SONA COLLEGE OF ARTS AND SCIENCE

Affiliated to Periyar University, Salem -11

DEPARTMENT OF COMPUTER SCIENCE WITH CYBER SECURITY

Practical Record

Name :

Register Number :

Subject Code : **23USEP04**

Subject Title : **PRACTICAL: CYBER SECURITY LAB**

Year / Sem : **II / IV**

ACADEMIC YEAR: 2024 - 2025

SONA COLLEGE OF ARTS AND SCIENCE
Affiliated to Periyar University, Salem

Certificate

This is to Certify that the Practical Record **“Practical: Cyber Security Lab”** is a bonafide work done by ______________________________

Reg. No_______________________submitted to the Department of Computer Science with Cyber Security, during the academic year 2024 - 2025.

SUBJECT IN-CHARGE
(Mr. K. Gopinath)

HEAD OF THE DEPARTMENT
(Dr. S. Mohanapriya)

Submitted for University Practical Examination held on ______________

INTERNAL EXAMINER

EXTERNAL EXAMINER

INDEX

Ex.No: 01.a)	Implement the following Substitution & Transposition Techniques concepts: **Caesar Cipher**
Date:	

AIM

To write the program in python to implement the Substitution and Transposition Techniques concepts Caesar Cipher.

ALGORITHM

Step 1. Input Validation

a) Validate user input for text and shift values.

b) Ensure shift is an integer.

Step 2. Encryption/Decryption Mode Selection

a) Determine whether to encrypt (E) or decrypt (D) based on user input.

Step 3. Caesar Cipher Formula Application

a) Apply the Caesar cipher formula to each character in the text:

b) encrypted_char = (ord(char) - shift_base + shift) % 26 + shift_base

c) Handle non-alphabetic characters by leaving them unchanged.

Step 4. Shift Value Adjustment

a) Adjust the shift value based on the encryption/decryption mode:

- shift = -shift for decryption

Step 5. Output Generation: Generate the encrypted or decrypted text based on the transformed characters.

Step 6. Result Output - Print the final encrypted or decrypted text.

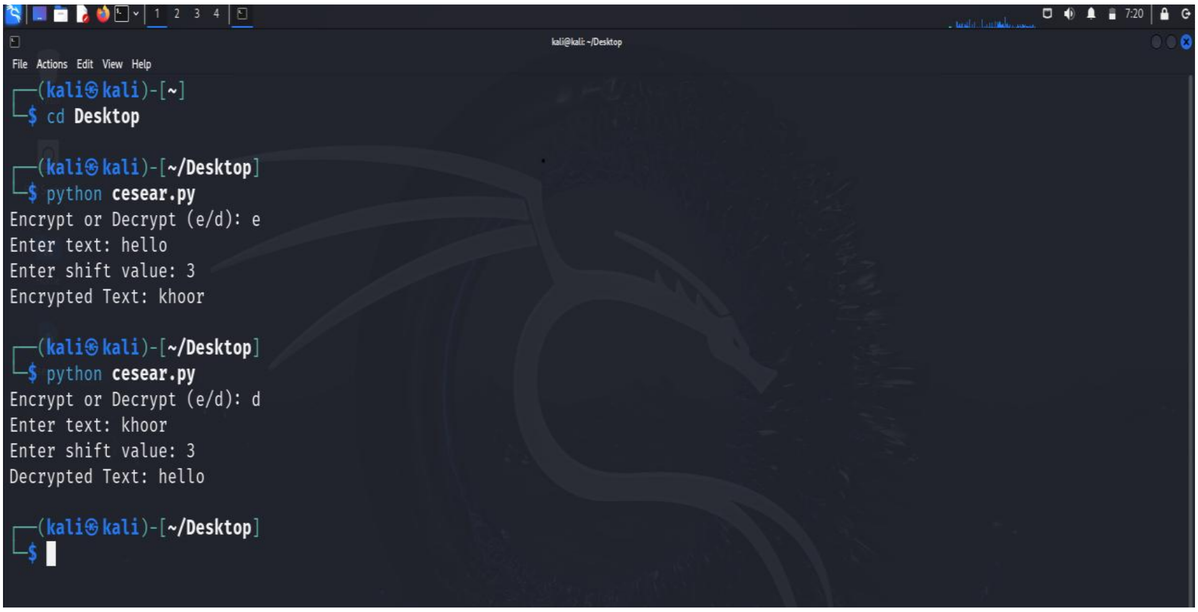
CODING:

```
def caesar_cipher(text, shift, decrypt=False):
        result = ""
If decrypt:
        shift = -shift  # Reverse shift for decryption
 for char in text:
        if char.isalpha():
                shift_base = 65 if char.isupper() else 97
                result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
         else:
                result += char
        return result

choice = input("Encrypt or Decrypt (e/d): ").lower()
text = input("Enter text: ")
shift = int(input("Enter shift value: "))

if choice == 'e':
        print("Encrypted Text:", caesar_cipher(text, shift))
elif choice == 'd':
         print("Decrypted Text:", caesar_cipher(text, shift, decrypt=True))
else:
        print("Invalid Choice!")
```

OUTPUT:



RESULT

The Caesar Cipher program successfully encrypts the plaintext by shifting each letter by the specified key value, producing the corresponding ciphertext.

Ex.No:01.b)	Implement the following Substitution & Transposition Techniques concepts:
Date:	**Rail Fence Cipher**

AIM

To write the program in python to implement the Substitution and Transposition Techniques concepts Railfence Rows & Columns transformation.

ALGORITHM

Step1. Input Validation

a) Validate user input for text and rails values.

b) Ensure rails is an integer.

Step 2. Rail Fence Structure Creation

a) Create a rail fence structure with rails number of rows and text length number of columns.

Step 3. Text Weaving

a) Weave the text into the rail fence structure by moving diagonally up and down the rails.

Step 4. Encryption/Decryption Mode Selection

a) Determine whether to encrypt (E) or decrypt (D) based on user input.

Step 5. Ciphertext/Plaintext Generation

a) Generate the ciphertext (encrypted text) or plaintext (decrypted text) by reading the characters from the rail fence structure.

Step 6. Result Output - Print the final encrypted or decrypted text.

CODING

```
def rail_fence_encrypt(text, rails):
     fence = [['\n'] * len(text) for _ in range(rails)]
    rail = 0
    var = 1

for i in range(len(text)):
    fence[rail][i] = text[i]
    rail += var
     if rail == rails - 1 or rail == 0:
            var = -var
result = []
for rail in fence:
    for char in rail:
        if char != '\n':
            result.append(char)
         return "".join(result)

def rail_fence_decrypt(cipher, rails):
     fence = [['\n'] * len(cipher) for _ in range(rails)]
     rail = 0
    var = 1

 for i in range(len(cipher)):
    fence[rail][i] = '*'
    rail += var
     if rail == rails - 1 or rail == 0:
            var = -var

idx = 0
 for i in range(rails):
 for j in range(len(cipher)):
        if fence[i][j] == '*' and idx < len(cipher):
```

```
                        fence[i][j] = cipher[idx]
    I                   dx += 1

result = []
rail = 0
var = 1
for i in range(len(cipher)):
        result.append(fence[rail][i])
        rail += var
        if rail == rails - 1 or rail == 0:
                        var = -var
                return "".join(result)

choice = input("Encrypt or Decrypt (e/d): ").lower()
text = input("Enter text: ")
rails = int(input("Enter number of rails: "))

if choice == 'e':
        print("Encrypted Text:", rail_fence_encrypt(text, rails))
elif choice == 'd':
        print("Decrypted Text:", rail_fence_decrypt(text, rails))
else:
        print("Invalid Choice!")
```

OUTPUT

RESULT

The Rail Fence Cipher successfully encrypts and decrypts the plaintext by arranging characters in a zigzag pattern across multiple rails and reading them row by row.

Ex.No: 02	Implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript
Date:	

AIM

To implement the Diffie-Hellman key exchange algorithm for securely sharing a secret key between two parties.

ALGORITHM

Step 1: Input the prime number pp, base gg, private key aa for User A, and private key bb for User B.

Step 2: Calculate A=gamod pA = g^a \mod p for User A.

Step 3: Calculate B=gbmod pB = g^b \mod p for User B.

Step 4: Send AA to User B and BB to User A.

Step 5: Calculate the shared secret for User A as sharedSecretA=Bamod psharedSecretA = B^a \mod p.

Step 6: Calculate the shared secret for User B as sharedSecretB=Abmod psharedSecretB = A^b \mod p.

Step 7: Compare if both shared secrets match.

Step 8: If the shared secrets match, display success; otherwise, display failure.

Step 9: Ensure that pp and gg are large prime numbers to maintain security.

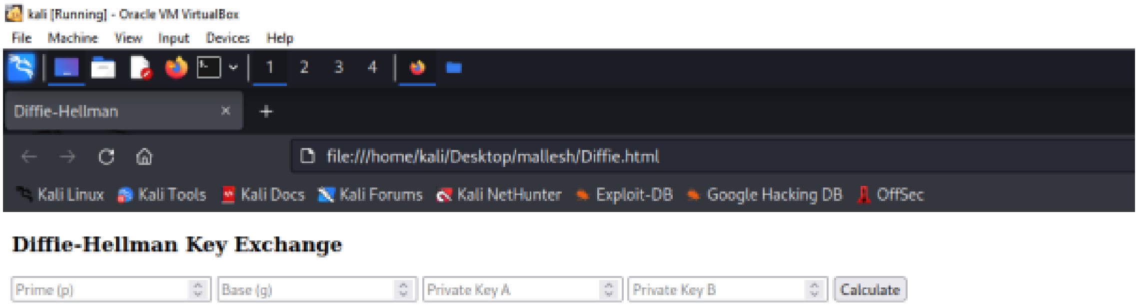
Step 10: Implement error handling to check for invalid inputs and computation errors during the process.

