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.

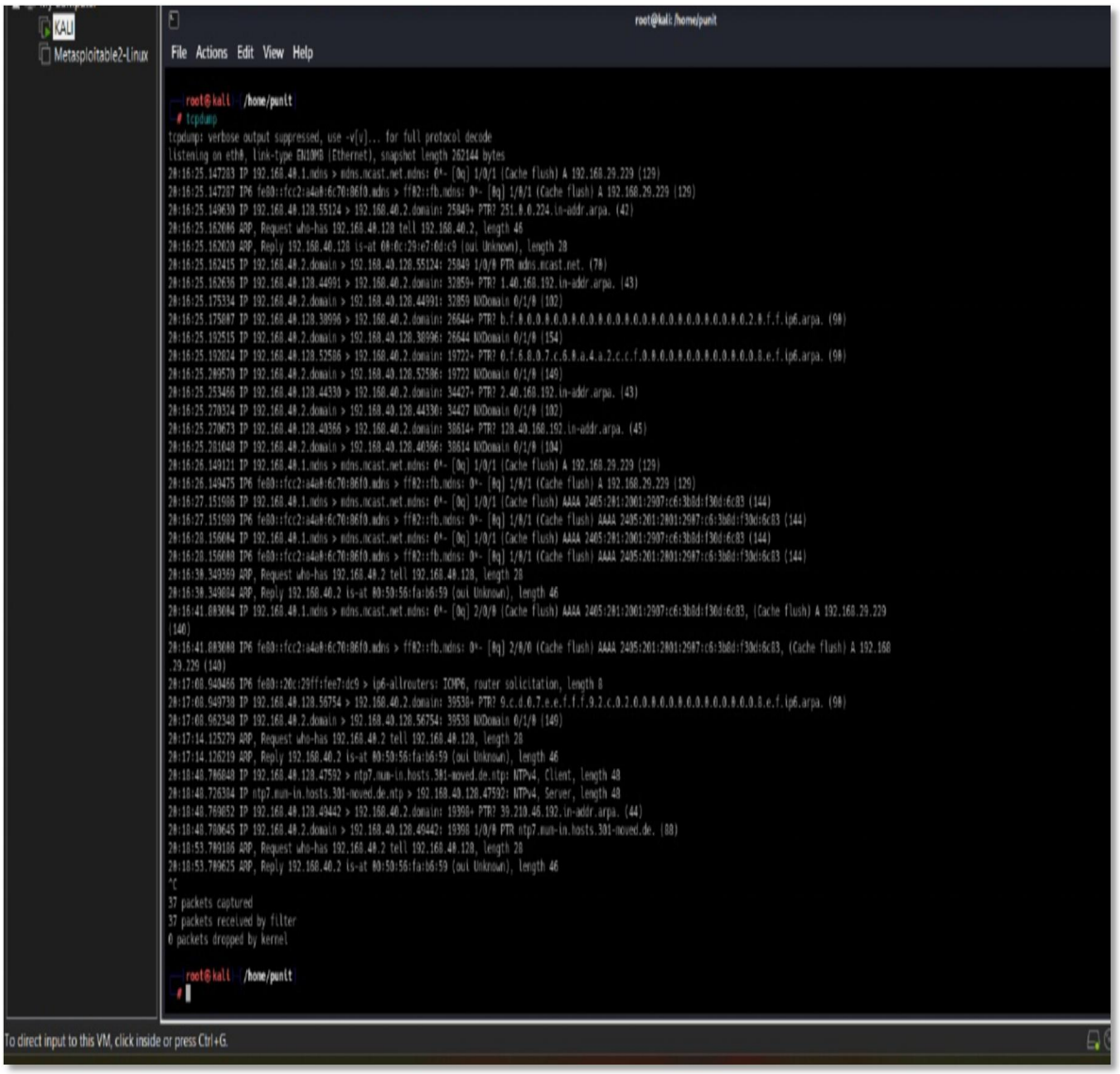

Figure 1

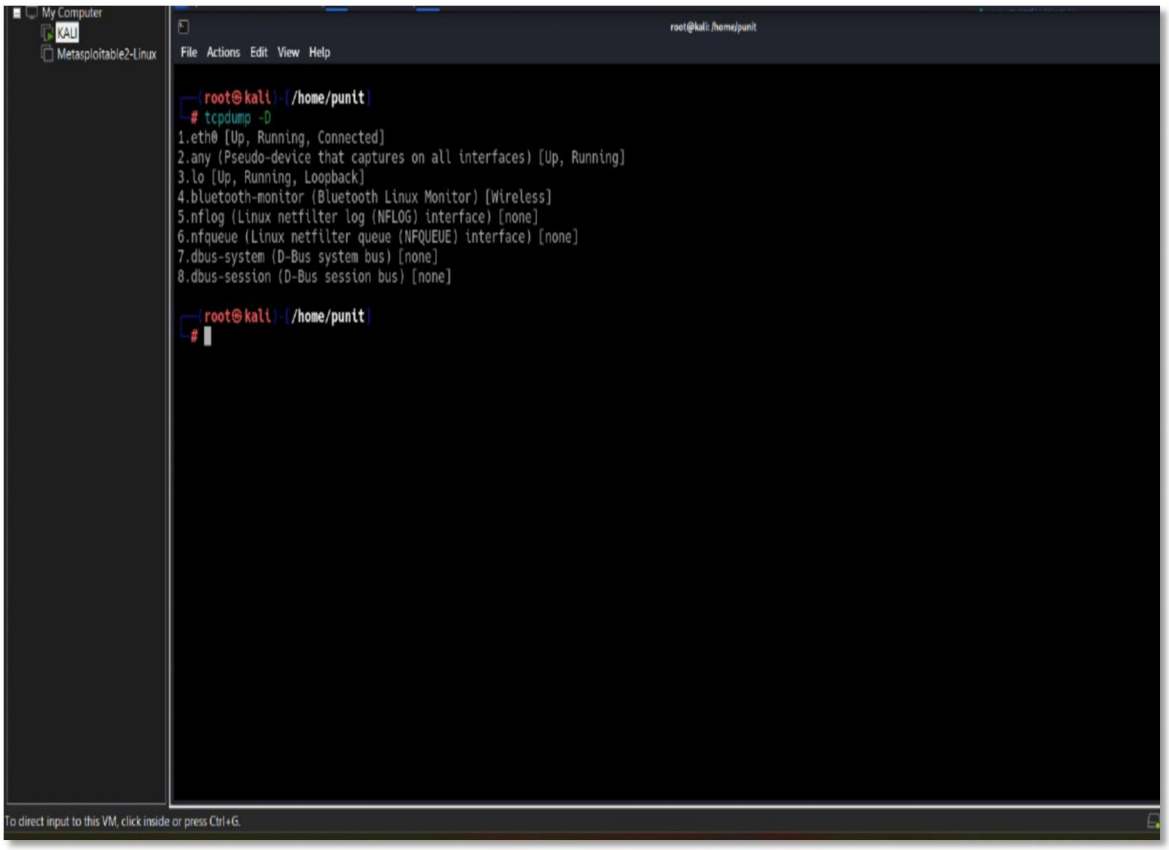

Figure 2

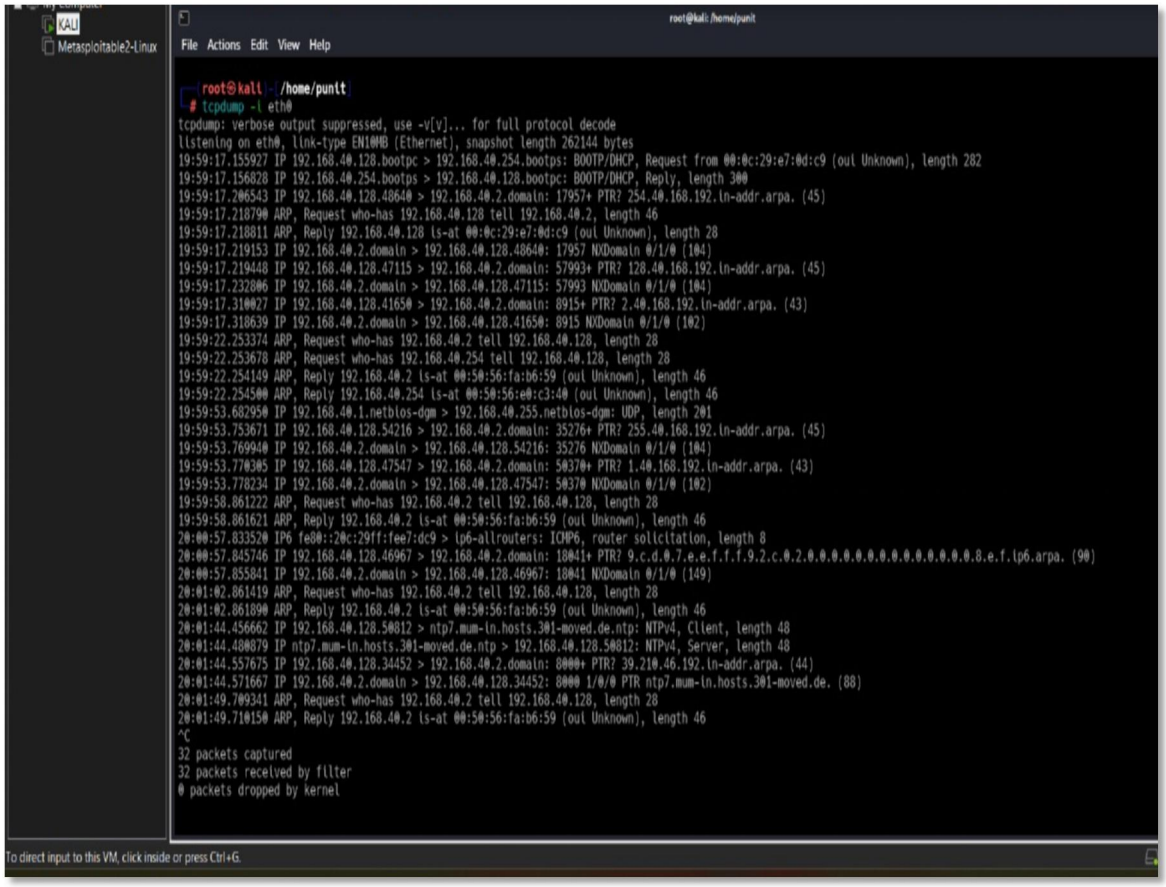

Figure 3

NETSTAT:

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

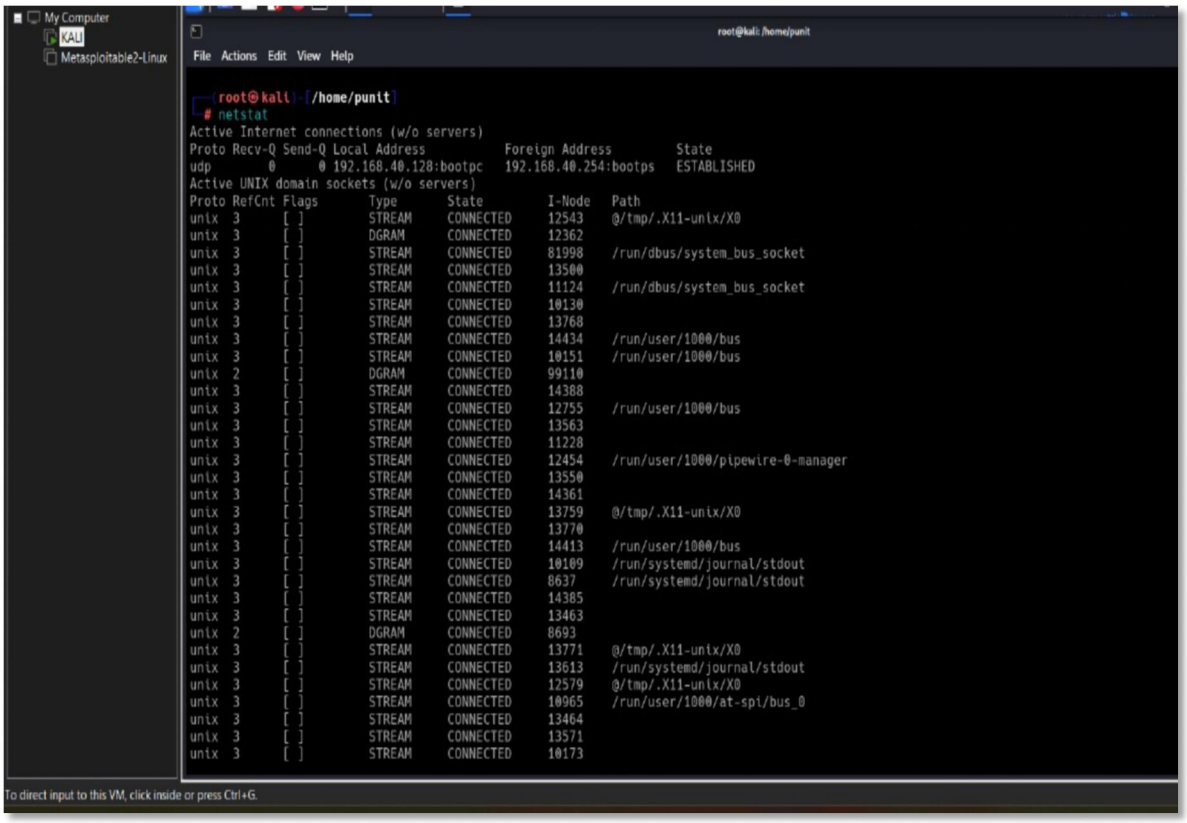

Figure 4

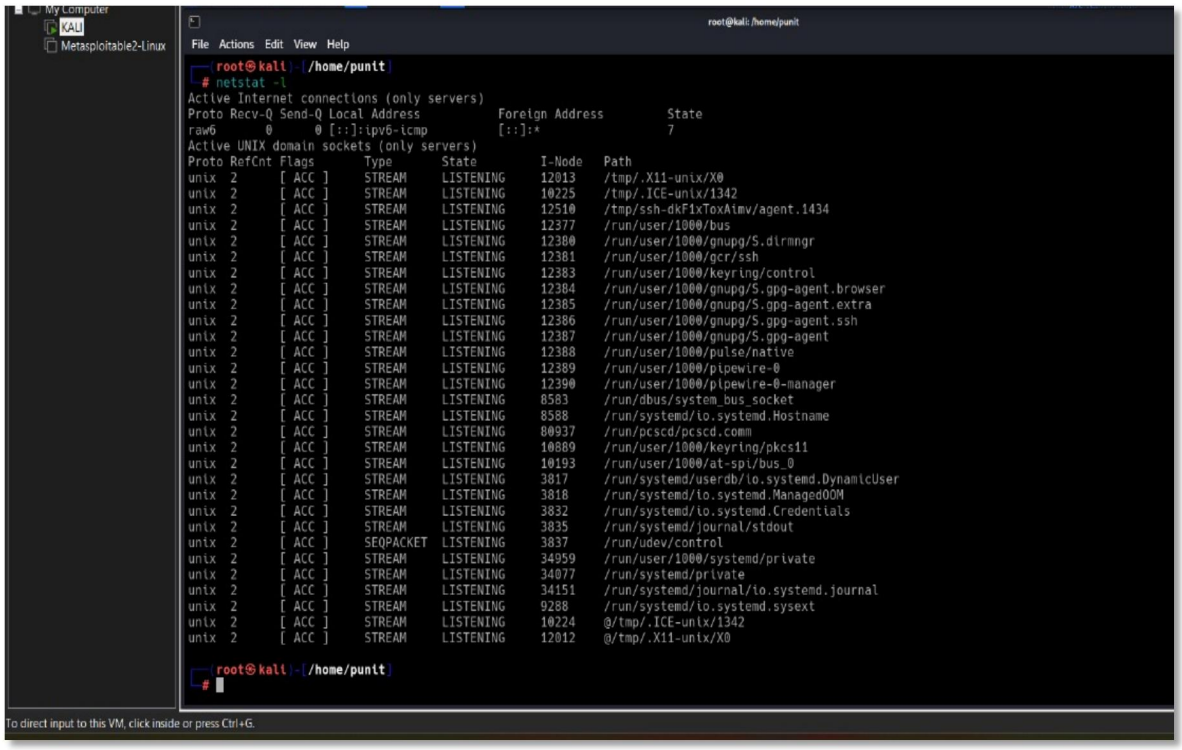

Figure 5

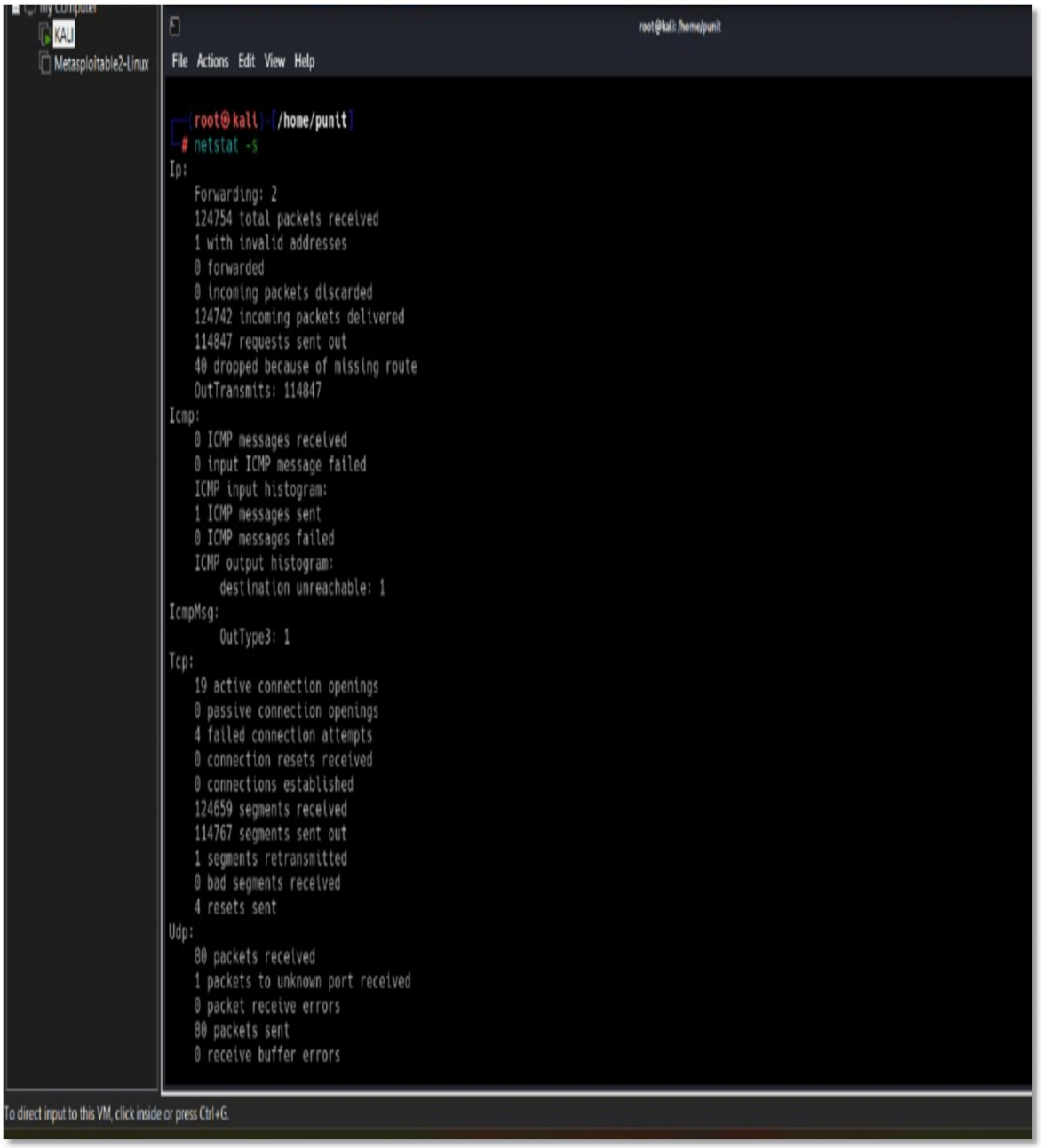
KALI
Metasploitable2-Linux
root@kali: /home/punit
File Actions Edit View Help
┌──(root@kali)-[/home/punit]
└─# netstat -s
Ip:
Forwarding: 2
124754 total packets received
1 with invalid addresses
0 forwarded
0 incoming packets discarded
124742 incoming packets delivered
114847 requests sent out
40 dropped 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.

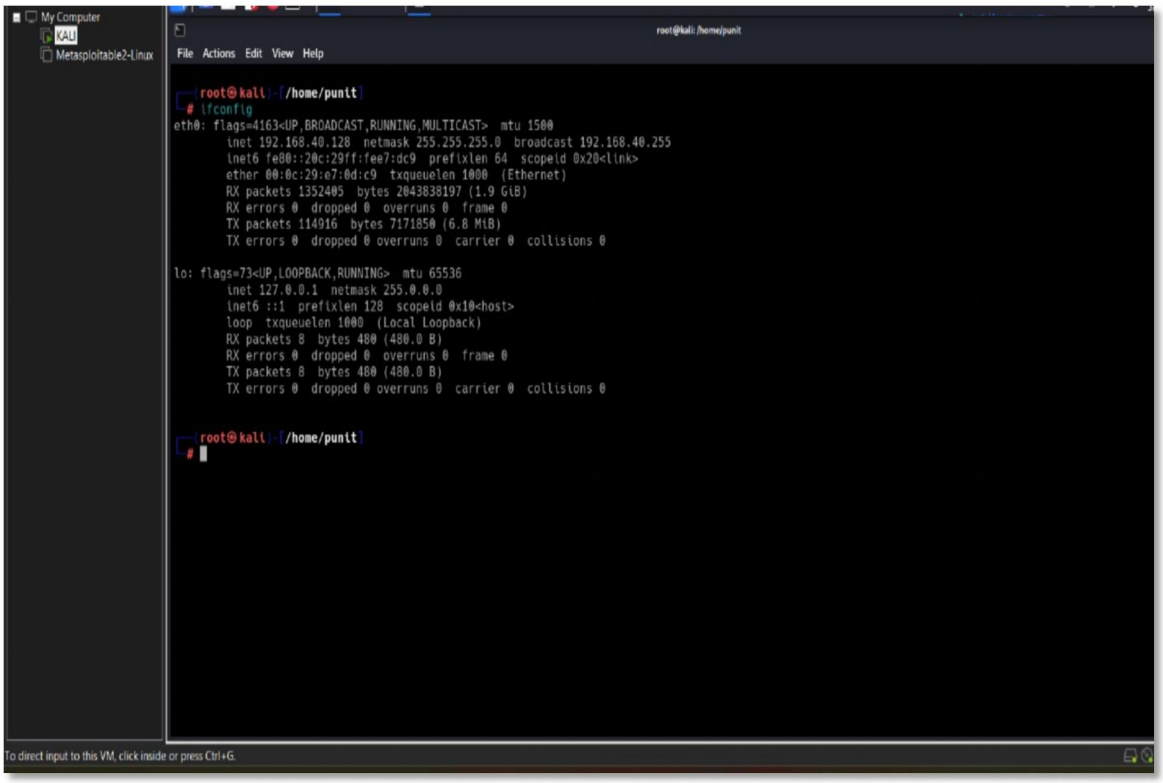

Figure 7

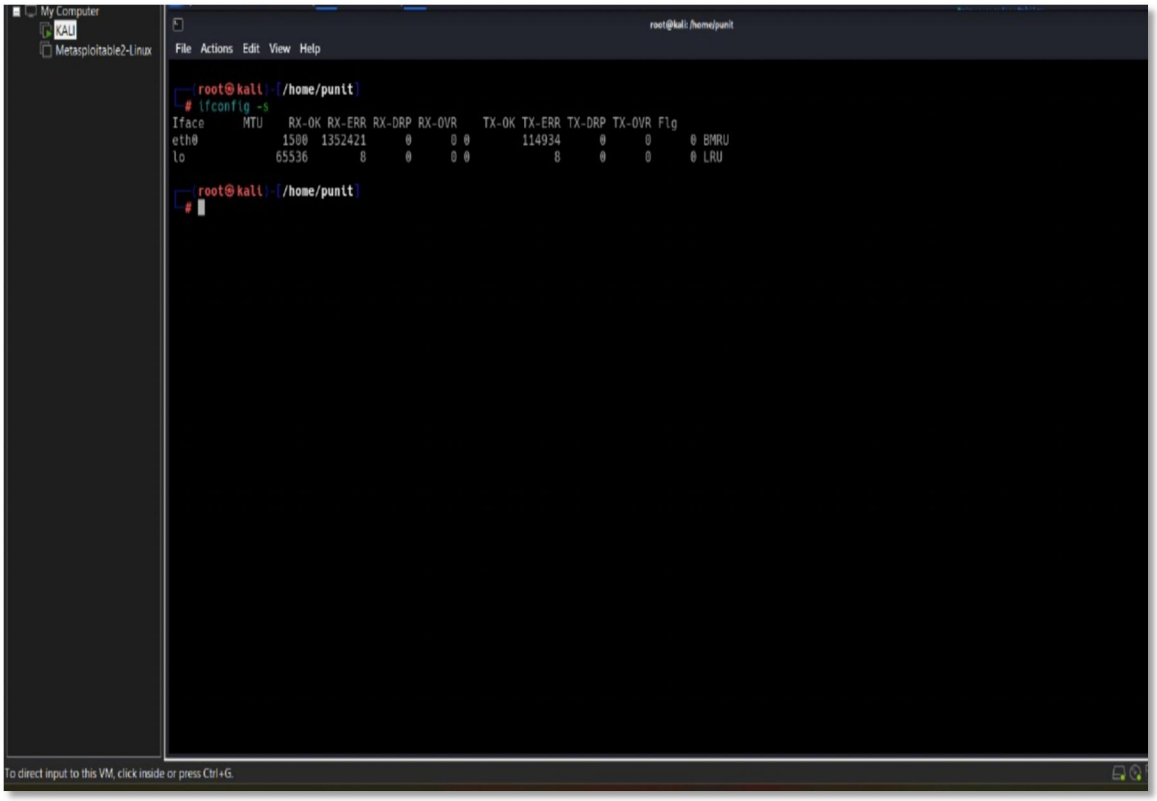

Figure 8

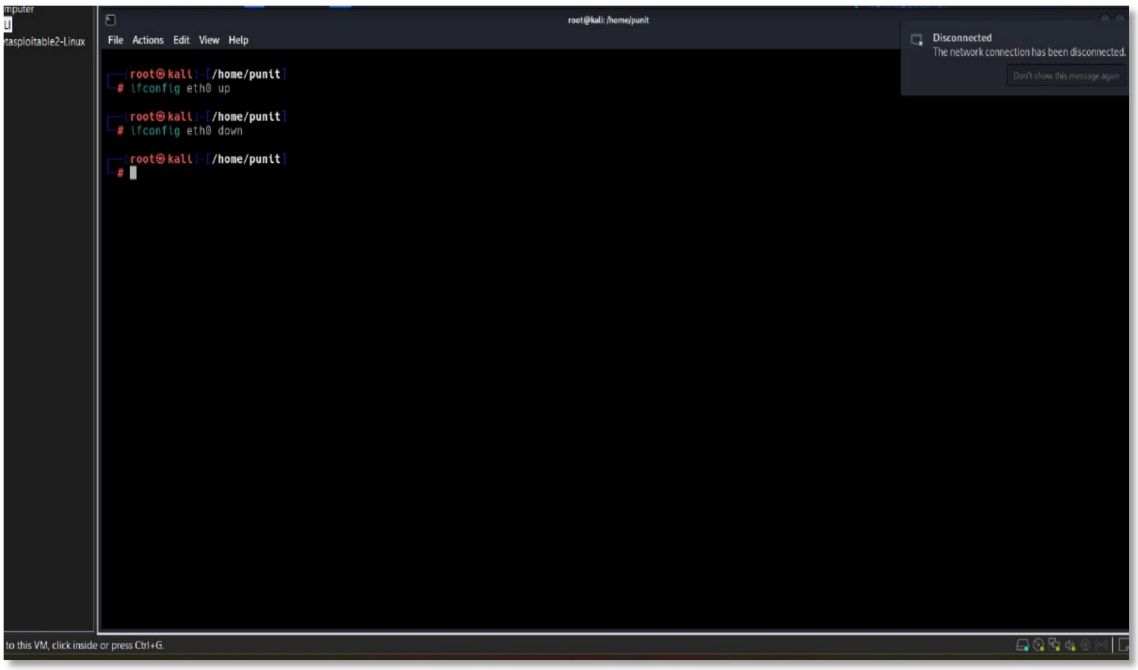

Figure 9

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).

