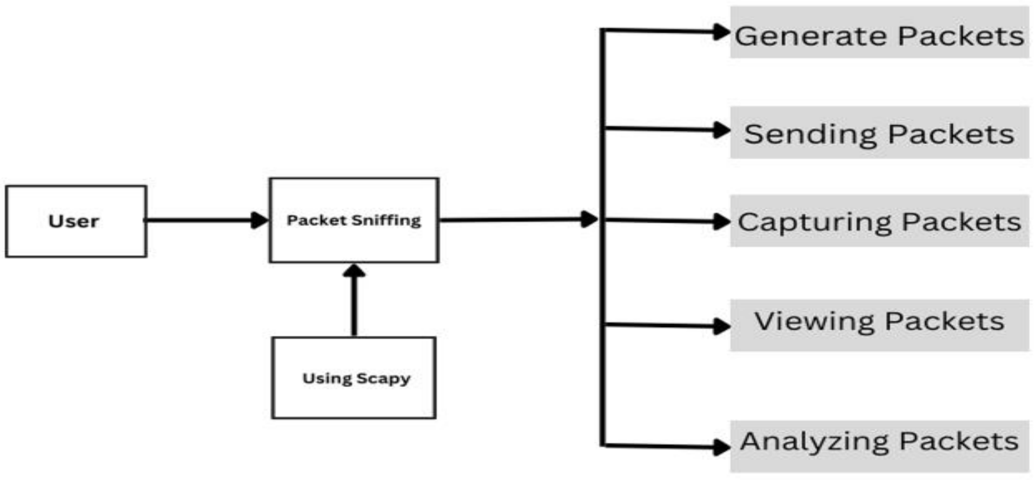


Fig 4.1: Data flow diagram of packet sniffer

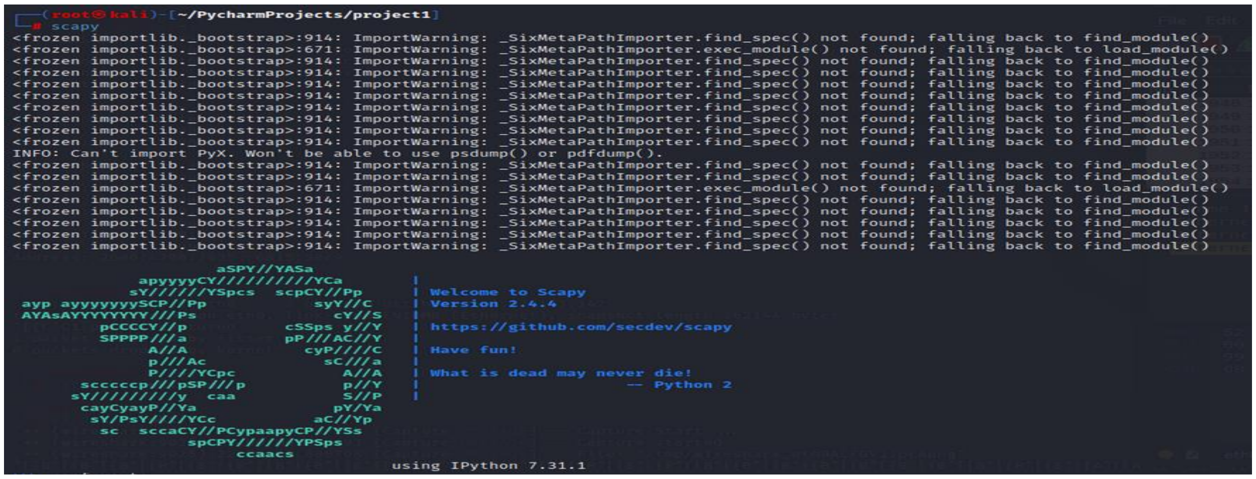
All standard networking operations including scanning, probing, tracing routing, and network discovery may be carried out with the Scapy module. Scapy's features may be used to develop new, high-level functions as per the author's demand, unlike other network modules (such as Wireshark or Nmap) were created for specific reasons like packet sniffing and network scanning.

5. CHAPTER
EXPERIMENTAL SETUP

Network Analysis Methods:

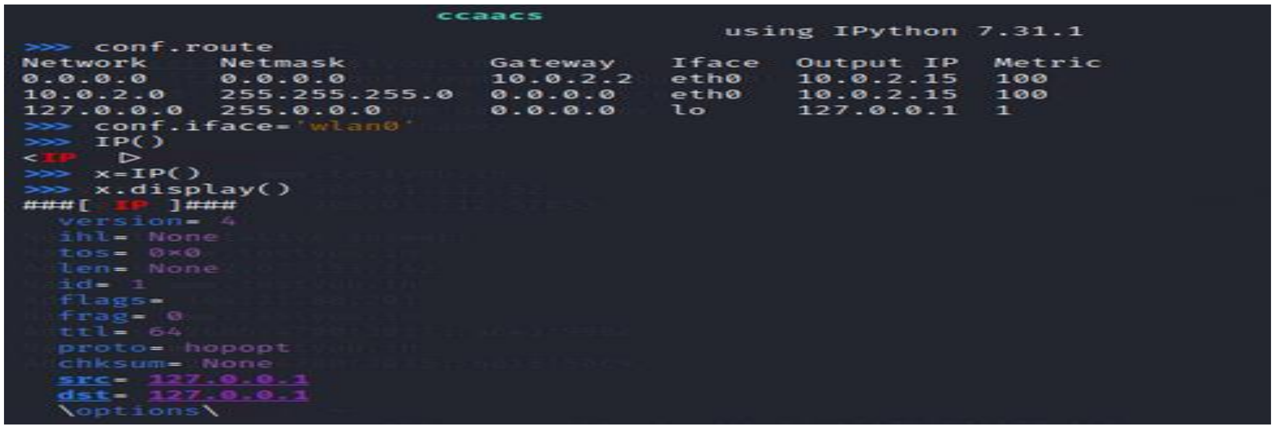
1.1 Scapy:

Scapy is a robust software for manipulating interactive packets. It can transmit packets over the wire, collect them, match requests and responses, forge or decode packets of many different protocols, and do a lot more. Python's interpreter serves as the command line interface for Scapy. Sending packets and getting responses are Scapy's key tasks. We create a collection of packets, it transmits them, gets responses, matches requests with responses, and then produces a list of packet pairs (request, answer) and a list of packets that weren't matched. This has the significant benefit over tools like Nmap or hping in that the entire packet is returned as an answer rather than just a (open/closed/filtered) response. To create our own automated tools, we can use Scapy. Scapy can also be expanded without modifying its source code. Scapy may be readily integrated into our own tools. Importing what we require will suffice.