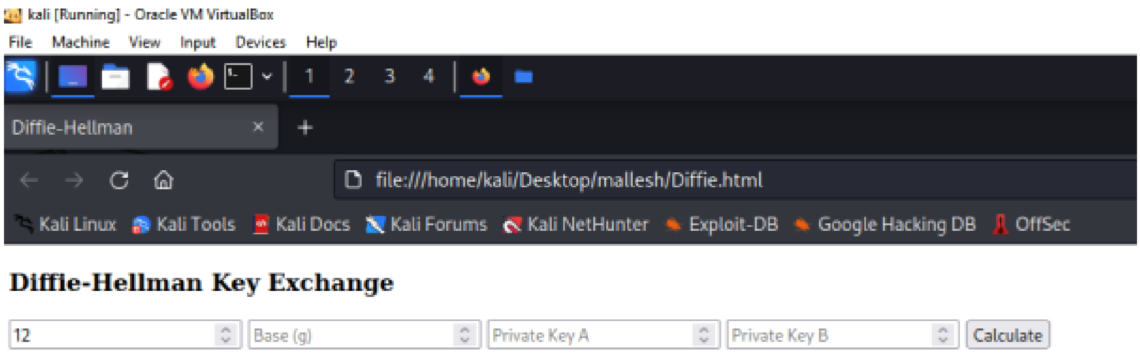
CODING

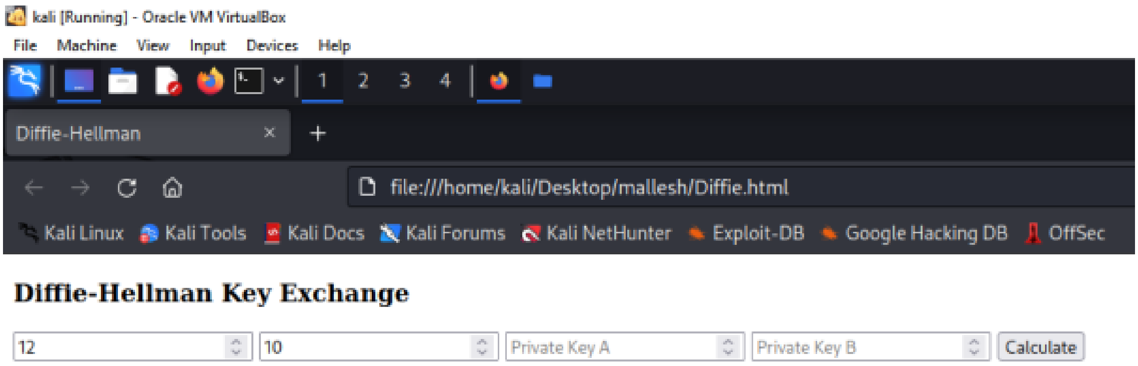
```
<!DOCTYPE html>
<html lang="en">
<head>
 <title>Diffie-Hellman</title>
</head>
<body>
 <h3>Diffie-Hellman Key Exchange</h3>
 <input id="p" placeholder="Prime (p)" type="number">
 <input id="g" placeholder="Base (g)" type="number">
 <input id="a" placeholder="Private Key A" type="number">
 <input id="b" placeholder="Private Key B" type="number">
 <button onclick="dh()">Calculate</button>
 <p id="out"></p>
 <script>
 // Modular Exponentiation function (iterative)
 const modExp = (base, exp, mod) => {
 let result = 1;
 base = base % mod;
 while (exp > 0) {
 if (exp % 2 === 1) result = (result * base) % mod;
 exp = Math.floor(exp / 2);
 base = (base * base) % mod;
 }
 return result;
 };
 // Diffie-Hellman Key Exchange Logic
 function dh() {
 const p = parseInt(document.getElementById('p').value);
 const g = parseInt(document.getElementById('g').value);
 const a = parseInt(document.getElementById('a').value);
 const b = parseInt(document.getElementById('b').value);
 // Calculate A and B using modular exponentiation
 const A = modExp(g, a, p); // A = g^a mod p
```

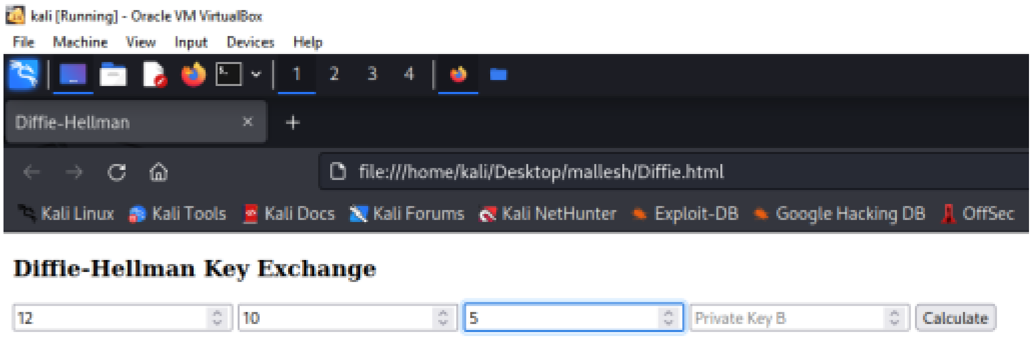
```
const B = modExp(g, b, p); // B = g^b mod p
// Calculate the shared secret
const sharedSecretA = modExp(B, a, p); // sharedSecretA = B^a mod p
const sharedSecretB = modExp(A, b, p); // sharedSecretB = A^b mod p
// Output the shared secret and check if they match
document.getElementById('out').innerText = "Shared Secret: ${sharedSecretA}
(User
A), ${sharedSecretB} (User B)";
if (sharedSecretA === sharedSecretB) {
alert("Key Exchange Successful! Shared Secret: " + sharedSecretA);
} else {
alert("Key Exchange Failed!");
}
}
</script>
</body>
</html>
```

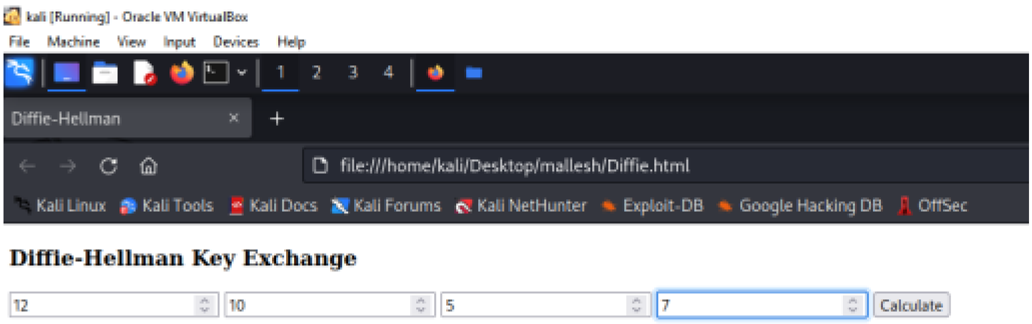
OUTPUT

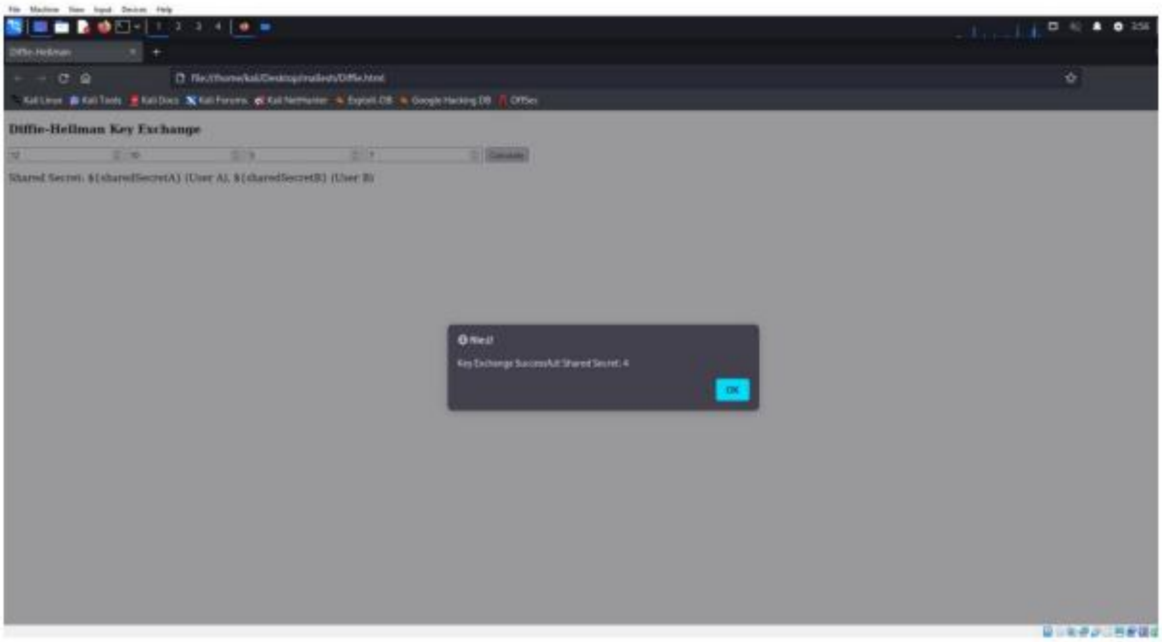

kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
Diffie-Hellman
file:///home/kali/Desktop/mallesh/Diffie.html
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Diffie-Hellman Key Exchange
Prime (p)
Base (g)
Private Key A
Private Key B
Calculate


kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
Diffie-Hellman
file:///home/kali/Desktop/mallesh/Diffie.html
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Diffie-Hellman Key Exchange
12
Base (g)
Private Key A
Private Key B
Calculate


kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
1 2 3 4
Diffie-Hellman
file:///home/kali/Desktop/mallesh/Diffie.html
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Diffie-Hellman Key Exchange
12
10
Private Key A
Private Key B
Calculate


kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Diffie-Hellman
file:///home/kali/Desktop/mallesh/Diffie.html
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Diffie-Hellman Key Exchange
12
10
5
Private Key B
Calculate


kali [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Diffie-Hellman
file:///home/kali/Desktop/mallesh/Diffie.html
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Diffie-Hellman Key Exchange
12
10
5
7
Calculate



RESULT

Thus the above implement the Diffie-Hellman Key Exchange mechanism using HTML and JavaScript successfully verified

Ex. No: 03 a)	**Implement the following Attack: Dictionary attack**
Date:	

AIM

To crack passwords using a dictionary attack method by employing a predefined wordlist (e.g., rockyou.txt) with **John the Ripper**.