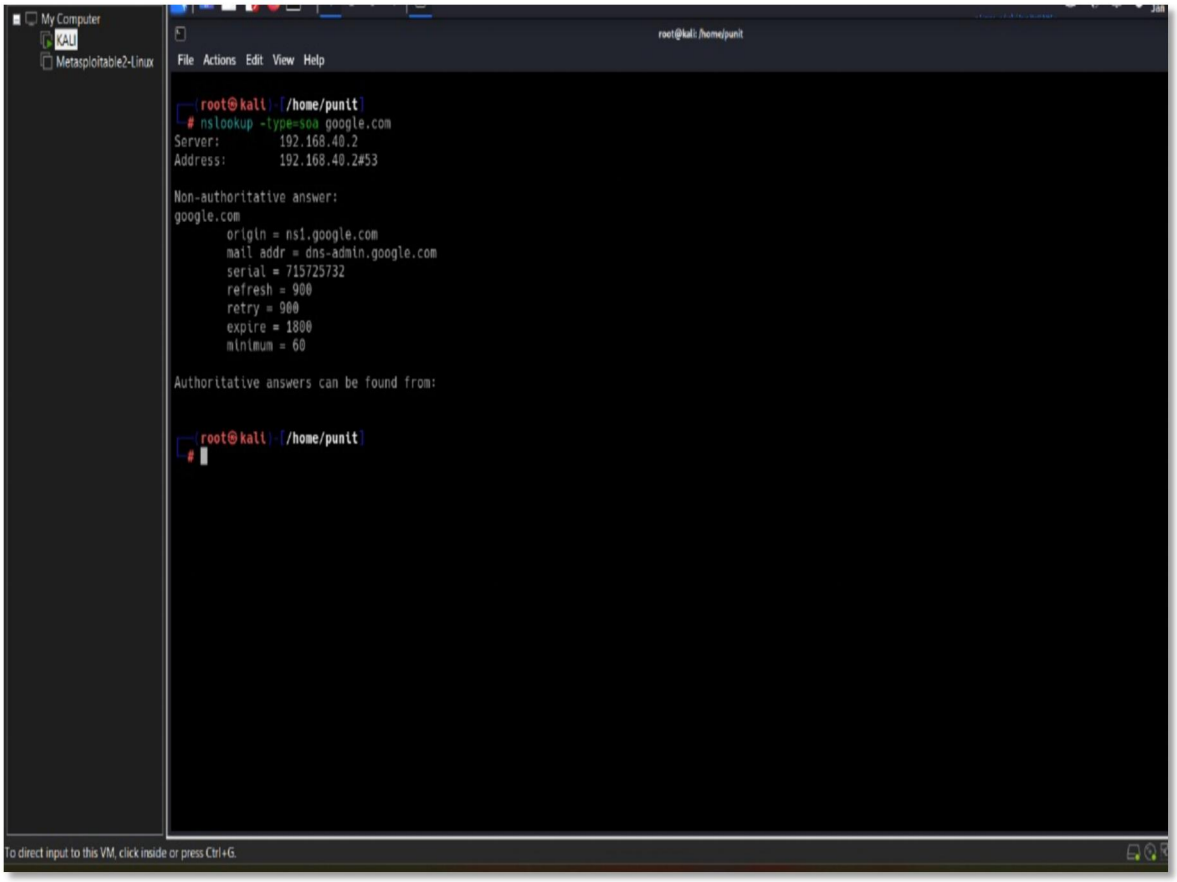

Figure 10

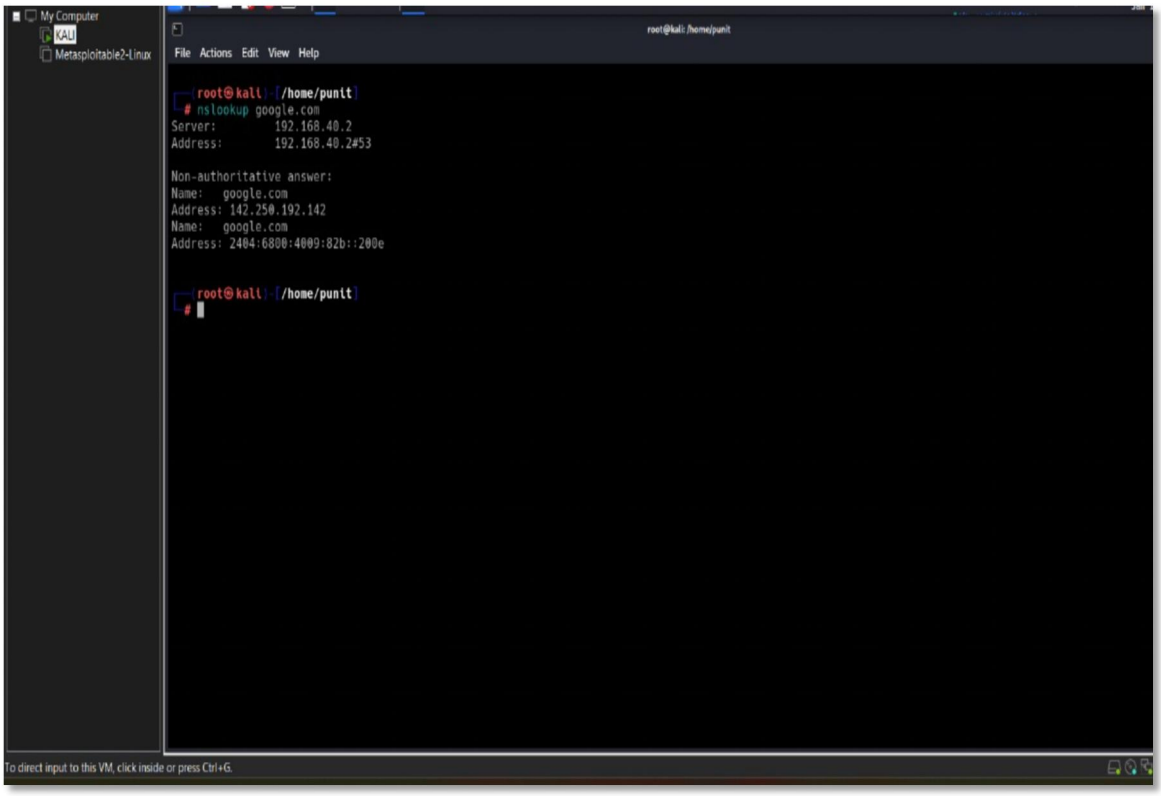

Figure 11

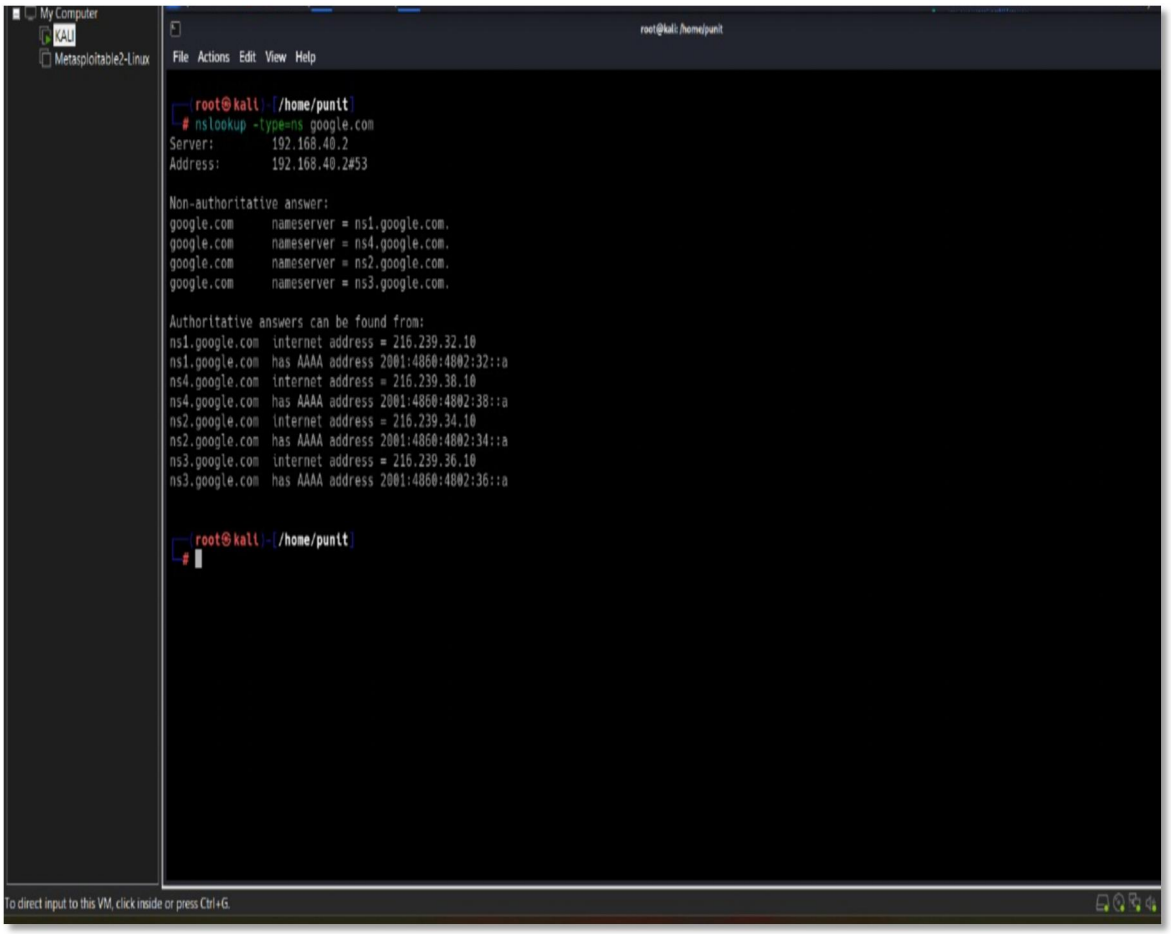

Figure 12

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.