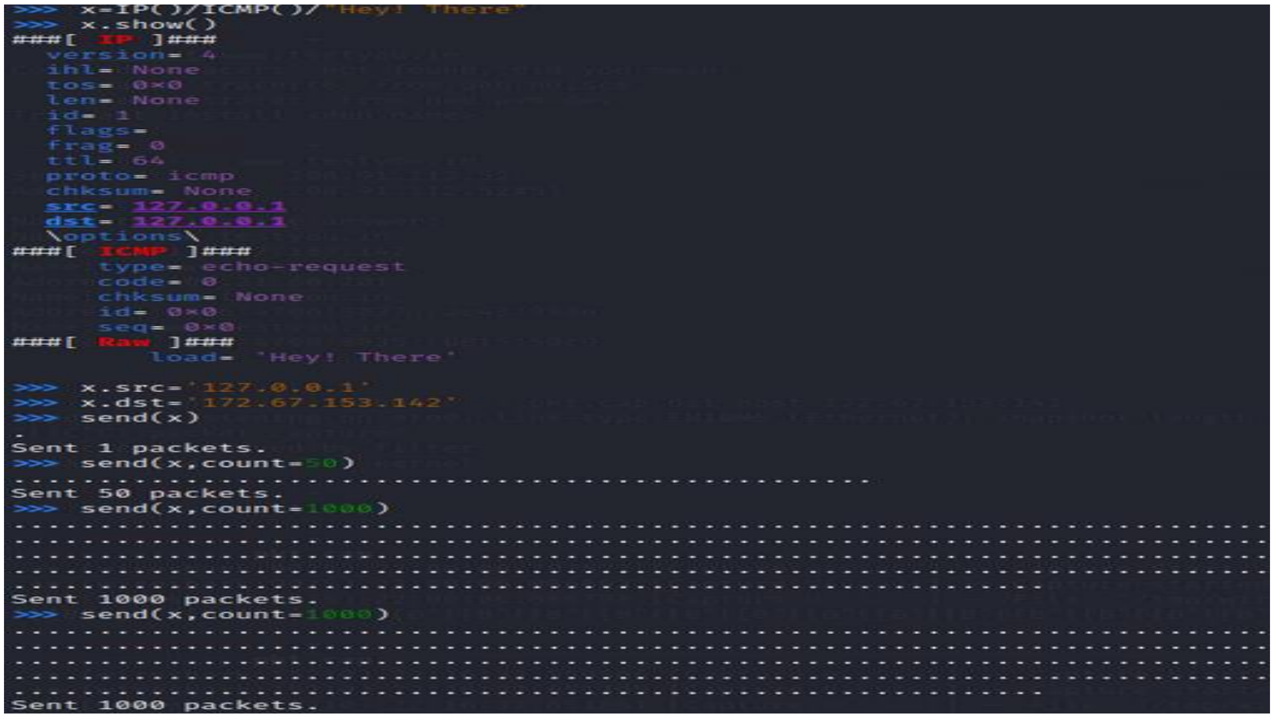
```
┌──(root㉿kali)-[~/PycharmProjects/project1]
└─# scapy
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:671: ImportWarning: _SixMetaPathImporter.exec_module() not found; falling back to load_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:671: ImportWarning: _SixMetaPathImporter.exec_module() not found; falling back to load_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()
<frozen importlib._bootstrap>:914: ImportWarning: _SixMetaPathImporter.find_spec() not found; falling back to find_module()

                     aSPY//YASa
             apyyyyCY//////////YCa       |
            sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.4
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | What is dead may never die!
       scccccp///pSP///p          p//Y   |                     -- Python 2
      sY/////////y  caa           S//P   |
       cayCyayP//Ya              pY/Ya
        sY/PsY////YCc          aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs
                                       using IPython 7.31.1
```

Fig 5.1: Starting Scapy



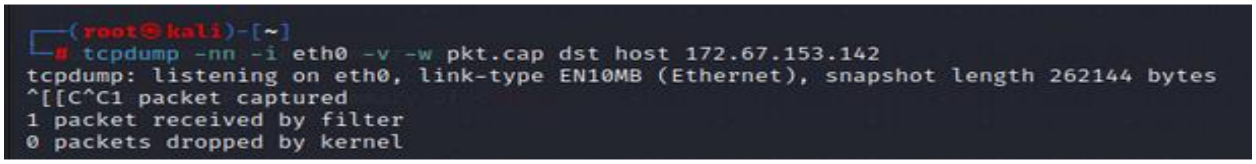
```
                         ccaacs
                                        using IPython 7.31.1
>>> conf.route
Network     Netmask          Gateway    Iface  Output IP  Metric
0.0.0.0     0.0.0.0          10.0.2.2   eth0   10.0.2.15  100
10.0.2.0    255.255.255.0    0.0.0.0    eth0   10.0.2.15  100
127.0.0.0   255.0.0.0        0.0.0.0    lo     127.0.0.1  1
>>> conf.iface='wlan0'
>>> IP()
<IP  |>
>>> x=IP()
>>> x.display()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= hopopt
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
```