ALGORITHM

Ste1: Locate the wordlist: locate rockyou.txt.

Step 2: Run John the Ripper with the wordlist: john wordlist = /usr/share/ wordlists/rockyou.txt.gz.

Step 3: Locate the target password file: locate password.txt.

Step 4: Crack passwords in the target file: john -- wordlist = /usr/share/ wordlists/rockyou.txt.gz password.txt.

COMMAND

Locate rockyou.txt

john –wordlist=/usr/share/wordlists/rockyou.txt.gz

 locate password.txt john
wordlist=/usr/share/wordlists/rockyou.txt.gz password.txt

OUTPUT

```
┌──(kali㉿kali)-[~/Desktop/GURU]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt.gz password.txt

Warning: detected hash type "cryptoSafe", but the string is also recognized as "gost"
Use the "--format=gost" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "HAVAL-256-3"
Use the "--format=HAVAL-256-3" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "Panama"
Use the "--format=Panama" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "po"
Use the "--format=po" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "Raw-Keccak-256"
Use the "--format=Raw-Keccak-256" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "Raw-SHA256"
Use the "--format=Raw-SHA256" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "skein-256"
Use the "--format=skein-256" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "Snefru-256"
Use the "--format=Snefru-256" option to force loading these as that type instead
Warning: detected hash type "cryptoSafe", but the string is also recognized as "Stribog-256"
Use the "--format=Stribog-256" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (cryptoSafe [AES-256-CBC])
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: UTF-16 BOM seen in wordlist. File may not be read properly unless you re-encode it
0g 0:00:00:00 DONE (2025-01-07 14:23) 0g/s 163521p/s 654087c/s 654087C/s Q◆C,◆)◆◆◆y◆E>◆.◆)▒o◆T◆◆b◆◆◆..◆◆◆◆◆V
Session completed.
```

RESULT

Password successfully cracked using a predefined wordlist by using Dictionary attack.

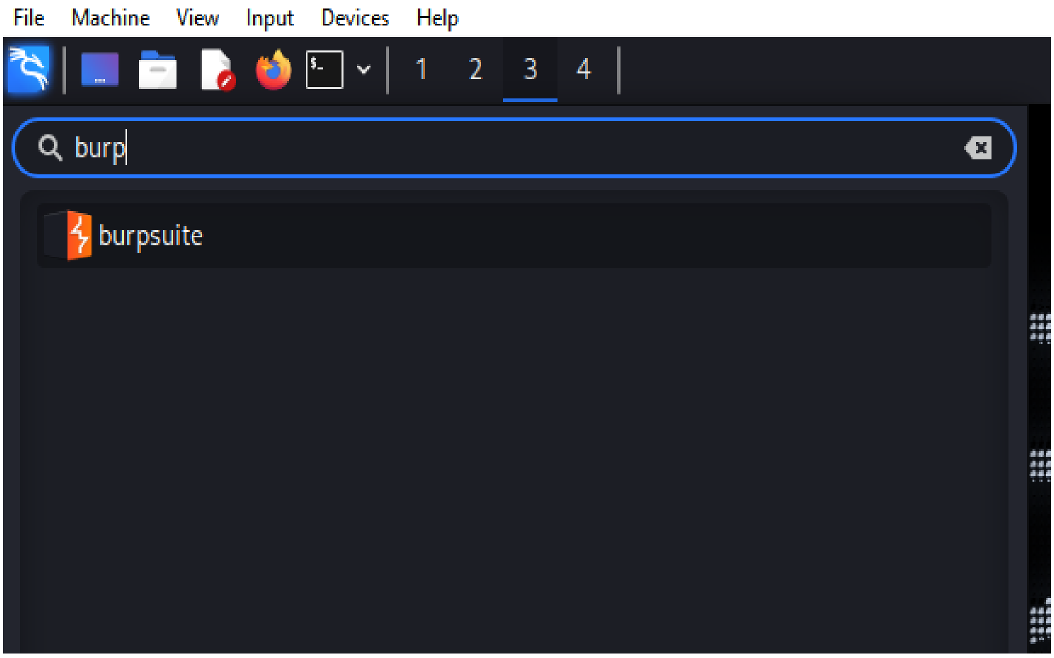
Ex.No:03 b)	Implement the following Attack: Brute Force Attack
Date:	

Aim:

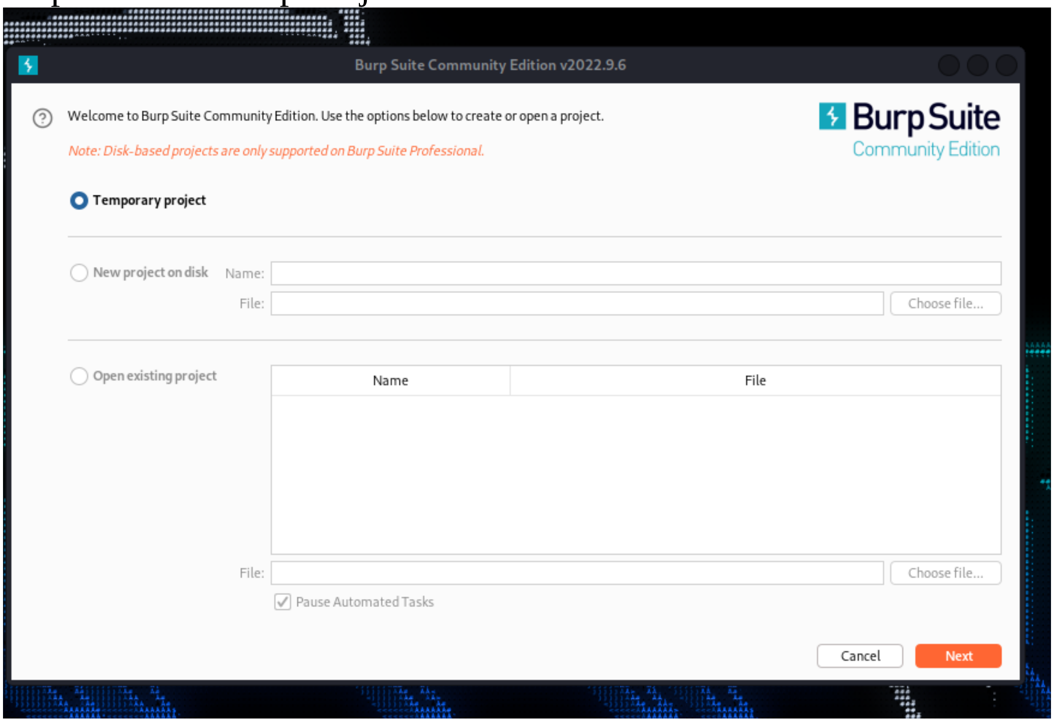
To perform a Brute Force Attack in the website using kali linux.

Procedure:

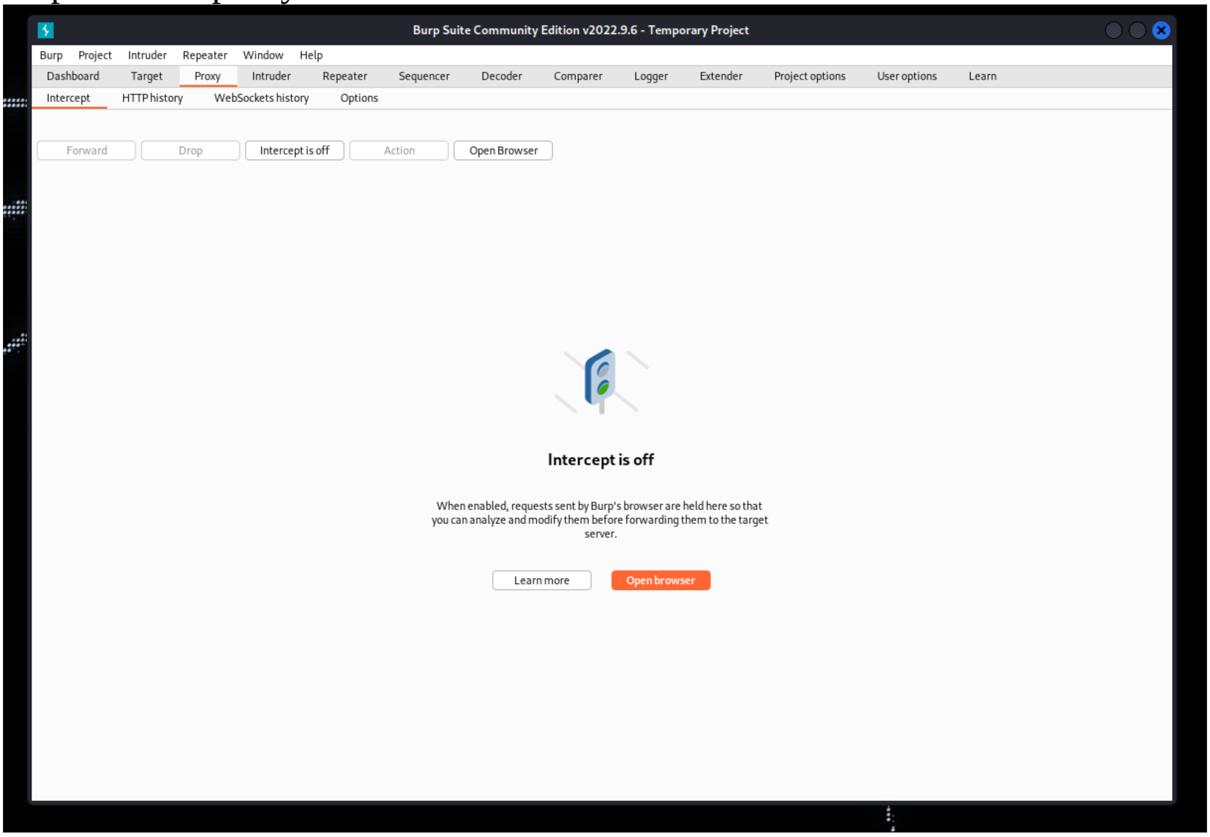
Step-1: Open burpsuit



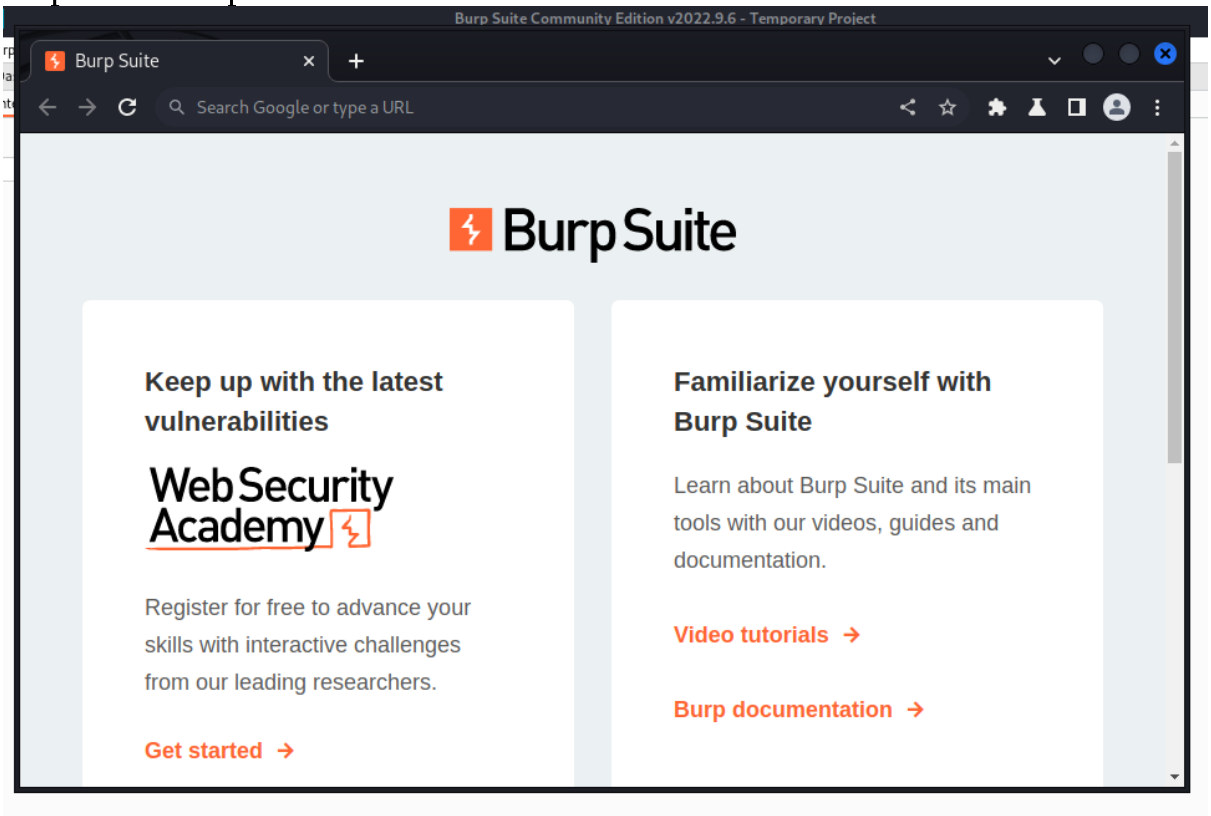
Step-2: Start a temp Project



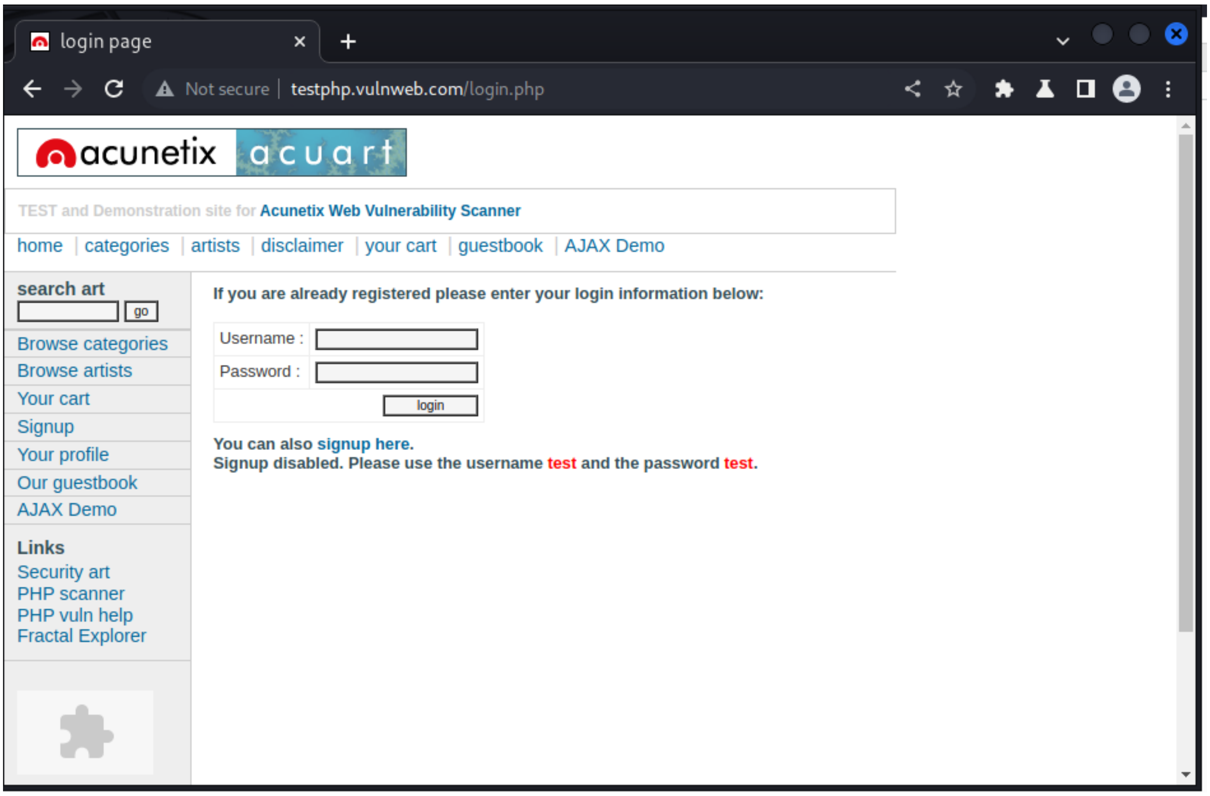
Step-3: Go to proxy



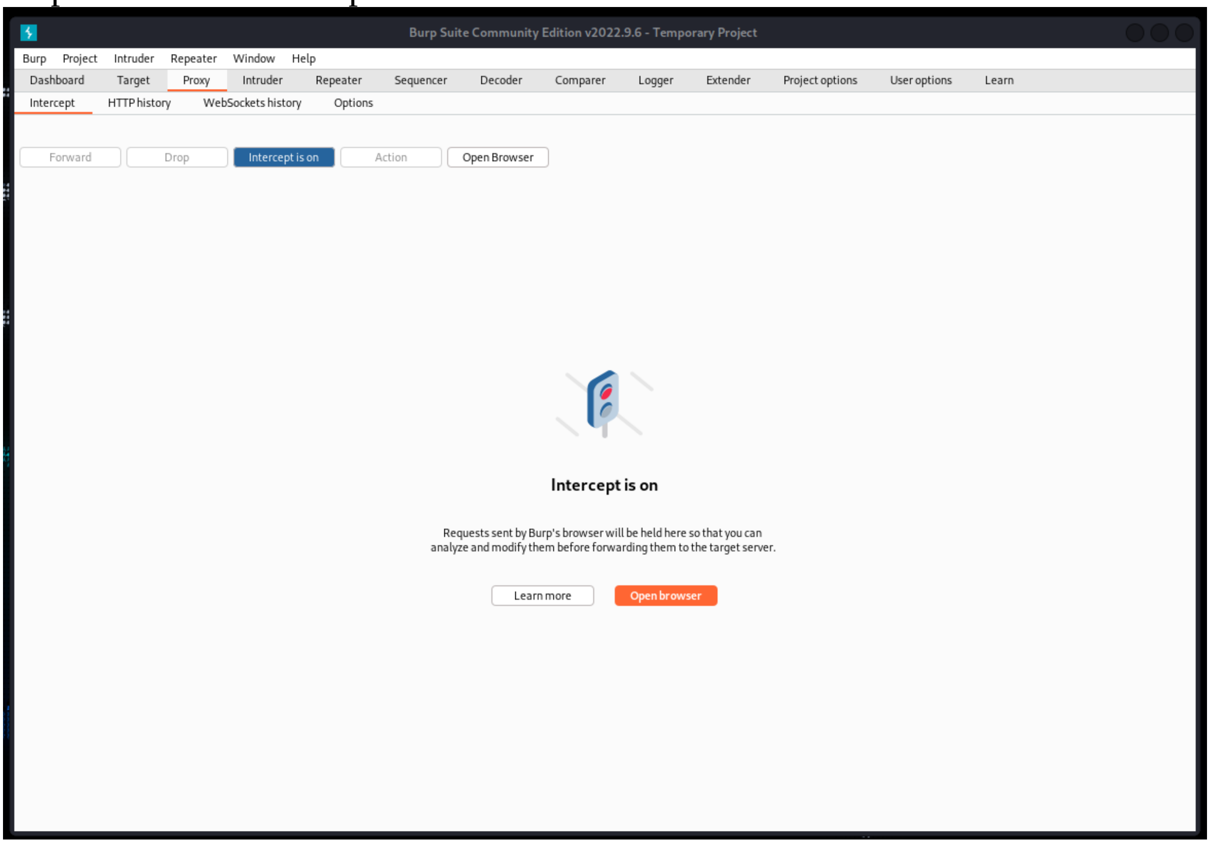
Step-4: Click open browser