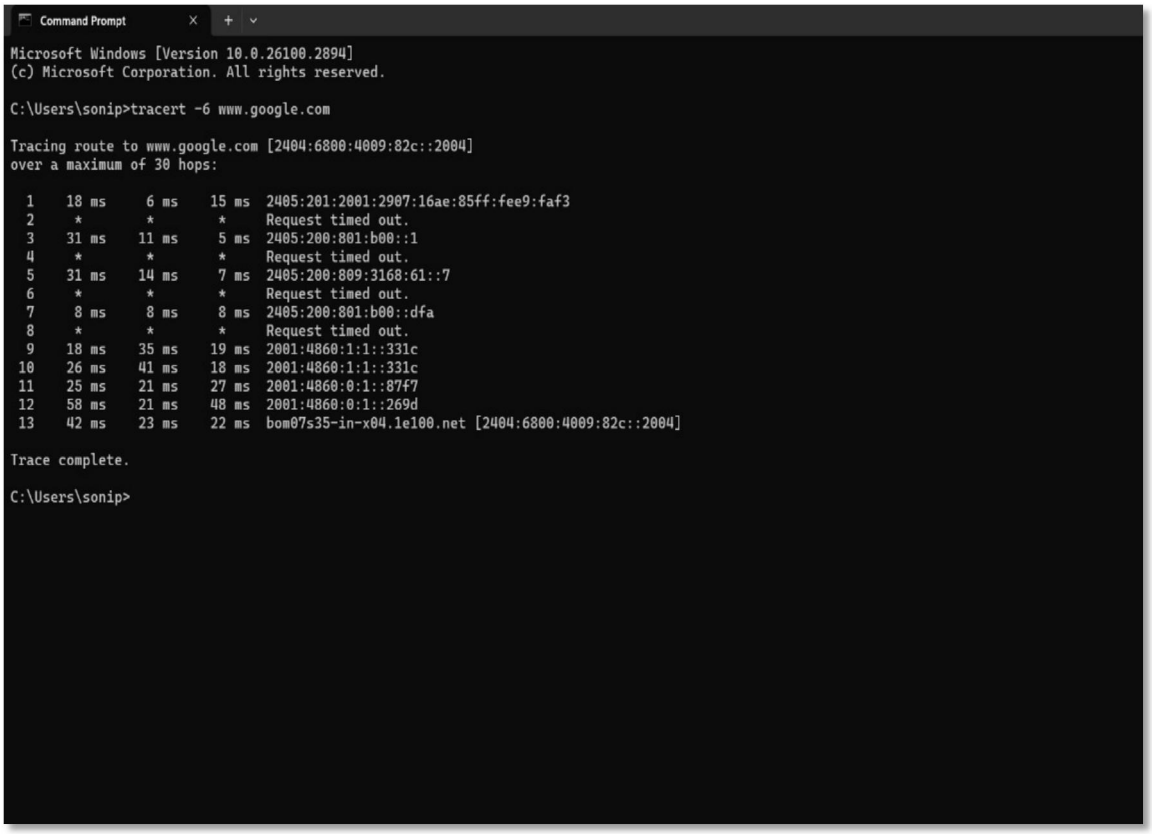

Figure 13

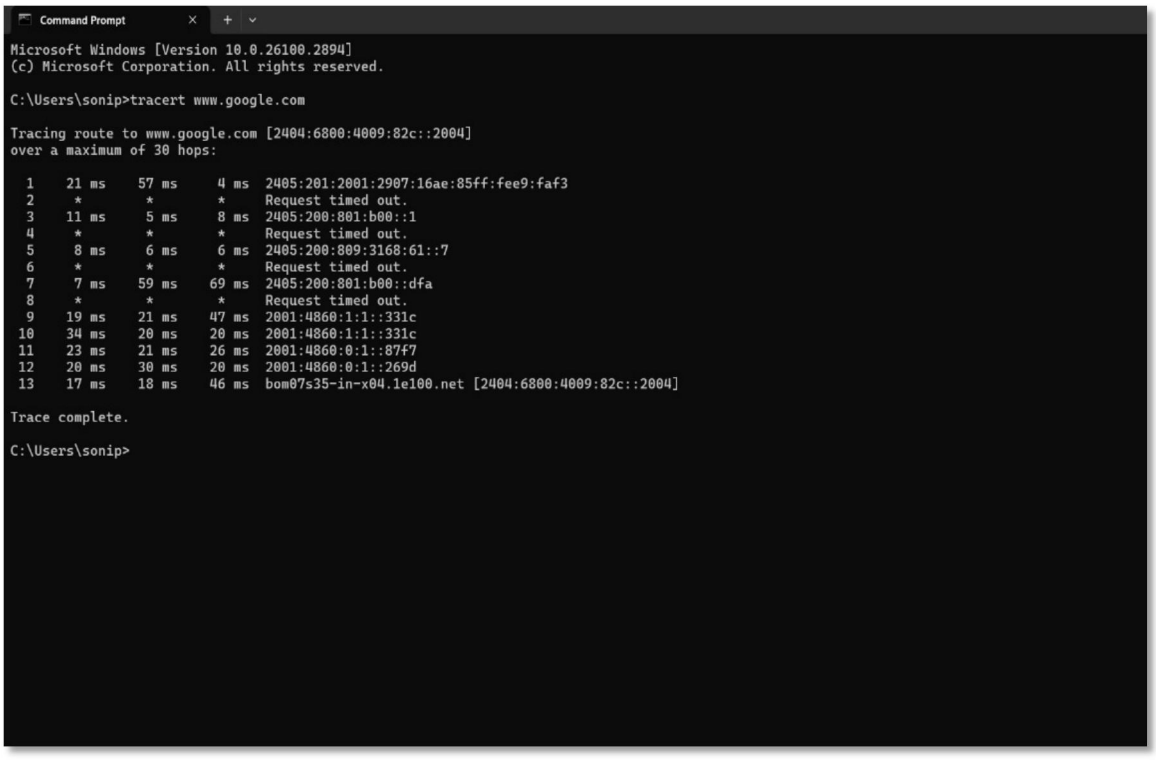

Figure 14

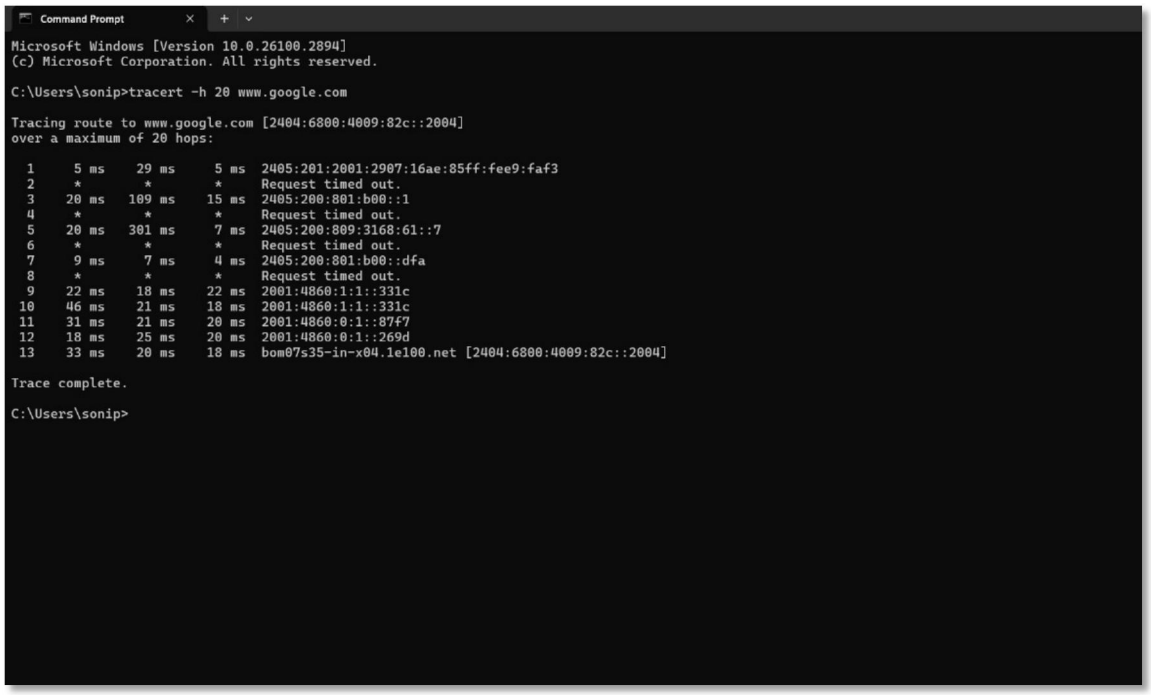

Figure 15

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.

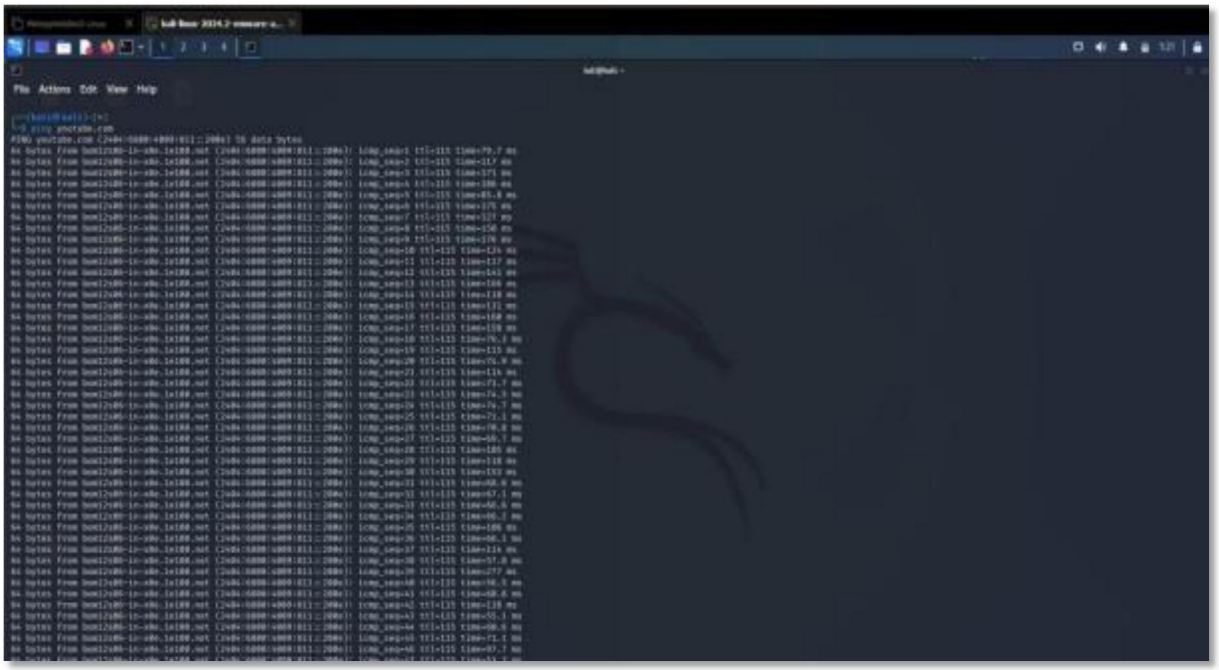

Figure 16

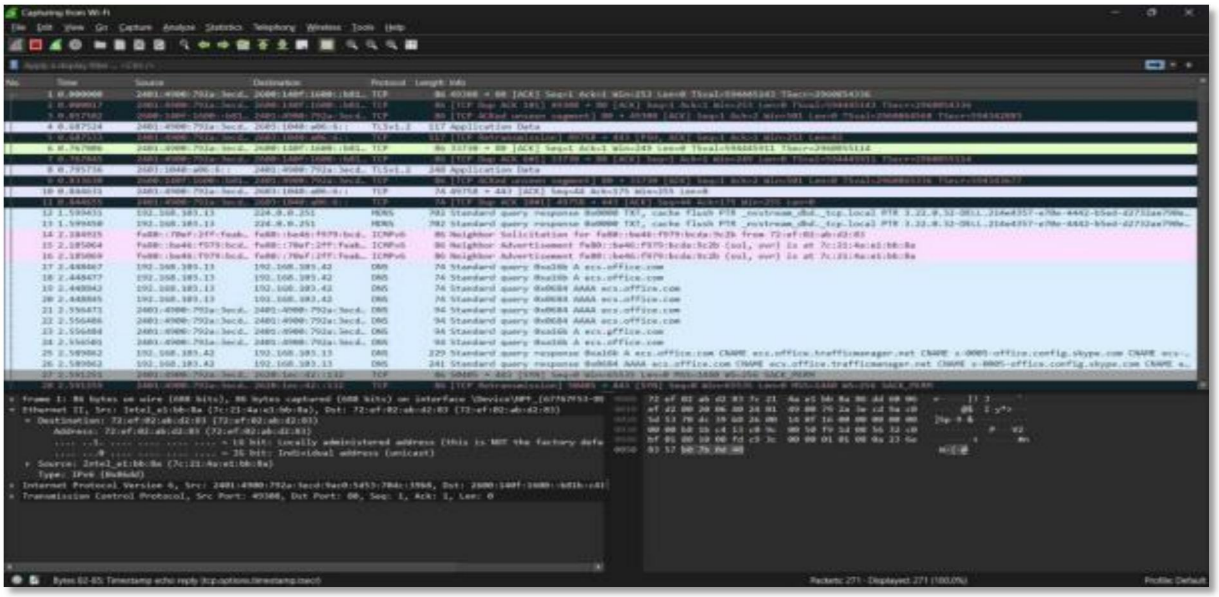

Figure 17

Step 2:

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

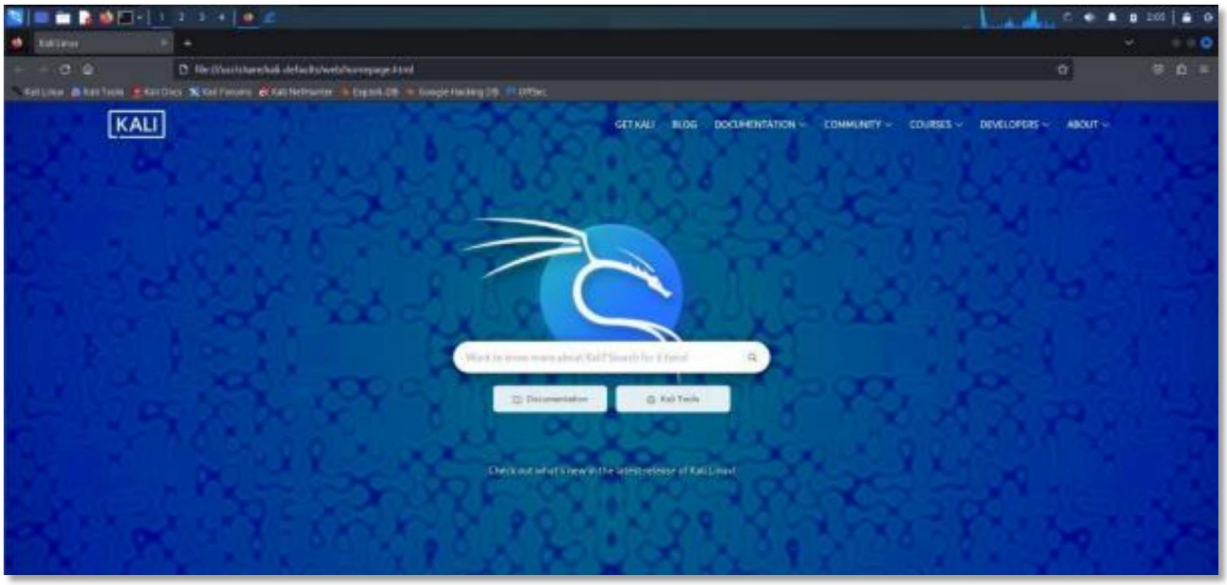

Figure 18

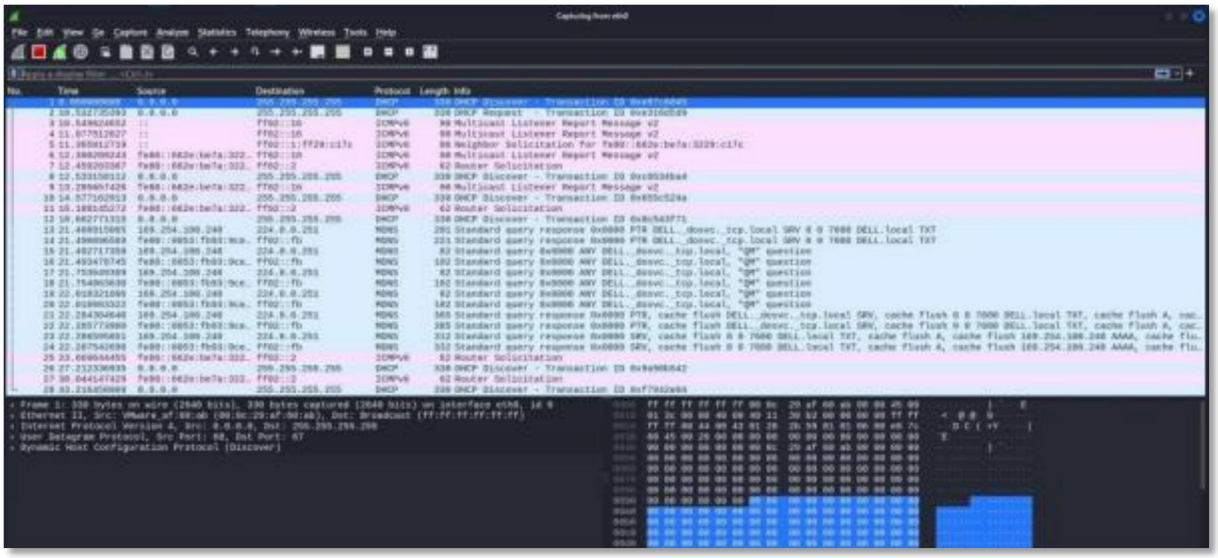

Figure 19

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.

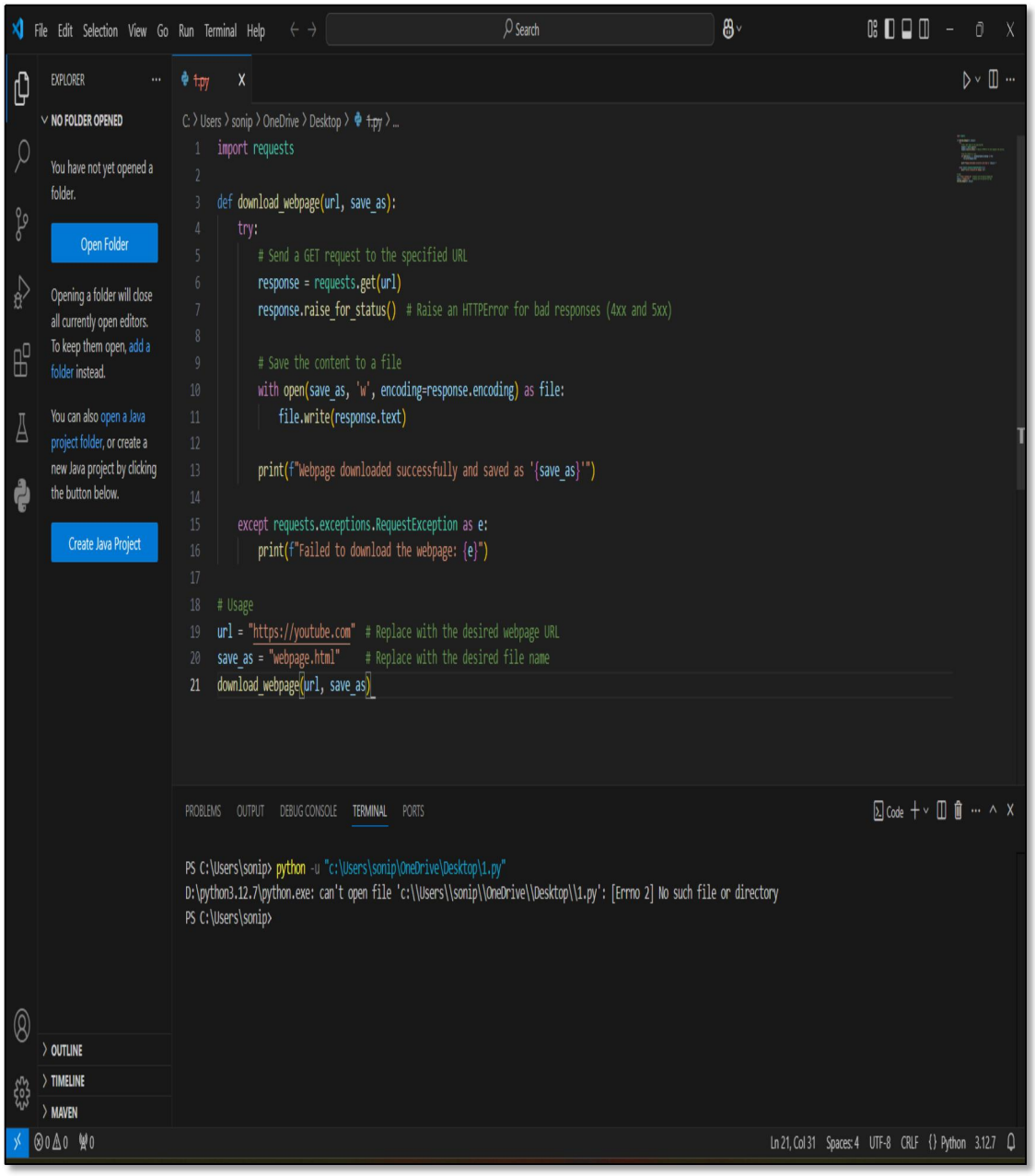

Figure 20

Output:

Cloned Webpage of Google and Youtube:

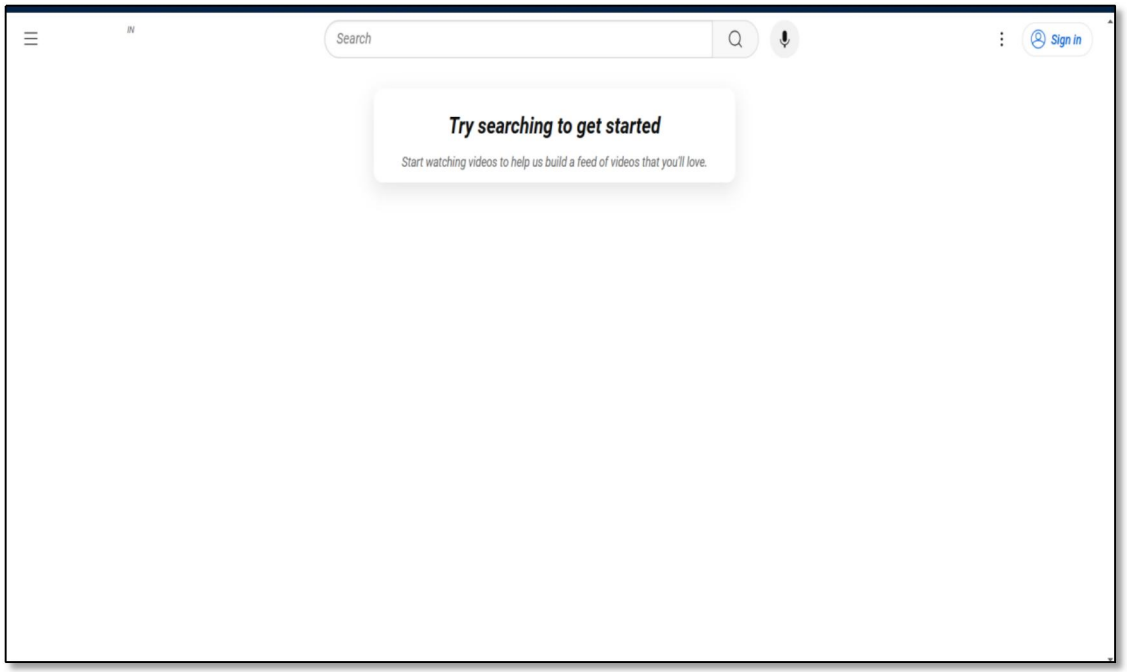

Figure 21

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 'I' 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 = QPIAUY

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.

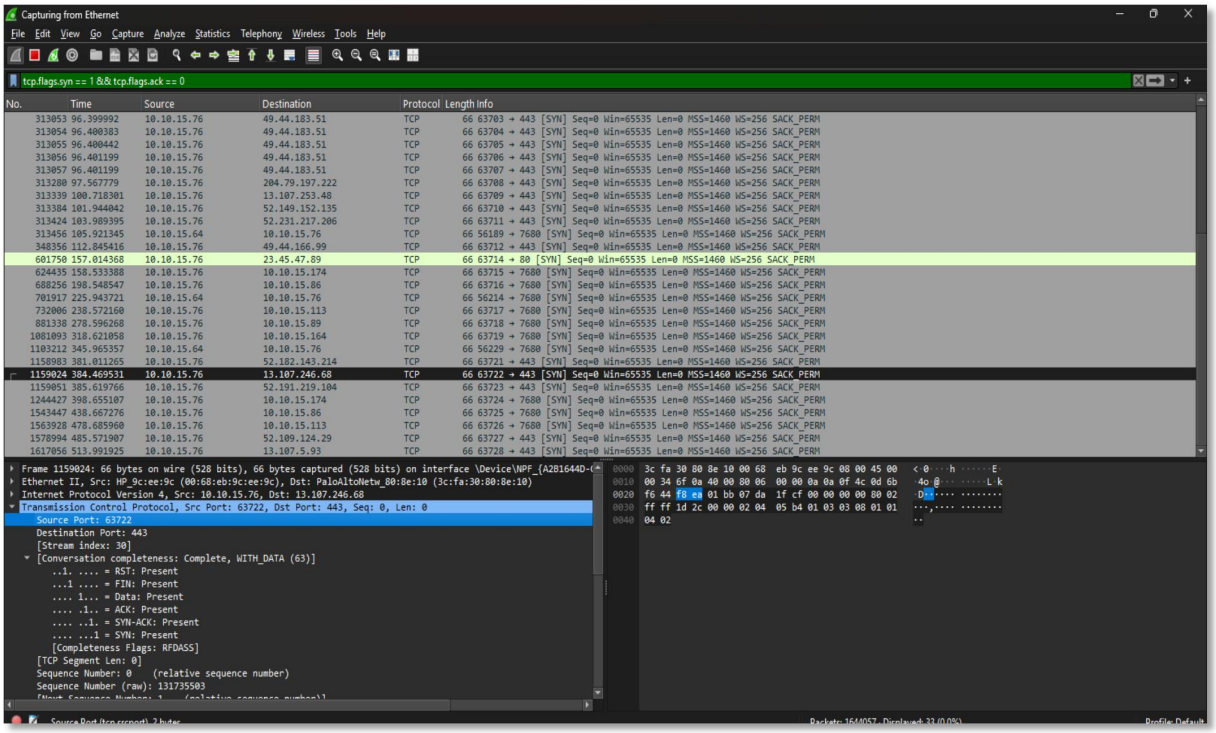

Figure 23

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).