Fig 5.2: To view Default setting



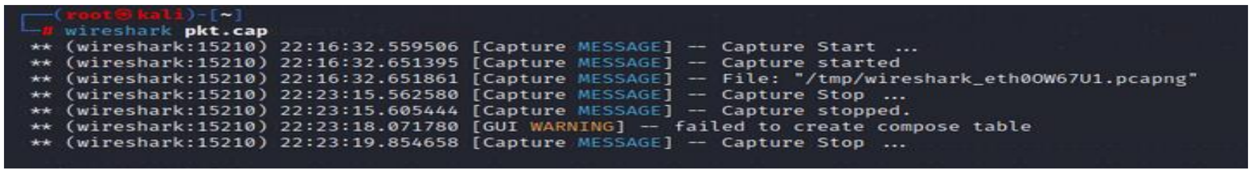
```
>>> x=IP()/ICMP()/"Hey! There"
>>> x.show()
###[ IP ]###
  version= 4
  ihl= None
  tos= 0x0
  len= None
  id= 1
  flags=
  frag= 0
  ttl= 64
  proto= icmp
  chksum= None
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
###[ ICMP ]###
     type= echo-request
     code= 0
     chksum= None
     id= 0x0
     seq= 0x0
###[ Raw ]###
        load= 'Hey! There'

>>> x.src='127.0.0.1'
>>> x.dst='172.67.153.142'
>>> send(x)
.
Sent 1 packets.
>>> send(x,count=50)
..................................................
Sent 50 packets.
>>> send(x,count=1000)
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
.............................................................
Sent 1000 packets.
>>> send(x,count=1000)
..........................................................................
..........................................................................
..........................................................................
..........................................................................
..........................................................................
.............................................................
Sent 1000 packets.
```

Fig 5.3: Generating and sending packets



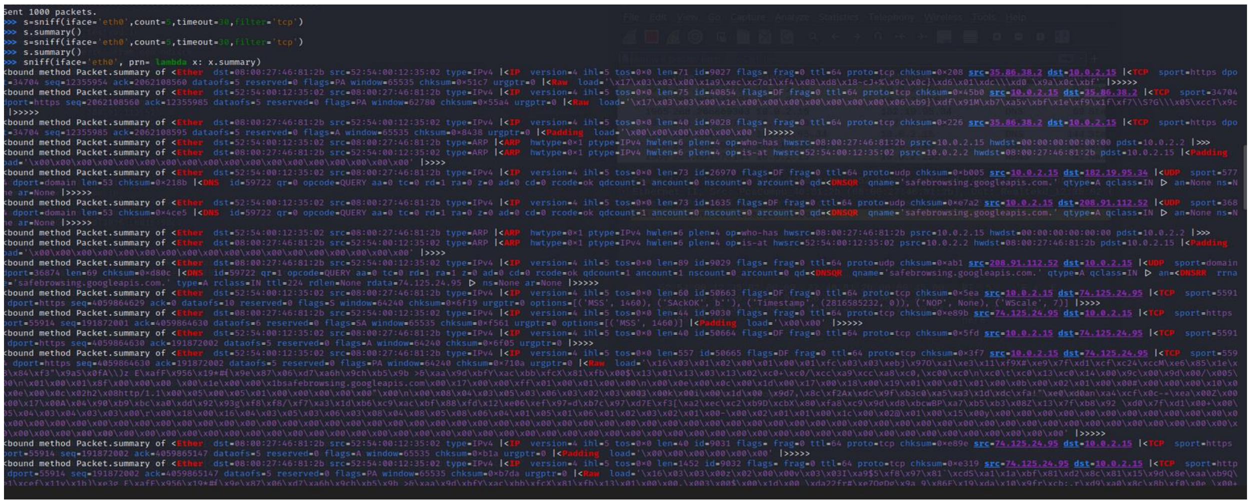
```
┌──(root㉿kali)-[~]
└─# tcpdump -nn -i eth0 -v -w pkt.cap dst host 172.67.153.142
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^[[C^C1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Fig 5.4: Capturing Packet Using Tcpdump



```
┌──(root㉿kali)-[~]
└─# wireshark pkt.cap
 ** (wireshark:15210) 22:16:32.559506 [Capture MESSAGE] -- Capture Start ...
 ** (wireshark:15210) 22:16:32.651395 [Capture MESSAGE] -- Capture started
 ** (wireshark:15210) 22:16:32.651861 [Capture MESSAGE] -- File: "/tmp/wireshark_eth0OW67U1.pcapng"
 ** (wireshark:15210) 22:23:15.562580 [Capture MESSAGE] -- Capture Stop ...
 ** (wireshark:15210) 22:23:15.605444 [Capture MESSAGE] -- Capture stopped.
 ** (wireshark:15210) 22:23:18.071780 [GUI WARNING] -- failed to create compose table
 ** (wireshark:15210) 22:23:19.854658 [Capture MESSAGE] -- Capture Stop ...
```

Fig 5.5: Viewing Data Packets in Wireshark

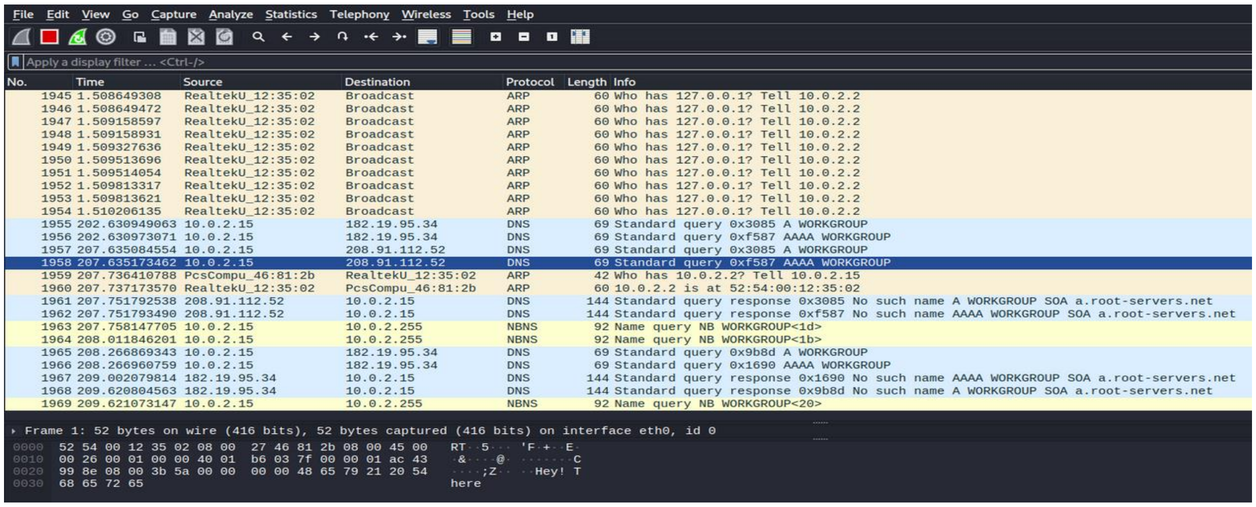