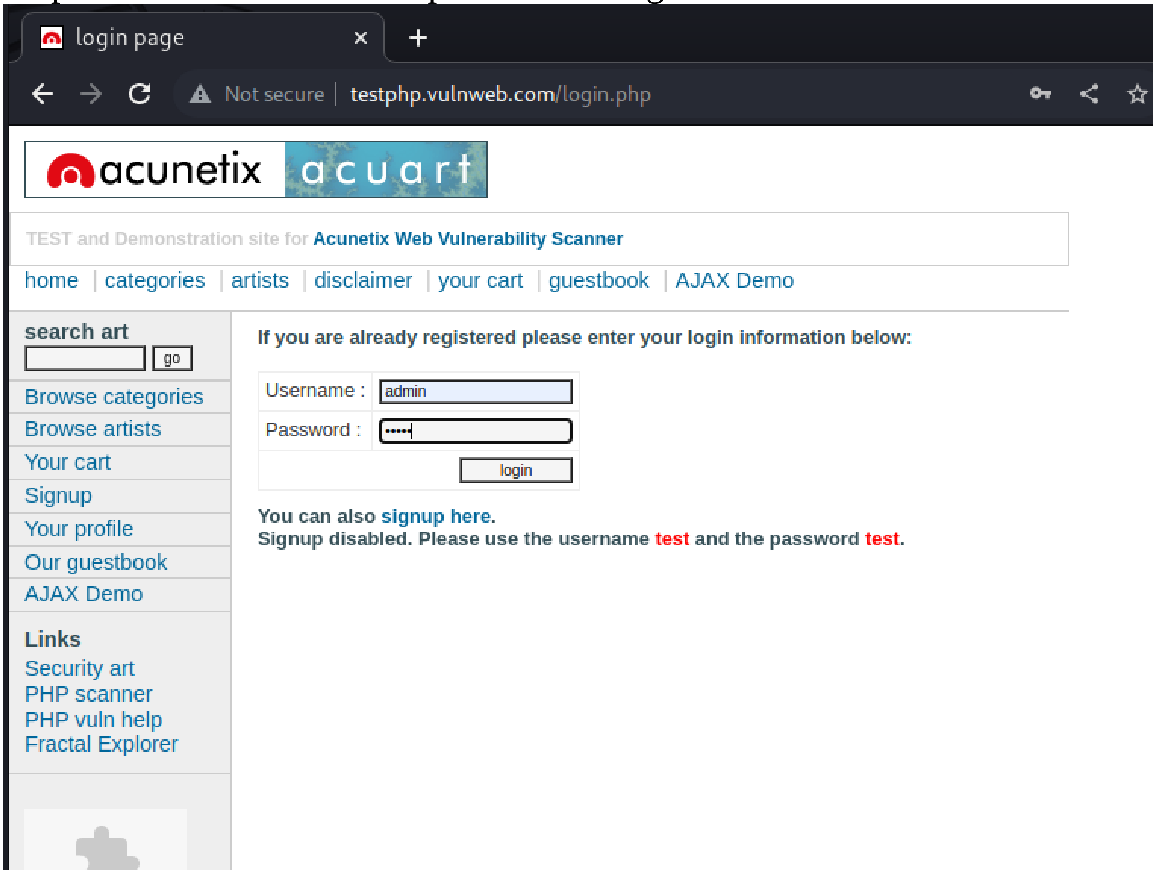
Step-5: Navigate to testphp.vulnweb.com/login.php



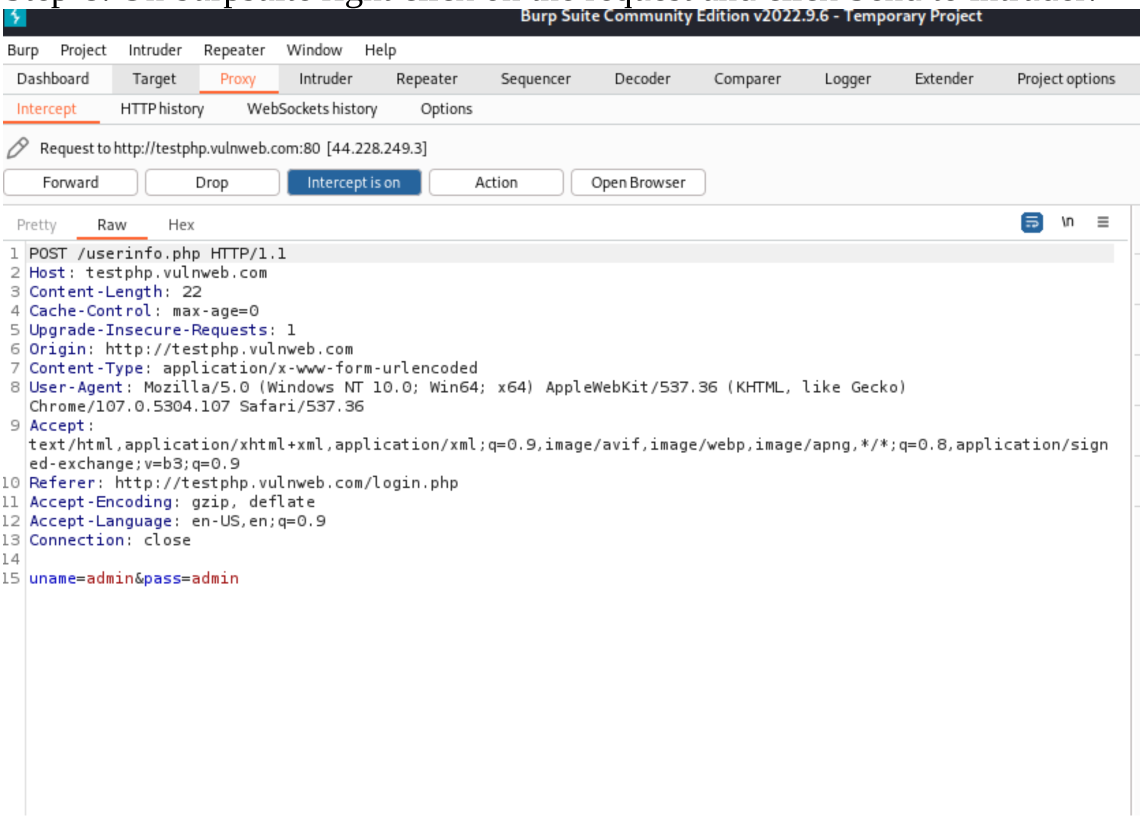
Step-6: Turn on intercept



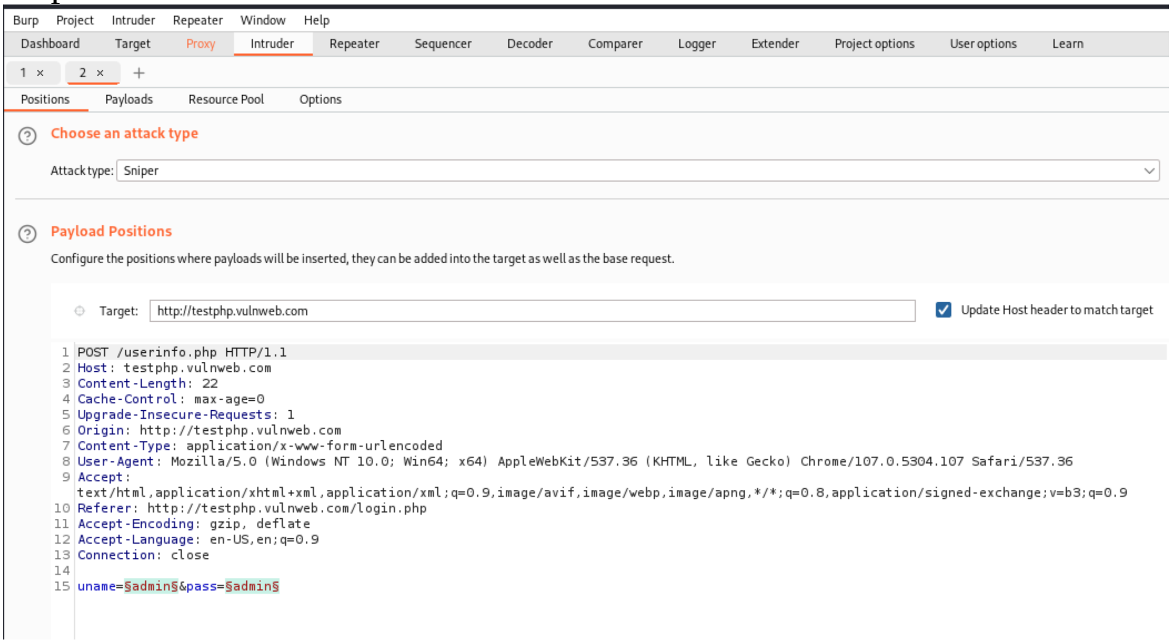
Step-7: Use a username and password to login