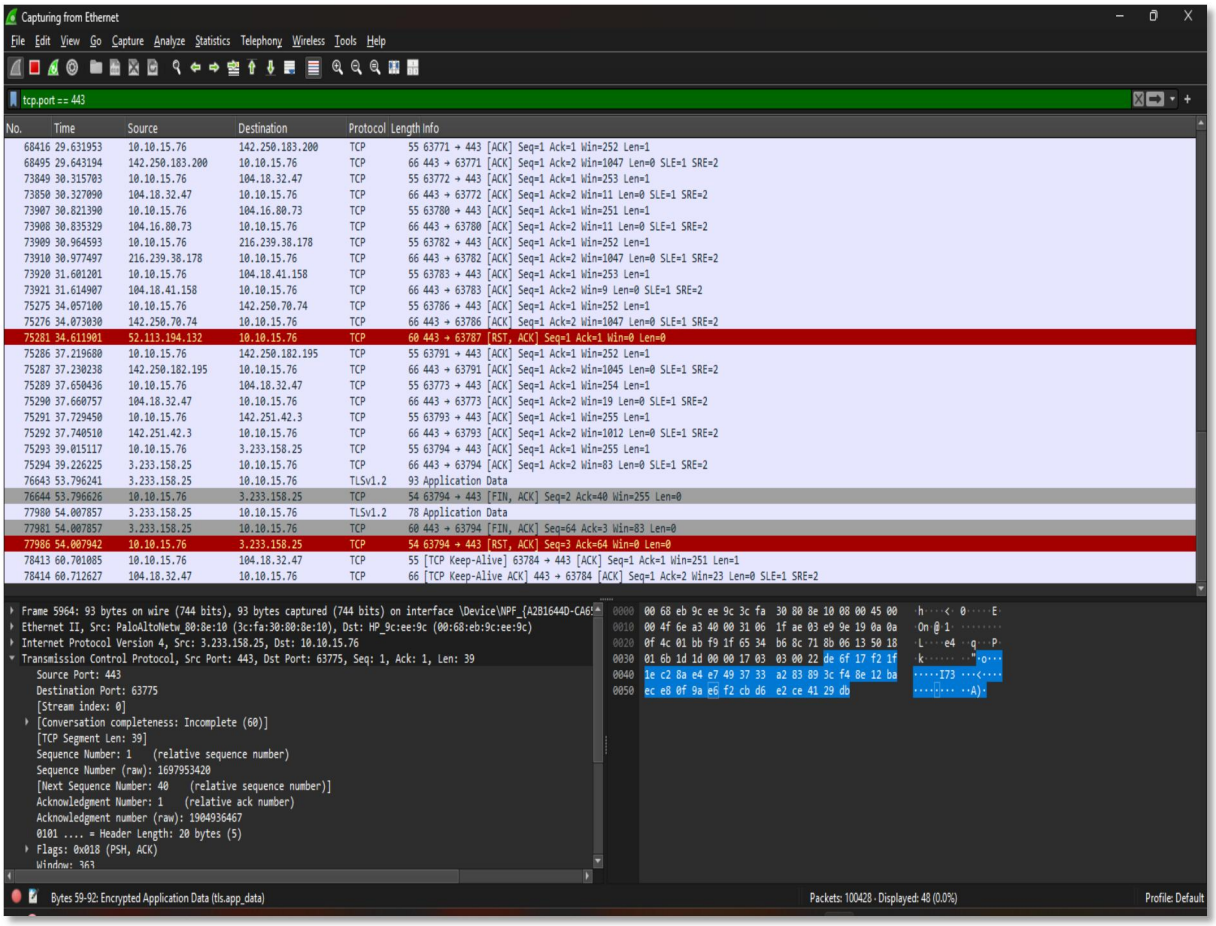

Figure 24

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

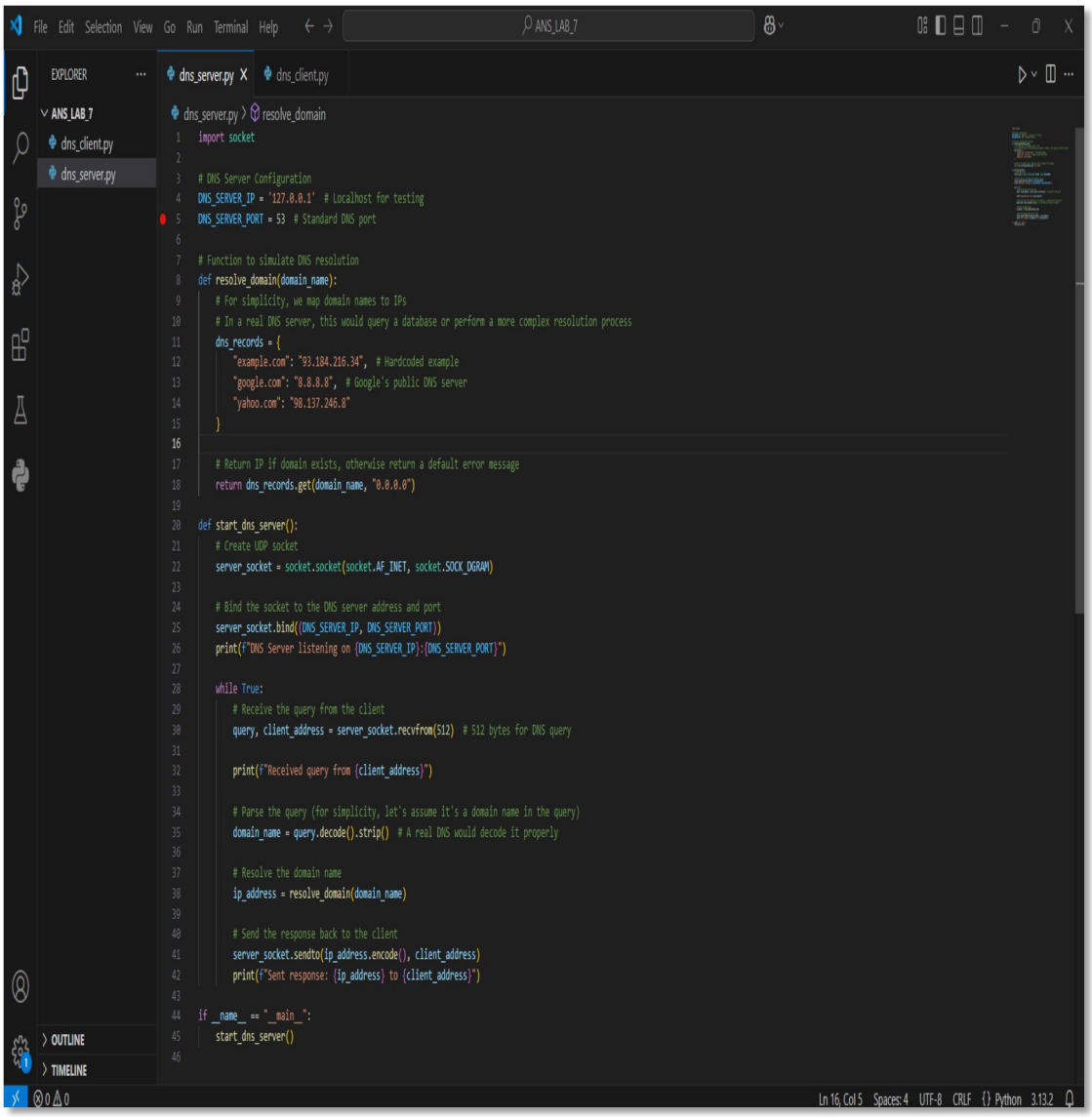

Figure 25

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

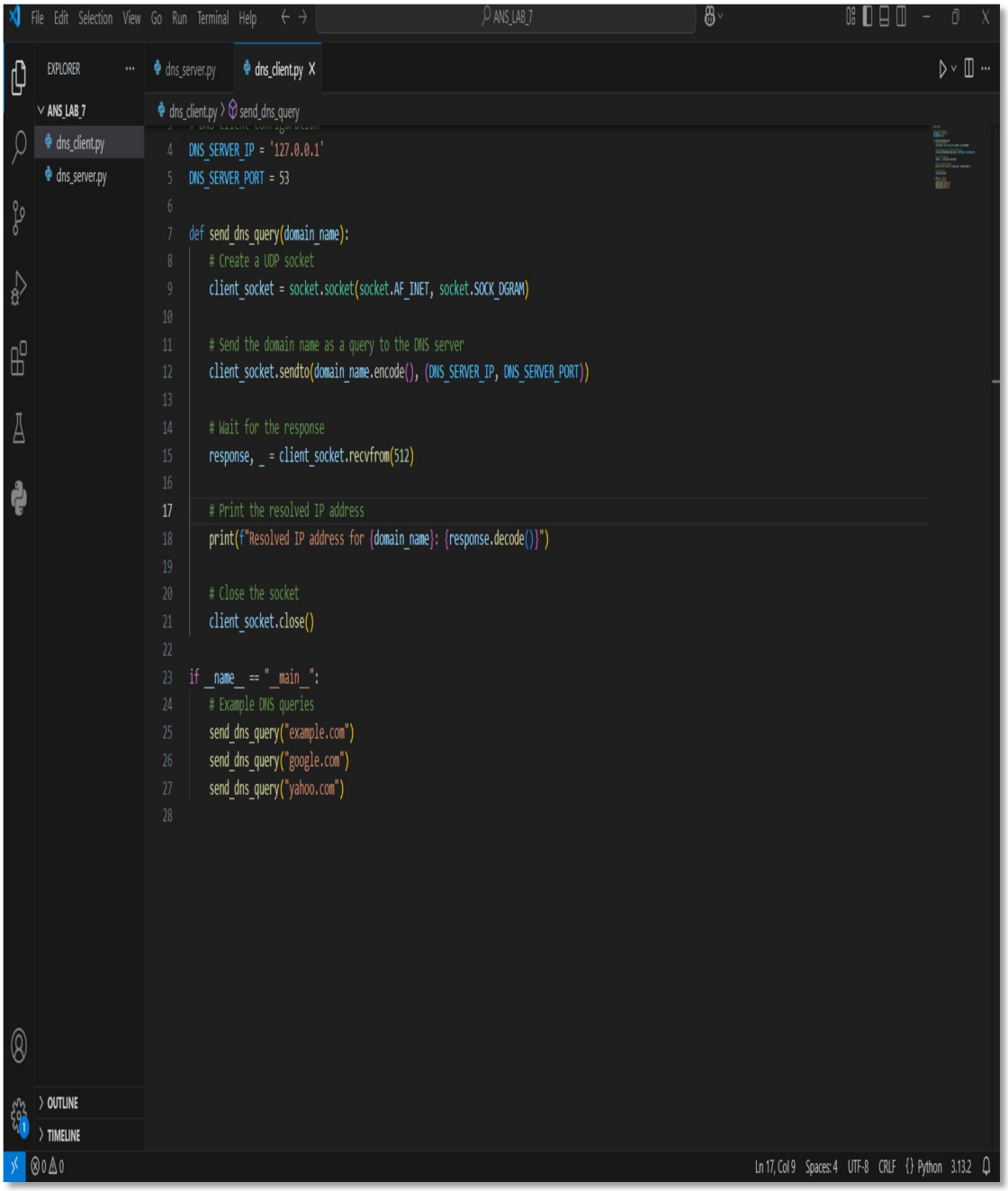

Figure 26

Output:

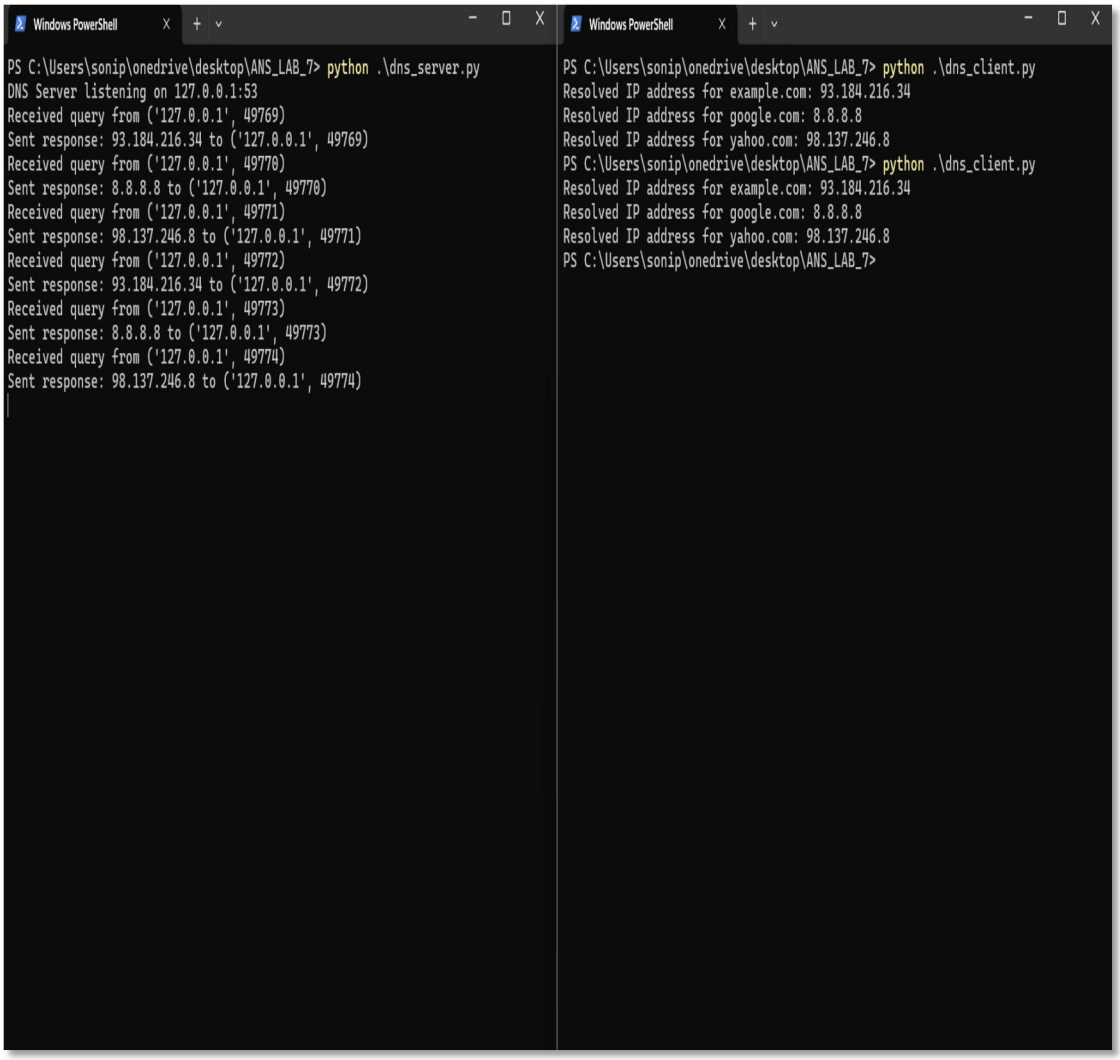

Figure 27

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.