```
Sent 1000 packets.
>>> s=sniff(iface='eth0',count=5,timeout=30,filter='tcp')
>>> s.summary()
>>> s=sniff(iface='eth0',count=5,timeout=30,filter='tcp')
>>> s.summary()
>>> sniff(iface='eth0', prn= lambda x: x.summary)
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=71 id=9027 flags= frag=0 ttl=64 proto=tcp chksum=0x208 src=35.86.38.2 dst=10.0.2.15 |<TCP  sport=https dpo
t=34704 seq=12355954 ack=2062108560 dataofs=5 reserved=0 flags=PA window=65535 chksum=0x51c7 urgptr=0 |<Raw  load='\x17\x03\x03\x00\x1a9\xec\xc7o1\xf4\x08\xd8\x18~cJ+%\x9c\x0c}\xd6\x01\xdc\\\xd0 \x9a\x0c\xbf' |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=75 id=40854 flags=DF frag=0 ttl=64 proto=tcp chksum=0x45b0 src=10.0.2.15 dst=35.86.38.2 |<TCP  sport=34704
dport=https seq=2062108560 ack=12355985 dataofs=5 reserved=0 flags=PA window=62780 chksum=0x55a4 urgptr=0 |<Raw  load='\x17\x03\x03\x00\x1e\x00\x00\x00\x00\x00\x00\x00\x06\xb9}\xdf\x91M\xb7\xa5v\xbf\x1e\xf9\x1f\xf7\\S?G\\\x05\xccT\x9c
 |>>>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=40 id=9028 flags= frag=0 ttl=64 proto=tcp chksum=0x226 src=35.86.38.2 dst=10.0.2.15 |<TCP  sport=https dpo
t=34704 seq=12355985 ack=2062108595 dataofs=5 reserved=0 flags=A window=65535 chksum=0x8438 urgptr=0 |<Padding  load='\x00\x00\x00\x00\x00\x00' |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=ARP |<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=08:00:27:46:81:2b psrc=10.0.2.15 hwdst=00:00:00:00:00:00 pdst=10.0.2.2 |>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=ARP |<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=52:54:00:12:35:02 psrc=10.0.2.2 hwdst=08:00:27:46:81:2b pdst=10.0.2.15 |<Padding
oad='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=73 id=26970 flags=DF frag=0 ttl=64 proto=udp chksum=0xb005 src=10.0.2.15 dst=182.19.95.34 |<UDP  sport=577
4 dport=domain len=53 chksum=0x218b |<DNS  id=59722 qr=0 opcode=QUERY aa=0 tc=0 rd=1 ra=0 z=0 ad=0 cd=0 rcode=ok qdcount=1 ancount=0 nscount=0 arcount=0 qd=<DNSQR  qname='safebrowsing.googleapis.com.' qtype=A qclass=IN |> an=None ns=N
ne ar=None |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=73 id=1635 flags=DF frag=0 ttl=64 proto=udp chksum=0xe7a2 src=10.0.2.15 dst=208.91.112.52 |<UDP  sport=368
4 dport=domain len=53 chksum=0x4ce5 |<DNS  id=59722 qr=0 opcode=QUERY aa=0 tc=0 rd=1 ra=0 z=0 ad=0 cd=0 rcode=ok qdcount=1 ancount=0 nscount=0 arcount=0 qd=<DNSQR  qname='safebrowsing.googleapis.com.' qtype=A qclass=IN |> an=None ns=N
ne ar=None |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=ARP |<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=who-has hwsrc=08:00:27:46:81:2b psrc=10.0.2.15 hwdst=00:00:00:00:00:00 pdst=10.0.2.2 |>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=ARP |<ARP  hwtype=0x1 ptype=IPv4 hwlen=6 plen=4 op=is-at hwsrc=52:54:00:12:35:02 psrc=10.0.2.2 hwdst=08:00:27:46:81:2b pdst=10.0.2.15 |<Padding
oad='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=89 id=9029 flags= frag=0 ttl=64 proto=udp chksum=0xab1 src=208.91.112.52 dst=10.0.2.15 |<UDP  sport=domain
dport=36874 len=69 chksum=0xd80c |<DNS  id=59722 qr=1 opcode=QUERY aa=0 tc=0 rd=1 ra=1 z=0 ad=0 cd=0 rcode=ok qdcount=1 ancount=1 nscount=0 arcount=0 qd=<DNSQR  qname='safebrowsing.googleapis.com.' qtype=A qclass=IN |> an=<DNSRR  rrna
e='safebrowsing.googleapis.com.' type=A rclass=IN ttl=224 rdlen=None rdata=74.125.24.95 |> ns=None ar=None |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=60 id=50663 flags=DF frag=0 ttl=64 proto=tcp chksum=0x5ea src=10.0.2.15 dst=74.125.24.95 |<TCP  sport=5591
 dport=https seq=4059864629 ack=0 dataofs=10 reserved=0 flags=S window=64240 chksum=0x6f19 urgptr=0 options=[('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2816585232, 0)), ('NOP', None), ('WScale', 7)] |>>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=44 id=9030 flags= frag=0 ttl=64 proto=tcp chksum=0xe89b src=74.125.24.95 dst=10.0.2.15 |<TCP  sport=https
ort=55914 seq=191872001 ack=4059864630 dataofs=6 reserved=0 flags=SA window=65535 chksum=0xf561 urgptr=0 options=[('MSS', 1460)] |<Padding  load='\x00\x00' |>>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=40 id=50664 flags=DF frag=0 ttl=64 proto=tcp chksum=0x5fd src=10.0.2.15 dst=74.125.24.95 |<TCP  sport=5591
 dport=https seq=4059864630 ack=191872002 dataofs=5 reserved=0 flags=A window=64240 chksum=0x6f05 urgptr=0 |>>>>
<bound method Packet.summary of <Ether  dst=52:54:00:12:35:02 src=08:00:27:46:81:2b type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=557 id=50665 flags=DF frag=0 ttl=64 proto=tcp chksum=0x3f7 src=10.0.2.15 dst=74.125.24.95 |<TCP  sport=559
4 dport=https seq=4059864630 ack=191872002 dataofs=5 reserved=0 flags=PA window=64240 chksum=0x710a urgptr=0 |<Raw  load='\x16\x03\x01\x02\x00\x01\x00\x01\xfc\x03\x03\xebj\x97O\xa1\xe3\x11\xf9X#\xe9\x7f\xd1\xcf\xc24\xccM\xe6\x85\x1e\x
3\x84\xf3"\x9aS\x0fA\\)z E\xafF\x956\x19*#{\x9e\x87\x06\xd7\xa6h\x9ch\xb5\x9b >6\xaa\x9d\xbfY\xac\xbb\xfcX\x81\xfb\x00$\x13\x01\x13\x03\x13\x02\xc0+\xc0/\xcc\xa9\xcc\xa8\xc0,\xc00\xc0\n\xc0\t\xc0\x13\xc0\x14\x00\x9c\x00\x9d\x00/\x005\
00\n\x01\x00\x01\x8f\x00\x00\x00 \x00\x1e\x00\x00\x1bsafebrowsing.googleapis.com\x00\x17\x00\x00\xff\x01\x00\x01\x00\x00\n\x00\x0e\x00\x0c\x00\x1d\x00\x17\x00\x18\x00\x19\x01\x00\x01\x01\x00\x0b\x00\x02\x01\x00\x00#\x00\x00\x00\x10\x0
x0e\x00\x0c\x02h2\x08http/1.1\x00\x05\x00\x05\x01\x00\x00\x00\x00\x00"\x00\n\x00\x08\x04\x03\x05\x03\x06\x03\x02\x03\x003\x00k\x00i\x00\x1d\x00 \x9d7,\x8c\xf2Ax\xdc\x9f\xb3c0\xa5\xa3\x1d\xdc\xfa!"\xe0\xd0an\xa4\xcf\x0c-=\xea\x00Z\x00