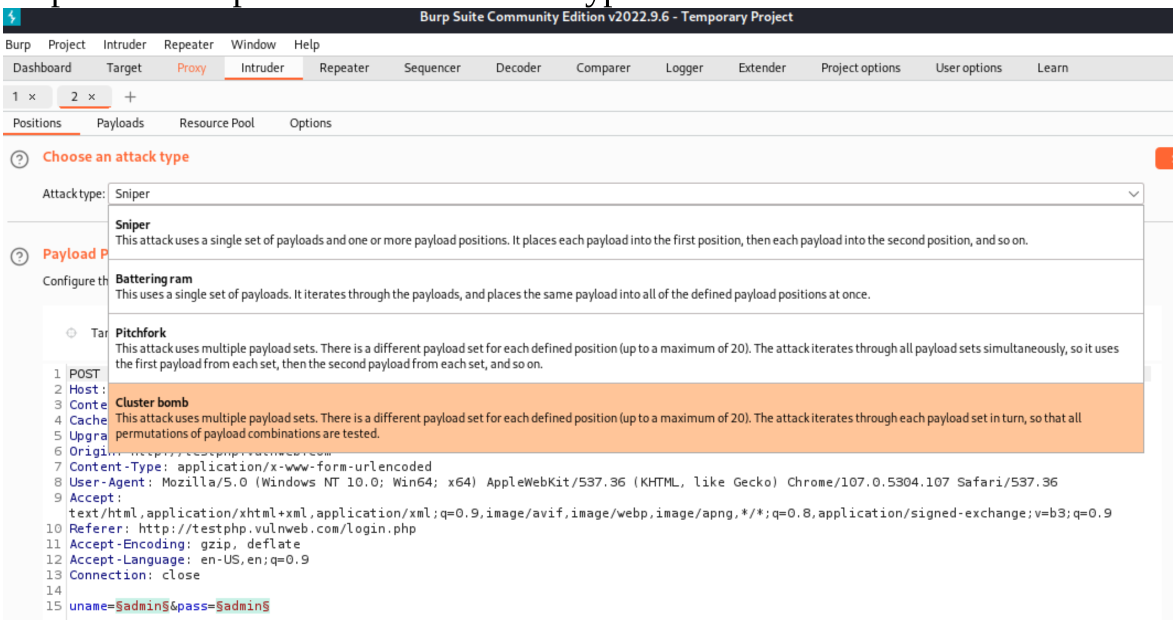
Step-8: On burpsuite right click on the request and click Send to intruder.



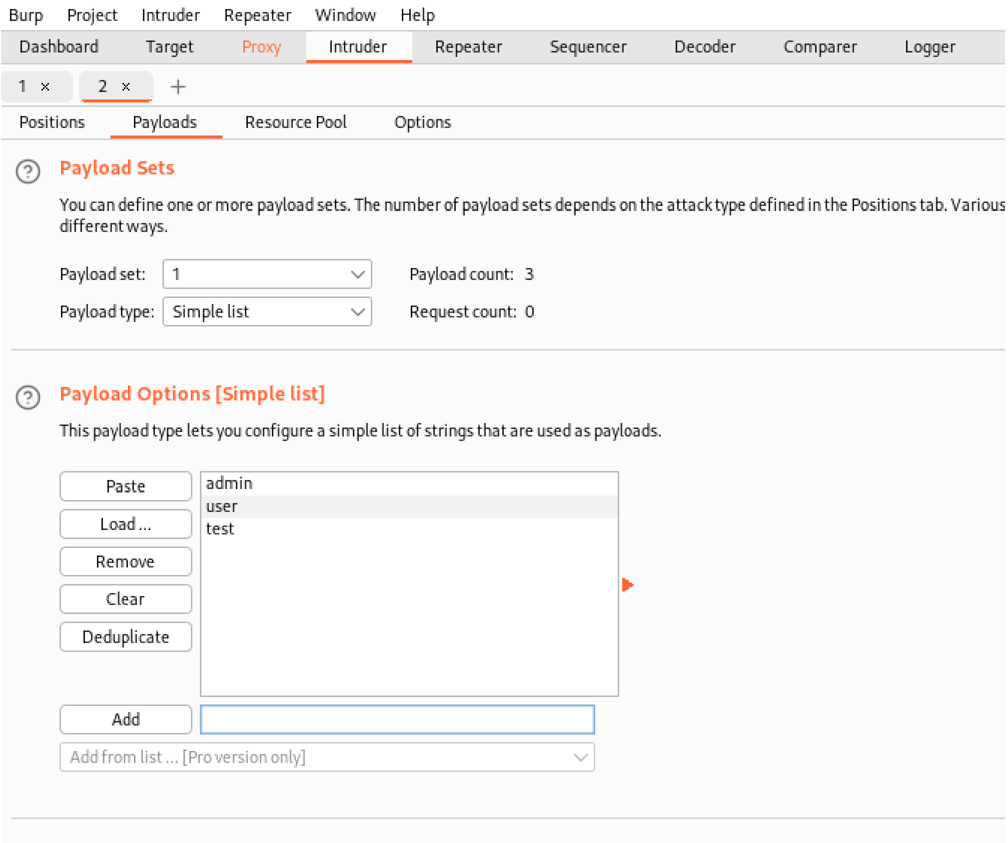
Step-9:Go to Intruder



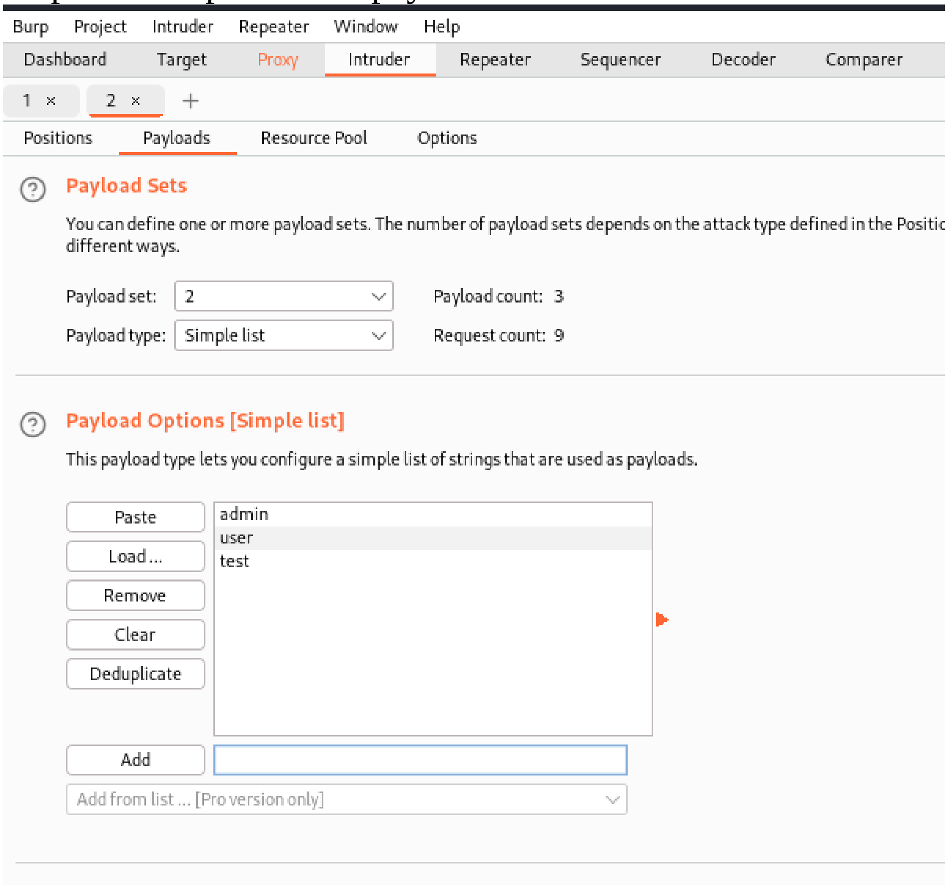
Step-10: Go to position and set attack type to cluster bomb.



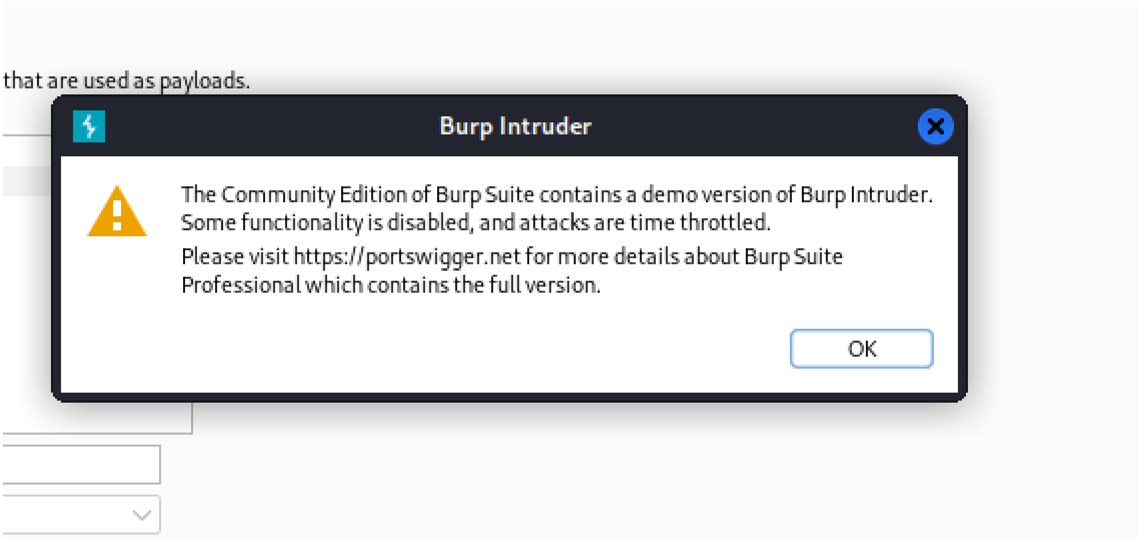
Step-11:Add payloads [admin, user, test]