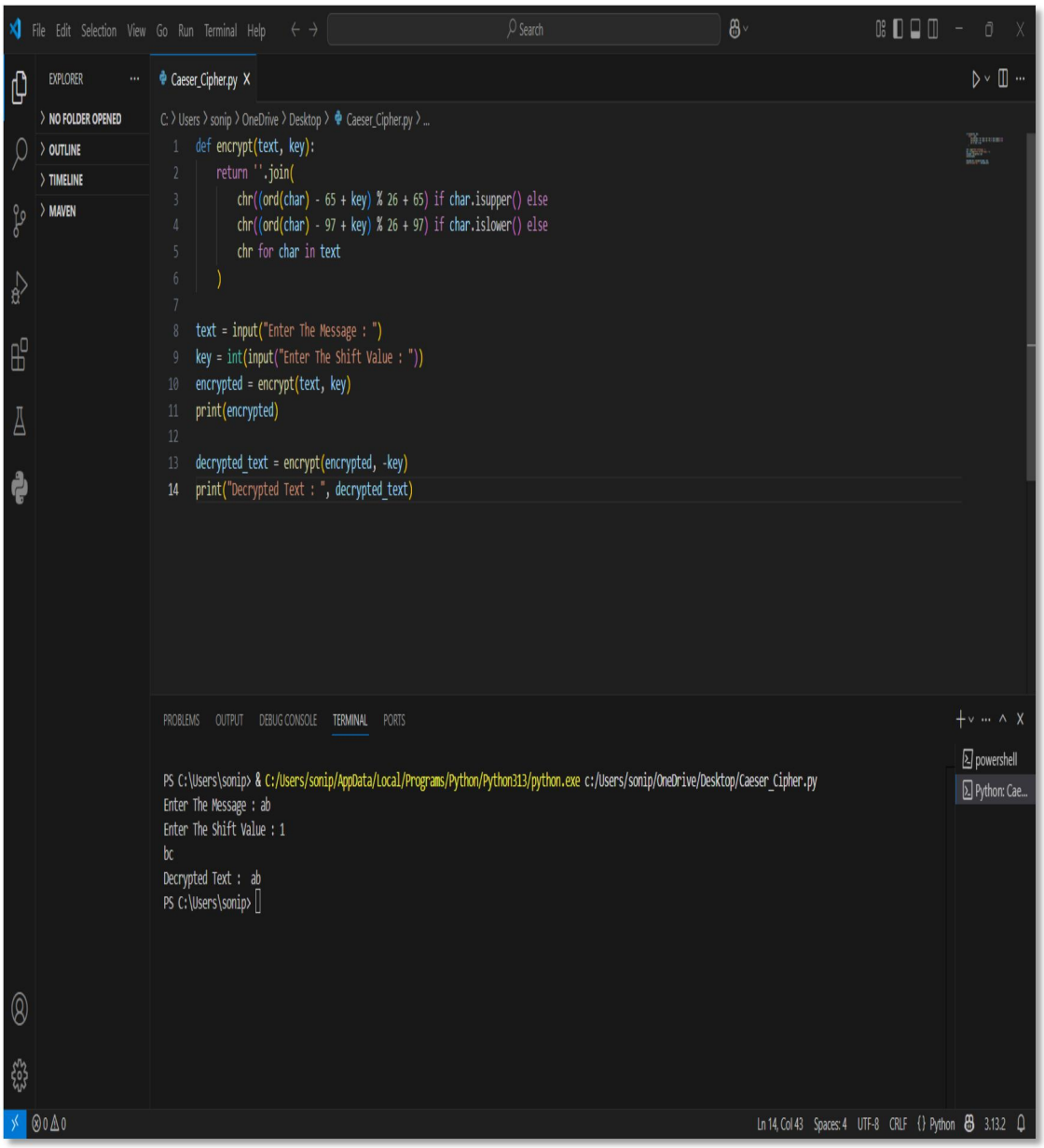

Figure 28

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:

p = 3
q = 11

2. Compute n and φ(n):

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

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

3. Choose a public key e:

e must be **coprime with φ(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 φ(n)

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

Try values:

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

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 mod n = 4^3 mod 33 = 64 mod 33 = 31

Ciphertext = 31

6. Decrypt the message:

Decryption formula:
M = C^d mod n = 31^7 mod 33

Let's calculate:

31^2 = 961 → 961 mod 33 = 4

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

31^7 = 31 × 31^2 × 31^4 = 31 × 4 × 16 = 1984

Now 1984 mod 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:

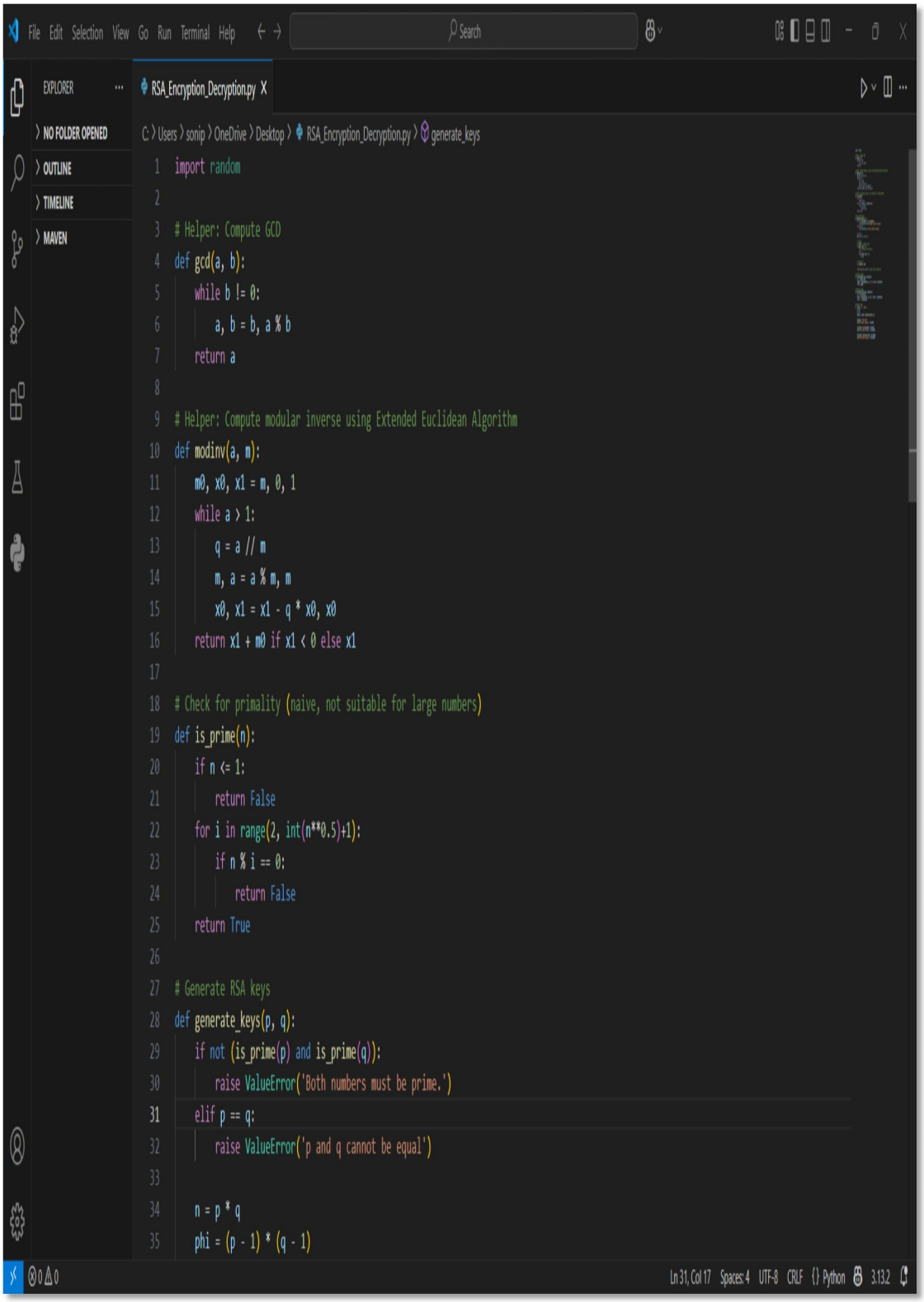

Figure 29


File Edit Selection View Go Run Terminal Help
Search
EXPLORER
RSA_Encryption_Decryption.py
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
C: > Users > sonip > OneDrive > Desktop > RSA_Encryption_Decryption.py > generate_keys
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)
Ln 46, Col 23 Spaces: 4 UTF-8 CRLF {} Python 3.13.2

Figure 30

Output:

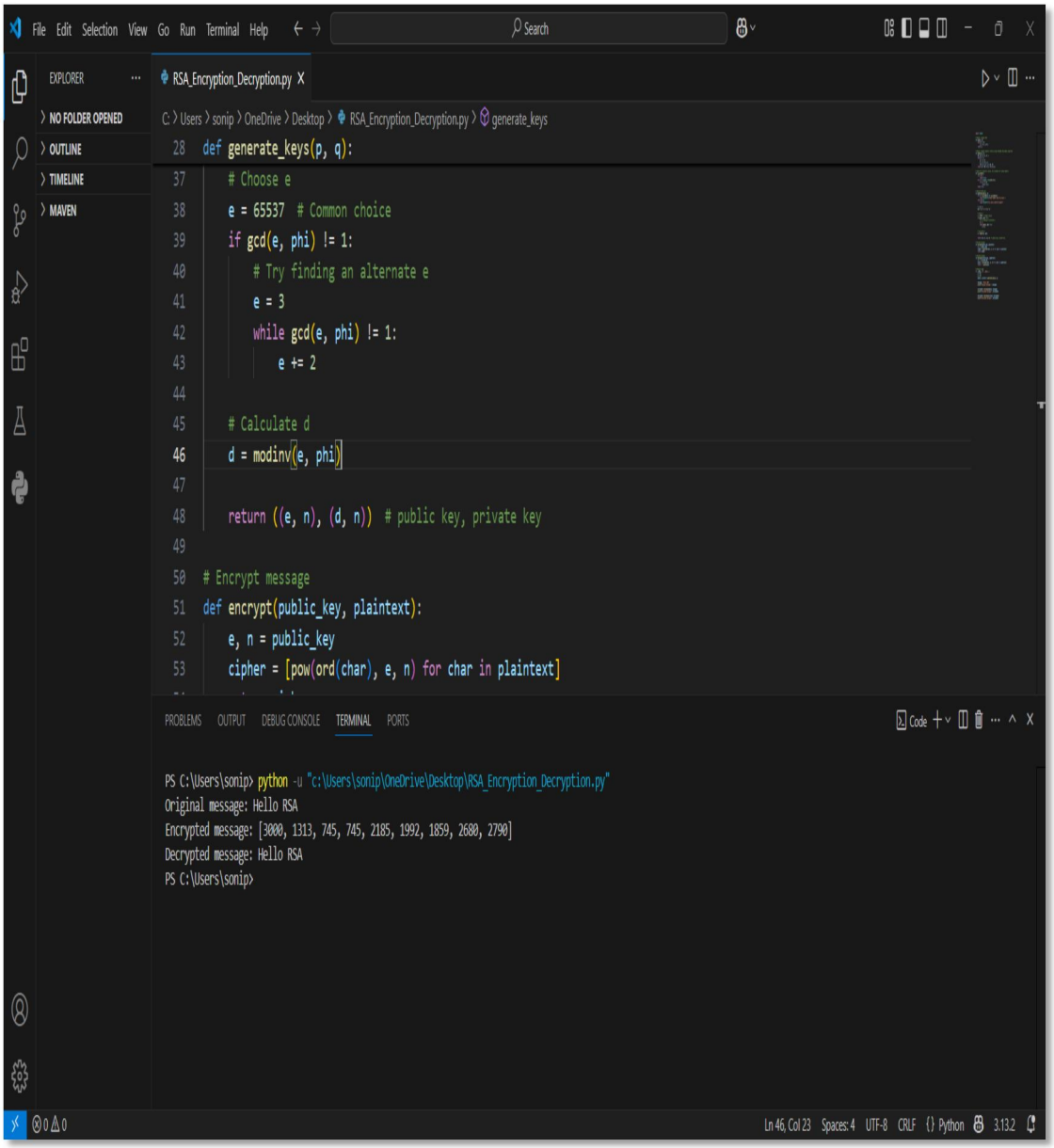

Figure 31

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:**

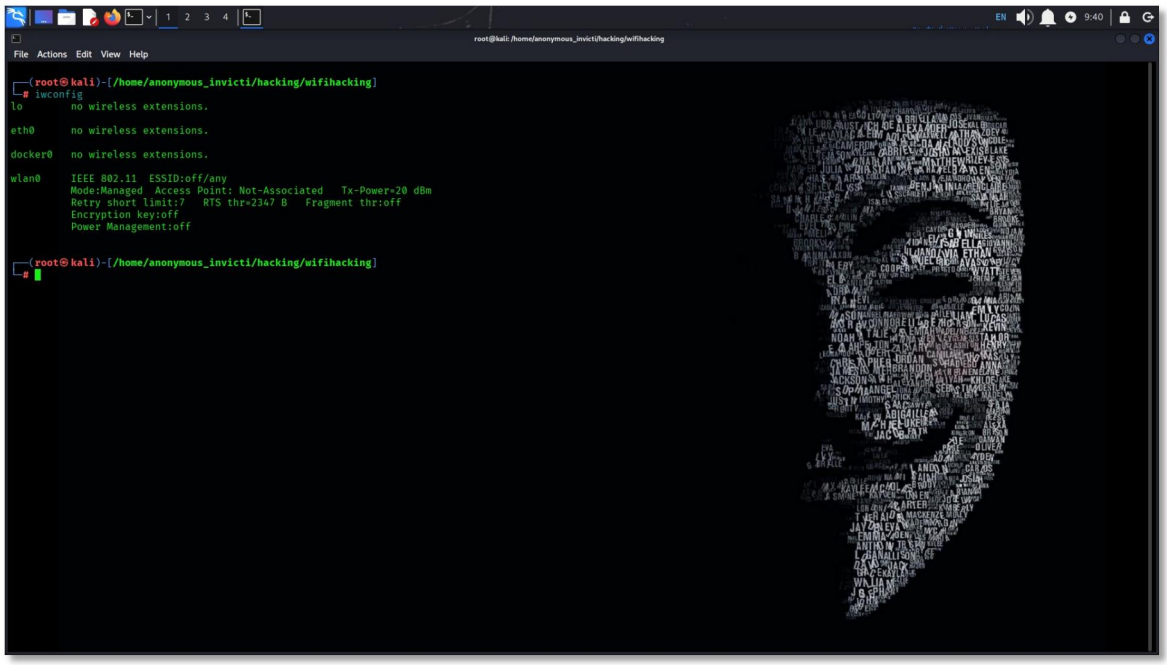

Figure 32

2. **Start Monitor Mode:**

sudo airmon-ng

sudo airmon-ng check kill

sudo airmon-ng start wlan0

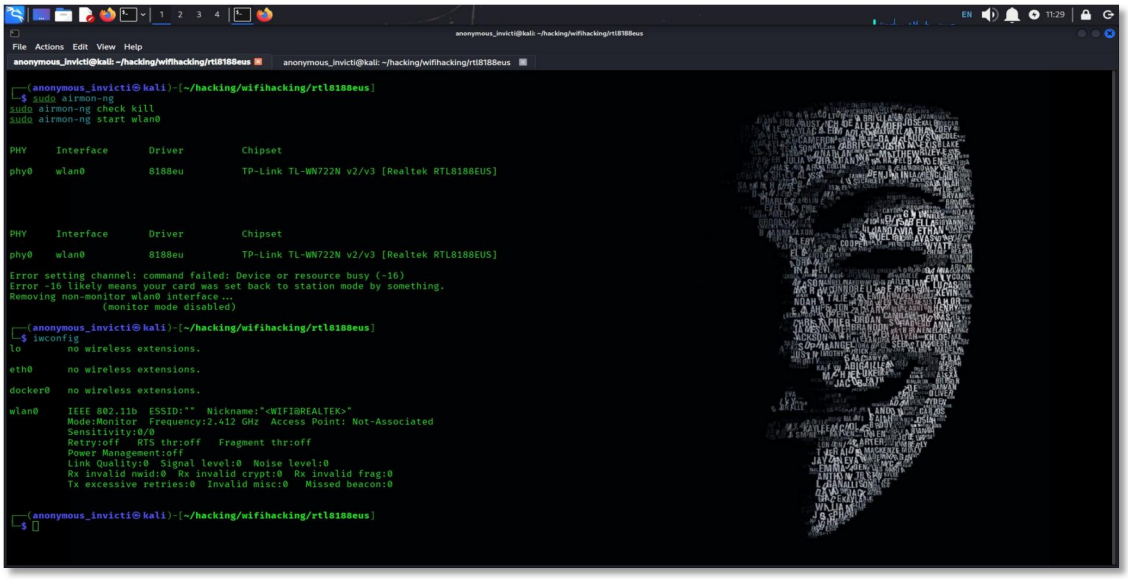

Figure 33

3. Scan Nearby Wi-Fi Networks:

sudo airodump-ng wlan0

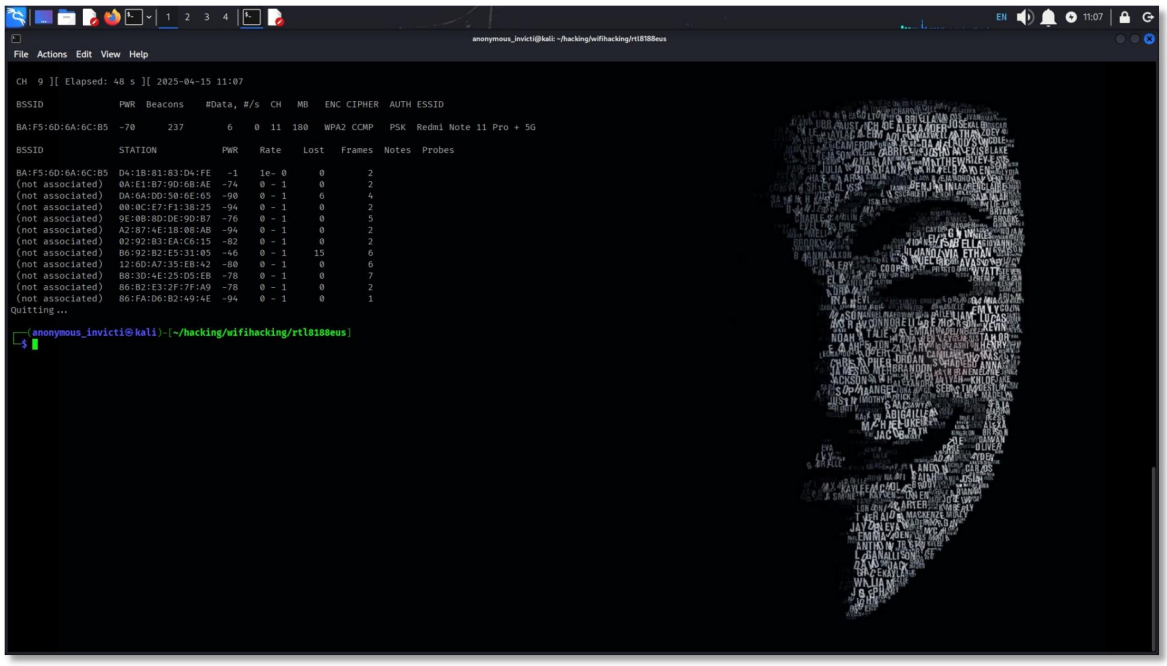

Figure 34

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

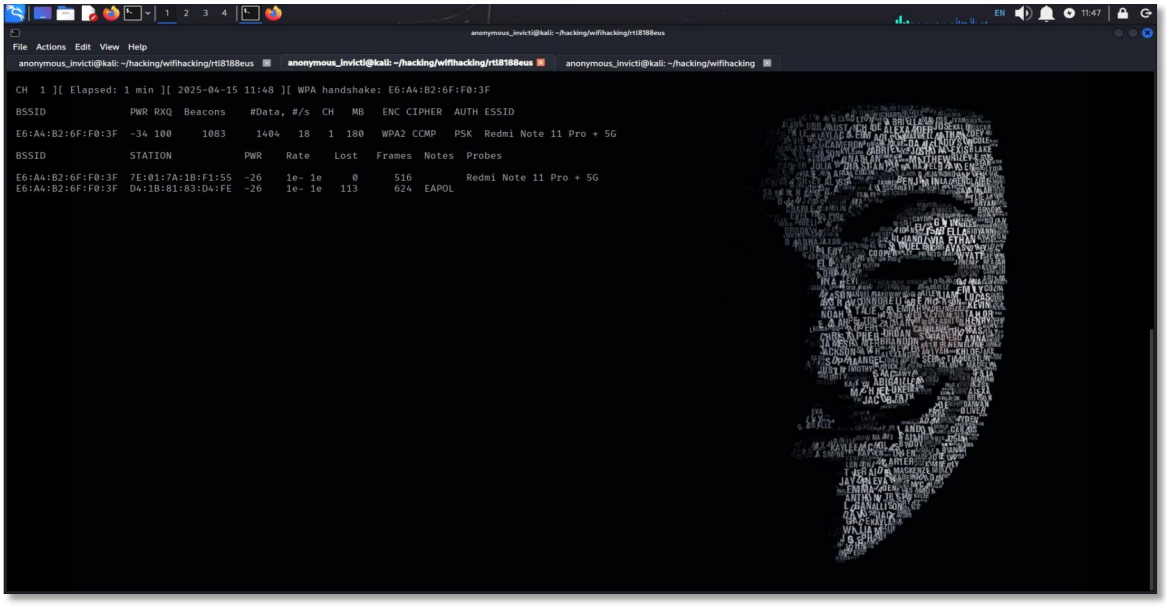

Figure 35

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

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

7. **Captured The WPA Handshake on top:**

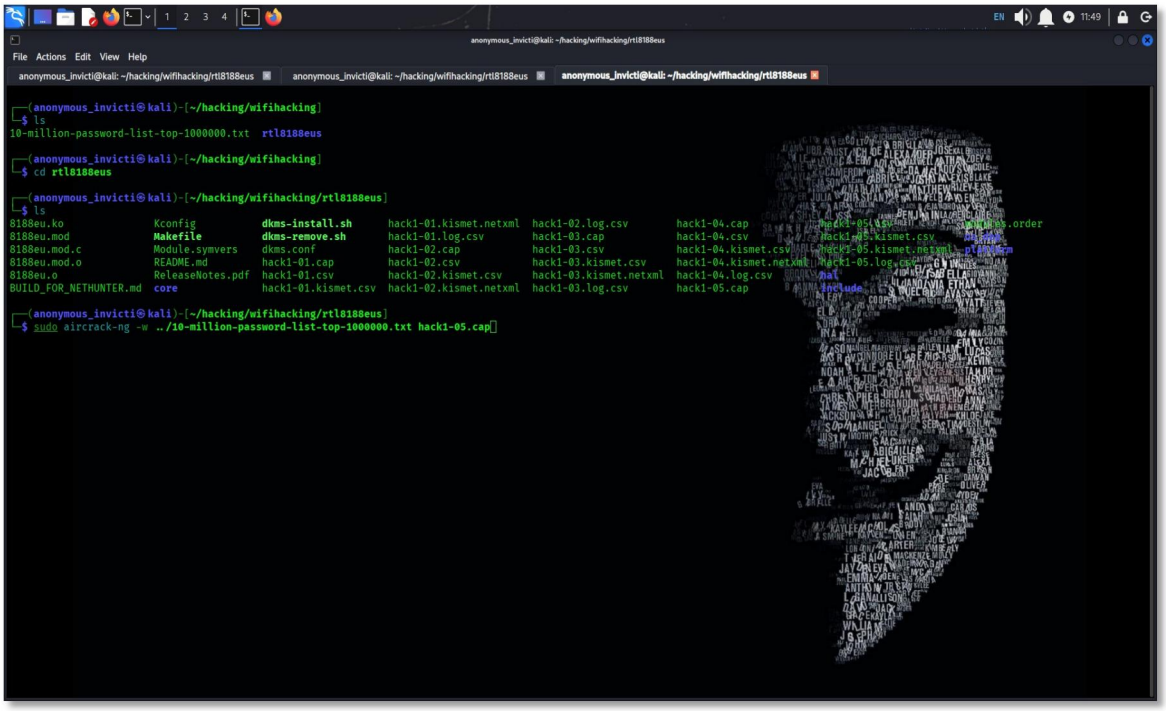

Figure 36

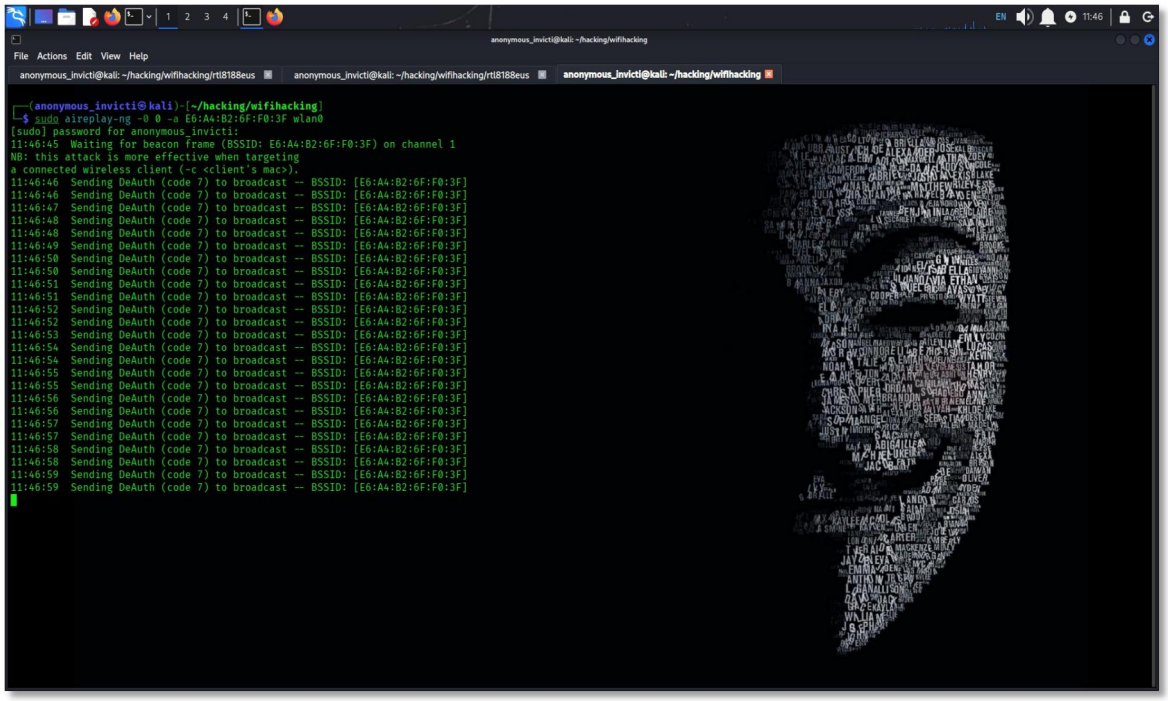

Figure 37

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.