x00\x17\x00A\x04\x90\xb9\xbc\xa0\xdd\x92\x93g\xf8\xf8/\xf7\xa3\x1d\xb6\xc9\xac\xbf\x88\xfd\x12\xe06\xef\x97=d\xb7c\x97\xd7E\xf3[\xa2\xec\xc2\xb9D\xcbX\x80\xfa8\xc9\x9d\xd8\xbcwBP\xa7\xb5\xb3\x08Z\x13\x7f\xb8\x92 \xd0\x7f\xd1\x00+\x00\
05\x04\x03\x04\x03\x03\x00\r\x00\x18\x00\x16\x04\x03\x05\x03\x06\x03\x08\x04\x08\x05\x08\x06\x04\x01\x05\x01\x06\x01\x02\x03\x02\x01\x00-\x00\x02\x01\x01\x00\x1c\x00\x02@\x01\x00\x15\x00y\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' |>>>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=40 id=9031 flags= frag=0 ttl=64 proto=tcp chksum=0xe89e src=74.125.24.95 dst=10.0.2.15 |<TCP  sport=https
ort=55914 seq=191872002 ack=4059865147 dataofs=5 reserved=0 flags=A window=65535 chksum=0xb1a urgptr=0 |<Padding  load='\x00\x00\x00\x00\x00\x00' |>>>>>
<bound method Packet.summary of <Ether  dst=08:00:27:46:81:2b src=52:54:00:12:35:02 type=IPv4 |<IP  version=4 ihl=5 tos=0x0 len=1452 id=9032 flags= frag=0 ttl=64 proto=tcp chksum=0xe319 src=74.125.24.95 dst=10.0.2.15 |<TCP  sport=http
 dport=55914 seq=191872002 ack=4059865147 dataofs=5 reserved=0 flags=PA window=65535 chksum=0xb7da urgptr=0 |<Raw  load='\x16\x03\x03\x00z\x02\x00\x00v\x03\x03I\xa9$5\xf8\x97\x81`\xcdS\xa1\x1a\xbf\x81\xd2\x8c\x81\x15\x9d\x8e\xaa\xb9Q\
=1\xcef\x11v\x1b)\xe3g E\xafF\x956\x19*#{\x9e\x87\x06\xd7\xa6h\x9ch\xb5\x9b >6\xaa\x9d\xbfY\xac\xbb\xfcX\x81\xfb\x13\x01\x00\x00.\x003\x00$\x00\x1d\x00 \xda22fr#\xe7OgDg\x9a 9\x86F\x19\xda\x10\x9fr\xcb:.r\xd9\xa0\x8c\x8b\xf0\x0e \x00+
```

Fig 5.6: Sniffing Packets

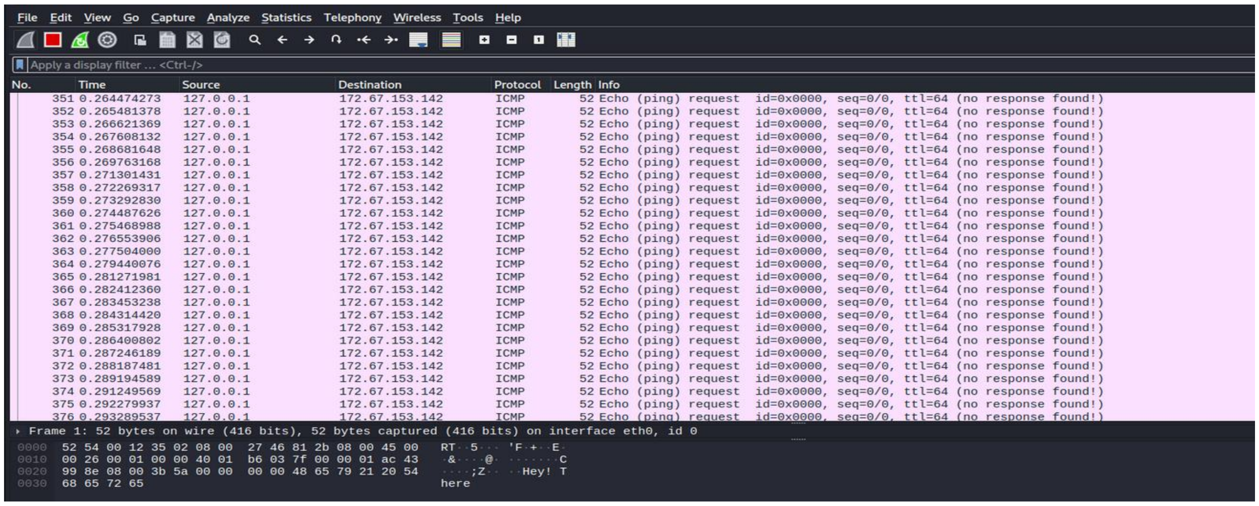
1.2 Wireshark:

Wireshark is an open source packet analyzer that is free to use. It is employed in the creation of software and communications protocols, network troubleshooting, analysis, and teaching. Simply said, Wireshark grabs data from a LAN line, from a live network connection, or from a file of previously collected packets. Wireshark is an open source programme for analysing network traffic. The most prevalent format is this one. Two filtering languages are available in Wireshark, one for capturing packets and the other for displaying them. Packets may be chosen based on the protocol, the existence of a field, its value, a comparison of fields, etc. The expression tab or the queries that can be typed in the field may be chosen to offer much more complex definitions and a list of all the protocols from a broad variety of protocols in Wireshark. [7]


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.     Time          Source               Destination          Protocol Length Info
   1945 1.508649308   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1946 1.508649472   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1947 1.509158597   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1948 1.509158931   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1949 1.509327636   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1950 1.509513696   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1951 1.509514054   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1952 1.509813317   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1953 1.509813621   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1954 1.510206135   RealtekU_12:35:02    Broadcast            ARP        60 Who has 127.0.0.1? Tell 10.0.2.2
   1955 202.630949063 10.0.2.15            182.19.95.34         DNS        69 Standard query 0x3085 A WORKGROUP
   1956 202.630973071 10.0.2.15            182.19.95.34         DNS        69 Standard query 0xf587 AAAA WORKGROUP
   1957 207.635084554 10.0.2.15            208.91.112.52        DNS        69 Standard query 0x3085 A WORKGROUP
   1958 207.635173462 10.0.2.15            208.91.112.52        DNS        69 Standard query 0xf587 AAAA WORKGROUP
   1959 207.736410788 PcsCompu_46:81:2b    RealtekU_12:35:02    ARP        42 Who has 10.0.2.2? Tell 10.0.2.15
   1960 207.737173570 RealtekU_12:35:02    PcsCompu_46:81:2b    ARP        60 10.0.2.2 is at 52:54:00:12:35:02
   1961 207.751792538 208.91.112.52        10.0.2.15            DNS       144 Standard query response 0x3085 No such name A WORKGROUP SOA a.root-servers.net
   1962 207.751793490 208.91.112.52        10.0.2.15            DNS       144 Standard query response 0xf587 No such name AAAA WORKGROUP SOA a.root-servers.net
   1963 207.758147705 10.0.2.15            10.0.2.255           NBNS       92 Name query NB WORKGROUP<1d>
   1964 208.011846201 10.0.2.15            10.0.2.255           NBNS       92 Name query NB WORKGROUP<1b>
   1965 208.266869343 10.0.2.15            182.19.95.34         DNS        69 Standard query 0x9b8d A WORKGROUP
   1966 208.266960759 10.0.2.15            182.19.95.34         DNS        69 Standard query 0x1690 AAAA WORKGROUP
   1967 209.002079814 182.19.95.34         10.0.2.15            DNS       144 Standard query response 0x1690 No such name AAAA WORKGROUP SOA a.root-servers.net
   1968 209.620804563 182.19.95.34         10.0.2.15            DNS       144 Standard query response 0x9b8d No such name A WORKGROUP SOA a.root-servers.net
   1969 209.621073147 10.0.2.15            10.0.2.255           NBNS       92 Name query NB WORKGROUP<20>

▸ Frame 1: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface eth0, id 0
0000  52 54 00 12 35 02 08 00  27 46 81 2b 08 00 45 00   RT··5··· 'F·+··E·
0010  00 26 00 01 00 00 40 01  b6 03 7f 00 00 01 ac 43   ·&····@· ·······C
0020  99 8e 08 00 3b 5a 00 00  00 00 48 65 79 21 20 54   ····;Z·· ··Hey! T
0030  68 65 72 65                                        here
```