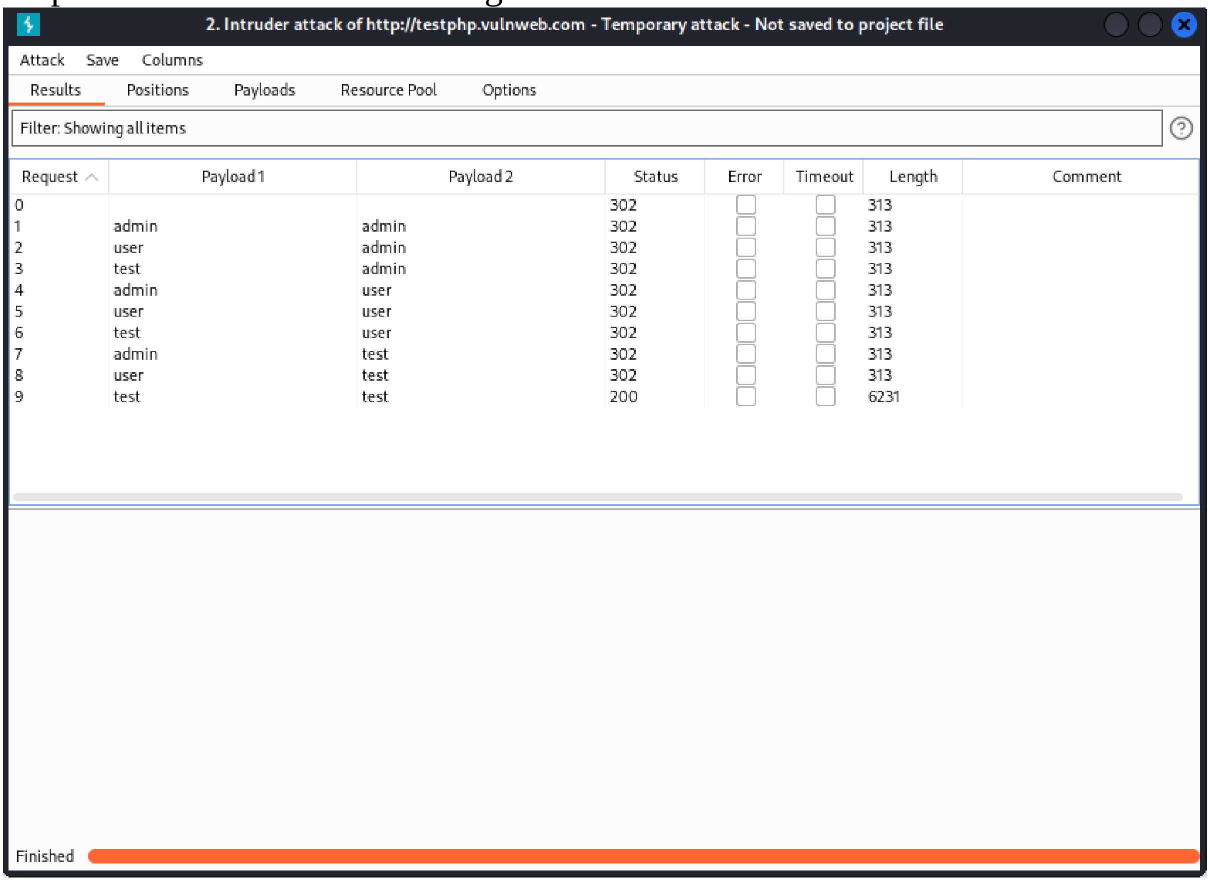
Step-12: Setup for both payloads



Step-13: Start attack and click ok



Step-14: Check of status showing '200' if the cowect value.



Request	Payload 1	Payload 2	Status	Error	Timeout	Length	Comment
0			302	☐	☐	313	
1	admin	admin	302	☐	☐	313	
2	user	admin	302	☐	☐	313	
3	test	admin	302	☐	☐	313	
4	admin	user	302	☐	☐	313	
5	user	user	302	☐	☐	313	
6	test	user	302	☐	☐	313	
7	admin	test	302	☐	☐	313	
8	user	test	302	☐	☐	313	
9	test	test	200	☐	☐	6231	

Result:

Thus the above methods are used to open and execute the Brute Force Attack .

Ex.No:04	Installation of Wire shark, tcpdump, etc and observe data transferred in client server communication using UDP/TCP and identify the UDP/TCP datagram.
Date:	

AIM

To install Wireshark and TCPDump, capture client-server communication using UDP/TCP, and identify the transmitted data packets.

PROCEDURE

Step 1: Login to kali Linux(user:kali,password:kali)

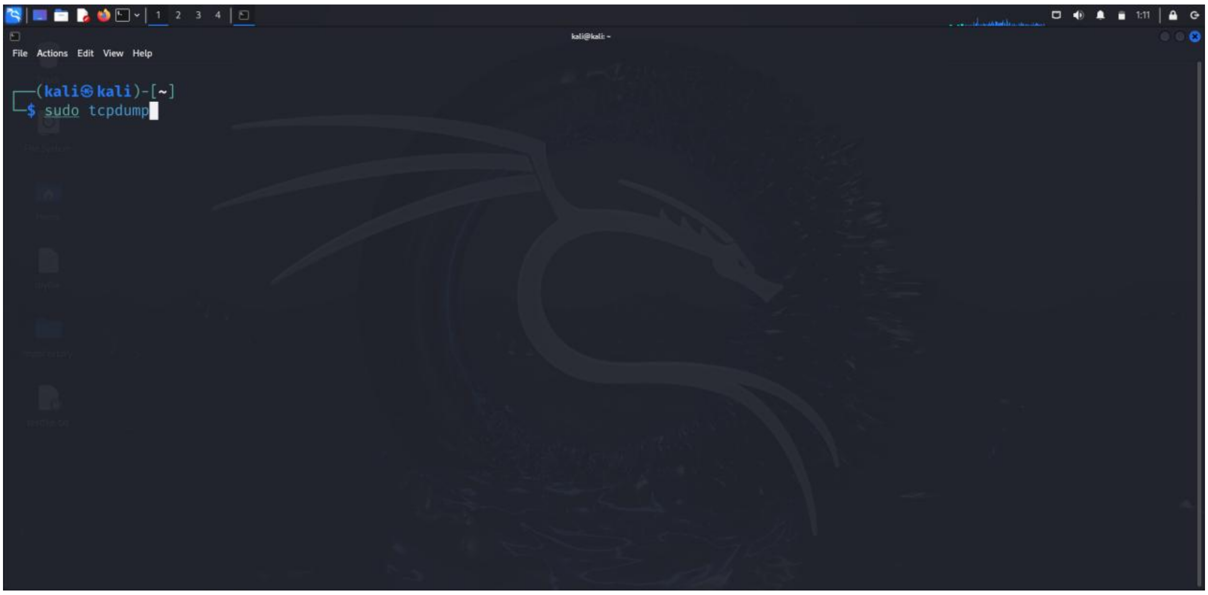
Step 2: Open Terminal

Step 3: Install tcpdump

COMMAND: sudo apt install tcpdump

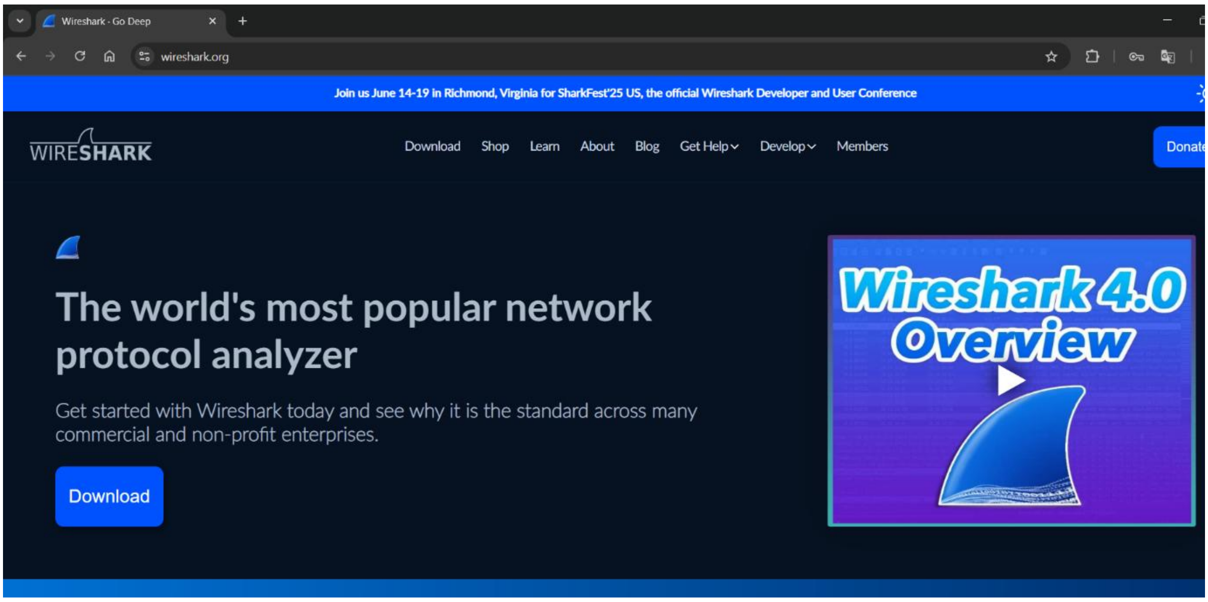
Step 4: Then type command tcpdump below

COMMAND : sudo tcpdump



Step 5: Watch the data traffic in the terminal

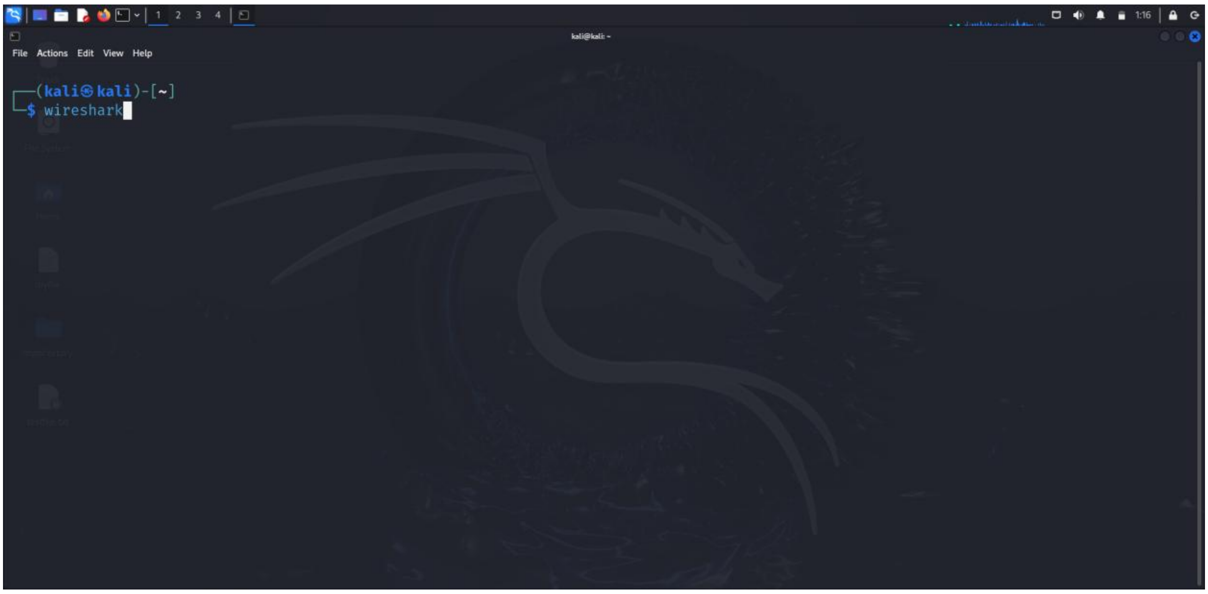
Step 6: Install wireshark visit the official website and follow the instruction