Fig 5.7: Viewing Packets of Different Protocols


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

```
No.     Time          Source               Destination          Protocol Length Info
    351 0.264474273   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    352 0.265481378   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    353 0.266621369   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    354 0.267608132   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    355 0.268681648   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    356 0.269763168   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    357 0.271301431   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    358 0.272269317   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    359 0.273292830   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    360 0.274487626   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    361 0.275468988   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    362 0.276553906   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    363 0.277504000   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    364 0.279440076   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    365 0.281271981   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    366 0.282412360   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    367 0.283453238   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    368 0.284314420   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    369 0.285317928   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    370 0.286400802   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    371 0.287246189   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    372 0.288187481   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    373 0.289194589   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    374 0.291249569   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    375 0.292279937   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
    376 0.293289537   127.0.0.1            172.67.153.142       ICMP       52 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response found!)
▸ Frame 1: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface eth0, id 0
0000  52 54 00 12 35 02 08 00  27 46 81 2b 08 00 45 00   RT··5··· 'F·+··E·
0010  00 26 00 01 00 00 40 01  b6 03 7f 00 00 01 ac 43   ·&····@· ·······C
0020  99 8e 08 00 3b 5a 00 00  00 00 48 65 79 21 20 54   ····;Z·· ··Hey! T
0030  68 65 72 65                                        here
```

Fig 5.8: Viewing Packets of Same Protocols

1.3 Tcpdump:

It is a standard packet analyzer that operates from the command line. It enables the user to view and intercept TCP/IP and other packets that are sent across a network to which the machine is connected. Tcpdump is free software that is distributed under the BSD licence. The majority of Unix-like operating systems support tcpdump: Tcpdump utilises the libpcap library in those systems to collect packets. WinPcap, the Windows version of libpcap, is used by the port of tcpdump for Windows known as Win Dump.

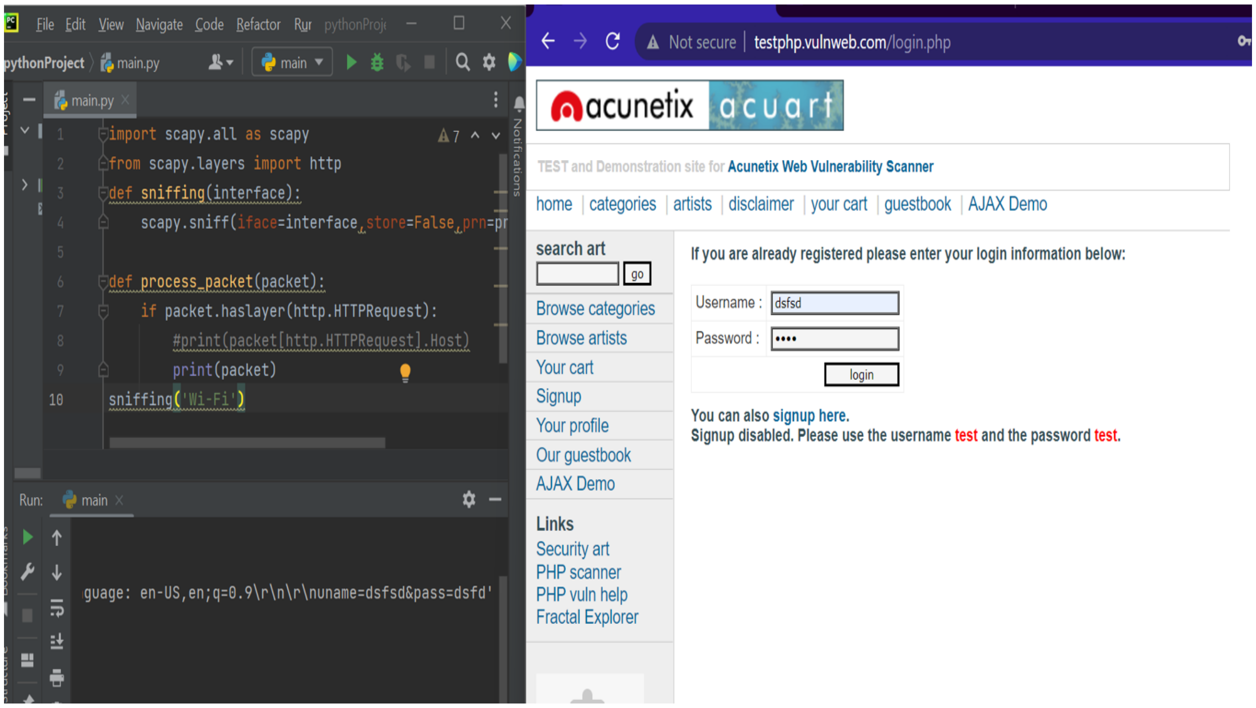
2.4 Ettercap:

Ettercap is a flexible network manipulating tool. Many of its other features are launched from its simple man-in-the-middle (MITM) attack capability in a switched LAN context. Once ettercap has entered a switched connection, it is able to record and evaluate all communication between the two victim hosts and then utilise the following features:

Packet filtering: Automatically filter the TCP or UDP payload of packets in a live connection by searching for an arbitrary ASCII or hexadecimal string, replacing it with your own string, or simply dropping the filtered packet.

Character injection: It is the practice of introducing random characters into an active connection in either way to simulate instructions from the client or responses from the server. [8]

RESULT AND DISCUSSION



```
import scapy.all as scapy
from scapy.layers import http
def sniffing(interface):
    scapy.sniff(iface=interface,store=False,prn=pr

def process_packet(packet):
    if packet.haslayer(http.HTTPRequest):
        #print(packet[http.HTTPRequest].Host)
        print(packet)
sniffing('Wi-Fi')
```

```
guage: en-US,en;q=0.9\r\n\r\nuname=dsfsd&pass=dsfd'
```

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

If you are already registered please enter your login information below:

Username : dsfsd
Password : ••••

You can also signup here.
Signup disabled. Please use the username test and the password test.

Fig 6.1: Capturing Username and Password from Vulnerable site

A packet sniffer is a tool for collecting packets, and a packet may include sensitive information like clear-text passwords or user identities. Here is an illustration of how to use the Scapy module to sniff login and password information from unsecure websites. However, it may also be employed maliciously to eavesdrop on private data transfers. This covers things like download history, email recipients, and online surfing history.