Click the download Button that your desktop support it

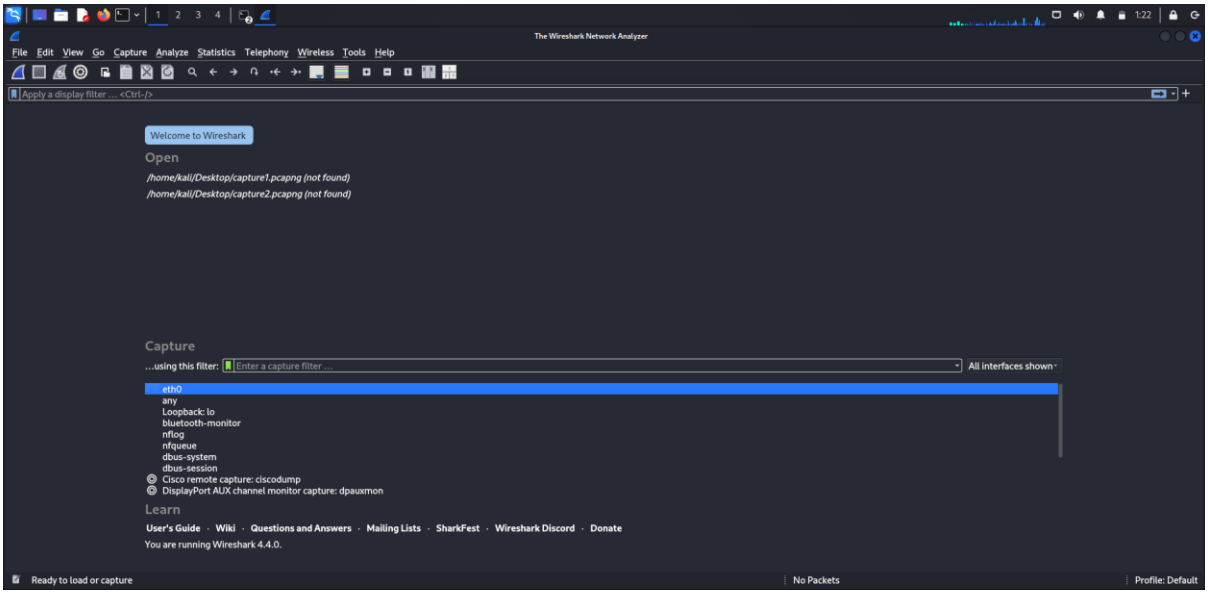
Step 7: To start capturing enter the command wireshark in termial and run it

COMMAND: wireshark



This command open the wireshark in the kali linux

Step 8: click the start button to start capturing



RESULT

Wireshark and TCPDump were successfully installed. The captured network packets were analyzed, and the TCP/UDP datagrams were identified in the client-server communication.

Ex.No:05	Installation of rootkits and study of various options
Date:	

AIM

To install and use the **Chkrootkit** tool in Kali Linux to detect rootkits and unauthorized access attempts.

PROCEDURE

Step1: Begin by installing the Chkrootkit tool, which is available for Debian-based Linux distributions, including Kali Linux.

Step 2: Once the installation is complete, run the Chkrootkit tool to initiate a scan for potential rootkits present in the system.

Step 3: The results of the scan can be lengthy, so using a paginated view makes it easier to read and analyze the output.

Step 4: By default, Chkrootkit requires manual execution. However, to enable automatic daily scans, modify the configuration file and update the setting to allow scheduled scanning.

Step 5: Save the changes made in the configuration file to ensure the system runs automatic scans in the future.

Step 6: Finally, verify the successful execution of Chkrootkit and check whether any rootkits have been detected in the system.

Command

1. sudo apt install chkrootkit -y → Installs Chkrootkit on the system.
2. sudo chkrootkit → Scans the system for rootkits.
3. sudo chkrootkit | less → Displays scan results in a scrollable format.
4. sudo nano /etc/chkrootkit.conf → Opens the configuration file for editing.
5. sudo chkrootkit -q → Runs a scan in quiet mode, showing only infected files.

Output

After executing the Chkrootkit scan, the output will display the results of various security checks.

1. If no rootkits are found, the output may display: Searching for rootkit... No rootkits found.

2. If a potential threat is detected, it will show warnings such as: INFECTED: Possible rootkit detected.

3. Additional options like version checks and help commands will return relevant information about the tool's functionality.

Result

Chkrootkit was successfully installed and executed. The system was scanned for rootkits, ensuring enhanced security against unauthorized access and potential malware threats.

Ex.No:06	Intrusion Detection using SNORT in Kali Linux
Date:	

AIM

To set up and use **Snort**, an open-source **Intrusion Detection System (IDS)**, on **Kali Linux** to monitor and analyze network traffic for potential threats.

PROCEDURE

1. Install **Snort** on Kali Linux.
2. Configure **Snort** by setting up network variables in the configuration file.
3. Run Snort in **different modes**:
   a. **Packet logging mode**
   b. **Intrusion detection mode**
4. Create and add **custom Snort rules**.

Test Snort by generating network activity and checking logs

COMMANDS

Install Snort:

sudo apt update
sudo apt install snort -y

Verify Installation:
snort -V

Configure Snort:
sudo nano /etc/snort/snort.conf

Modify the HOME_NET variable with your network range: ipvar HOME_NET 192.168.1.0/24

Test Snort Configuration:

sudo snort -T -c /etc/snort/snort.conf

Run Snort in Packet Logging Mode:

sudo snort -dev -l /var/log/snort/

Run Snort in Intrusion Detection Mode:

sudo snort -c /etc/snort/snort.conf -i eth0

(Replace eth0 with your actual network interface. Use ip a to check interfaces.)

Add Custom Snort Rules:

sudo nano /etc/snort/rules/local.rules

1. Add the rule: alert icmp any any -> any any (msg:"ICMP Packet Detected"; sid:1000001; rev:1;)

Restart Snort to Apply Rules:
sudo snort -q -c /etc/snort/snort.conf -i eth0

Generate Network Traffic (Ping Another System):

ping -c 3 <Target_IP>

Check Snort Alerts:

cat /var/log/snort/alert

Output:

1. **Snort Version Check:**

```
,,_     -*> Snort! <*-
o"  )~   Version 2.x.x ""
```

2. **Configuration Test Success:**

   Snort successfully validated the configuration! Snort exiting

3. **ICMP Alert in Log File:**

   [**] [1:1000001:1] ICMP Packet Detected [**]

RESULT

Thus the above Intrusion Detection using SNORT in Kali Linux verified successfully.

Ex.No:07	**Secure Data Storage, Secure Data Transmission, and Digital Signatures in Kali Linux**
Date:	

Aim

To demonstrate methods for providing **secure data storage, secure data transmission, and digital signatures** using encryption and cryptographic tools in Kali Linux.

Procedure

1. **Secure Data Storage:**
    a. Encrypt the data using **GnuPG (GPG)** to protect files from unauthorized access.
    b. Use strong encryption algorithms to prevent data leaks.
    c. Decrypt the data whenever required using the correct passphrase or private key.
2. **Secure Data Transmission:**
    a. Use **OpenSSL** or **SSH** to encrypt data before transmitting it over the network.
    b. Establish a secure connection using protocols like **SCP (Secure Copy Protocol)** or **SFTP (Secure File Transfer Protocol)** to prevent interception.
3. **Creating Digital Signatures:**
    a. Generate a **private key** and use it to create a digital signature for a file.
    b. Verify the authenticity of the file using the corresponding **public key**.
    c. This ensures that the file has not been tampered with during transmission.

COMMAND

1. **Secure Data Storage:**
   a. gpg -c file.txt → Encrypts a file using GPG.
   b. gpg file.txt.gpg → Decrypts an encrypted file.
2. **Secure Data Transmission:**
   a. scp file.txt user@remote_host:/destination/path/ → Securely transfers a file over SSH.
   b. openssl enc -aes-256-cbc -salt -in file.txt -out file.enc → Encrypts a file using OpenSSL.
3. **Creating Digital Signatures:**
   a. gpg --output file.sig --sign file.txt → Creates a digital signature.
   b. gpg --verify file.sig file.txt → Verifies the digital signature.

OUTPUT

1. **Encrypted File Output:** File 'file.txt.gpg' created (encrypted).

2. **Secure Transmission Output:** file.txt 100% transferred securely.

3. **Digital Signature Verification Output:** gpg: Signature made using RSA key
   gpg: Good signature from "User Name"

Result

Successfully demonstrated **secure data storage** using encryption, **secure data transmission** using encrypted transfer protocols, and **digital signature creation and verification** to ensure file authenticity in Kali Linux.