LIMITATIONS

- There are various tools available for capturing network traffic, however some of them have limitations. The researcher must utilise additional tools for analysis to obtain the necessary traffic characteristics and take the memory size of the system in use into account because some tools merely record network traffic without analysis while others demand huge memory sizes for installation.
- The traffic that is transferred between the machine and the destination is encrypted by a VPN. Only encrypted data that is transferred to your VPN service provider would be visible to a packet sniffer.
- Insecure http sites are where we are collecting usernames and passwords. Unlike https sites, which encrypt data transmission to prevent hackers or others from monitoring the connection from seeing it.

CONCLUSION AND FUTURE SCOPE

- The packet sniffer is not merely a tool for hackers. In addition to other helpful uses, it may be used for network traffic analysis, monitoring, and troubleshooting. Recent study has also shown that this project may be applied successfully in the field of education in addition to being valuable for network monitoring.
- Because of their adaptability, packet sniffers may be used to: Diagnose network issues. Identify attempts at network penetration. Obtain knowledge to carry out a network intrusion. Filter network traffic for questionable content. The system intrusion will be found by the intrusion detection module. focuses on securing a network against intrusion by unauthorised users.

APPENDIX

APPENDIX A: Common Cyberattacks:

- **Ransomware**: Malware called ransomware prevents authorised users from accessing their systems and demands a fee to get them back in.
- **Malware:** Malware, also known as malicious software, refers to any application or piece of code made with the intention of damaging a server, network, or computer.
- **DoS:** An intentional, targeted assault known as a denial-of-service (DoS) attack bombards a network with erroneous requests in an effort to interfere with company activities.
- **Phishing:** Phishing is a type of cyberattack that lures a victim into sharing sensitive information, like passwords or account numbers, or into downloading a malicious file that will infect their computer or phone with viruses using email, SMS, phone, social media, and social engineering techniques.
- **MITM:** A cyberattack known as a man-in-the-middle (MITM) attack involves a malicious attacker eavesdropping in on a conversation between a network user and a web application.
- **SQL Injection:** SQL Injection attack involves using system flaws to insert malicious SQL statements into a data-driven application, which then enables the attacker to extract data from a database. SQL Injection techniques are used by hackers to change, steal, or delete data.

APPENDIX B: Promiscuous Mode:

All network data packets can be seen and viewed by all network adapters operating in promiscuous mode, a particular sort of computer networking operational mode. Any specified network adapter on a host system may view full network data packets thanks to this technology for network security, monitoring, and management. In promiscuous mode, network communication is seen (sniffed). Promiscuous mode is also regarded as risky due to its capacity to access all network activity on a segment. This mode of operation is employed within the network for the purpose of packet sniffing, which is the technique of gathering and recording packets that traverse the network for later analysis, such as the examination of traffic or bandwidth utilisation. This mode is typically employed for packet sniffing on routers, computers linked to wired networks, and wireless local area networks.

APPENDIX C: Network Protocols:

Network protocols are guidelines that specify how experts transfer data across devices connected to the same network. Different types of Network Protocols are:

- **Transmission Control Protocol/Internet Protocol (TCP/IP)**: This protocol offers applications dependable delivery and makes sure that the message is delivered on schedule, accurately, and without duplicates.
- **User Datagram Protocol (UDP):** UDP is a communication protocol that replaces Transmission Control Protocol and is primarily used to link various applications in a loss-tolerant and low-latency manner.
- **File Transfer Protocol (FTP):** FTP enables users to move files between computers. Program files, multimedia files, text files, and documents, among other file types, are examples of file types.
- **Hypertext transfer protocol (HTTP):** HTTP is a protocol that enables the sharing of data like text files, images, and videos over the internet in distributed and collaborative hypermedia information systems.
- **Hypertext transfer protocol secure (HTTPS):** This protocol functions similarly to HTTP but employs encryption to guarantee secure data transmission over a network like the internet.
- **Internet control message protocol (ICMP):** This protocol has the ability to transmit error messages and operational data to devices or networks. They can alert users to errors and help with diagnostic procedures.
- **Telnet:** Telnet is a set of guidelines for establishing connections between systems. Here, the method of connecting is known as remote login.
- **ARP:** The ARP protocol aids in the mapping of logical addresses to the recognised physical addresses in a local network. A database called the ARP cache is employed for mapping and preserving a correlation between these logical and physical addresses.
- **DHCP:** DHCP is a protocol for network administration that is used to automate the configuration of devices on IP networks.

BIBLIOGRAPHY

[1] S. Ansari, Rajeev S.G. and Chandrasekhar H.S, “Packet Sniffing: Brief Introduction”, IEEE Potentials, Dec 2002- Jan 2003, Volume: 21 Issue: 5, pp: 17 – 19.

[2] Dr. Aruna Varanasi, P. Swathi. Comparative Study of Packet Sniffing tools for HTTP Network Monitoring and Analyzing. IJCSET(www.ijcset.net) | December 2016 | Vol 6, Issue 12,406-409

[3] Shivam Kumar , Suryansh Jigyasu , Vikas Singh. Network Traffic Monitor and Analysis Using Packet Sniffer. International Journal of Research in Engineering and Science (IJRES). Volume:9 Issue:7 ǁ 2021 ǁ pp: 77-82.

[4] Ryan Spangler. Packet Sniffing on Layer 2 Switched Local Area Networks. Packetwatch Research. December 2003.

[5] Jhilam Biswas, Ashutosh. An Insight in to Network Traffic Analysis using Packet Sniffer. International Journal of Computer Applications (0975 – 8887) Volume 94 – No 11, May 2014.

[6] Tom King. Packet Sniffing In a Switched Environment. GSEC Practical v1.4, Option 1 August 4th 2002, updated June/July 2006.

[7] Usha Banerjee, Ashutosh Vashishtha, Mukul Saxena. Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection. International Journal of Computer Applications (0975 – 8887) Volume 6– No.7, September 2010.

[8] Duane Norton. An Ettercap Primer. GIAC Security Essentials Certification Practical Assignment Version 1.4b Option 1 April 14, 2004.