R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

Institute Vision and Mission

Vision

To be knowledge hub of providing quality technical education and promoting research for building up of our nation and its contribution for the betterment of humanity

Mission

- To make the best use of state-of-the-art infrastructure to ensure quality technical education
- To develop industrial collaborations to promote innovation and research capabilities
- To inculcate values and ethics to serve humanity

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

To be a source of knowledge in the field of Computer Science and Engineering to cater to the growing need of industry and society

Mission

- To avail state-of-the art infrastructure for adopting cutting edge technologies and encouraging research activities
- To promote industrial collaborations for professional competency
- To nurture social responsibility and ethics to become worthy citizens

R.M.K. COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

R.S.M. Nagar, PUDUVOYAL-601 206

Approved by AICTE, New Delhi /Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi (All Eligible Courses)/ NAAC with 'A' Grade
An ISO 21001:2018 Certified Institution

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Educational Objectives (PEOs)

Graduates of Computer Science and Engineering Program will

1. Become a globally competent professional in all spheres and pursue higher education world over.
2. Successfully carry forward domain knowledge in computing and allied areas to solve complex real world engineering problems.
3. Continuously upgrade their technical knowledge and expertise to keep pace with the technological revolution.
4. Serve the humanity with social responsibility combined with ethics.

Program Specific Outcomes (PSOs)

Graduates of Computer Science and Engineering Program will be able to:

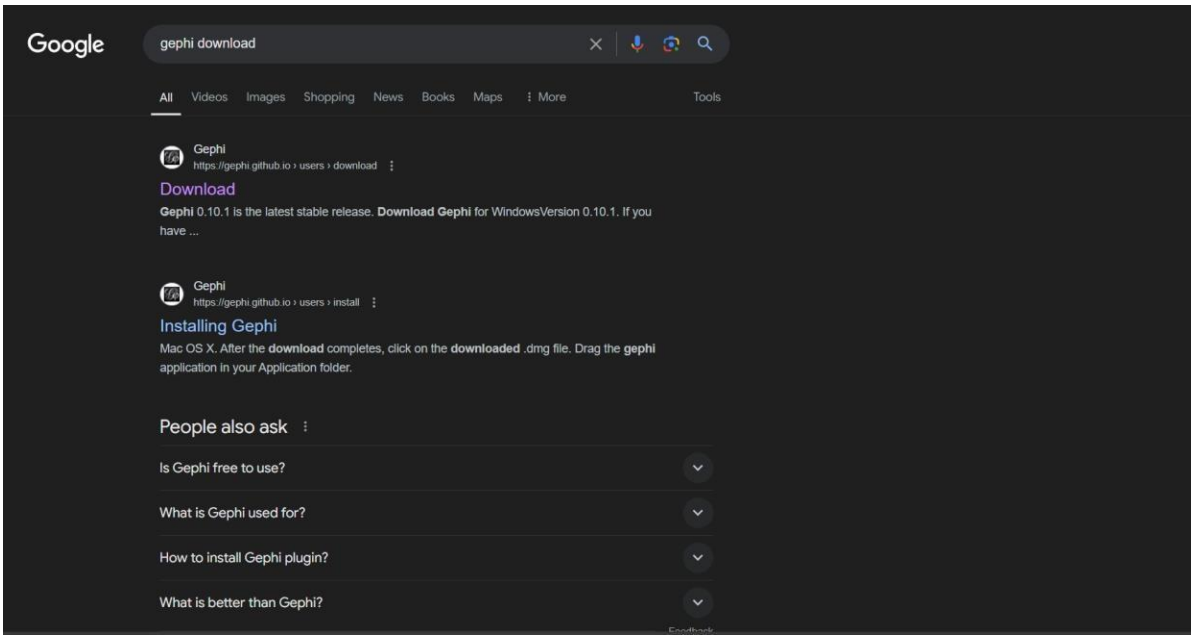
- Analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.
- Apply software engineering principles and practices for developing quality software for scientific and business applications.
- Implement cost-effective solutions for the betterment of both industry and society with the technological skills acquired through the Centres of Excellence.

EX:1 Explore Social network analysis tools to learn about the users and networks

Aim: To Explore a social network analysis tool to learn about the users and networks

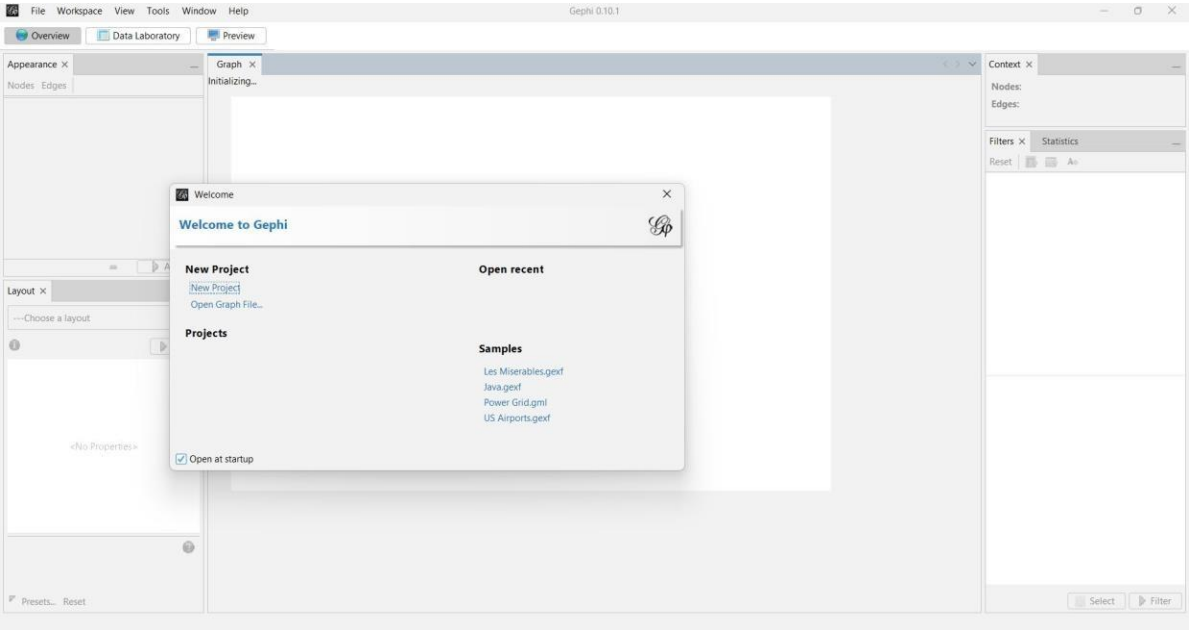
Procedure:

Step 1: Download the Gephi tool using any browser



Step 2: Install the Gephi tool Step 3:

Open the tool

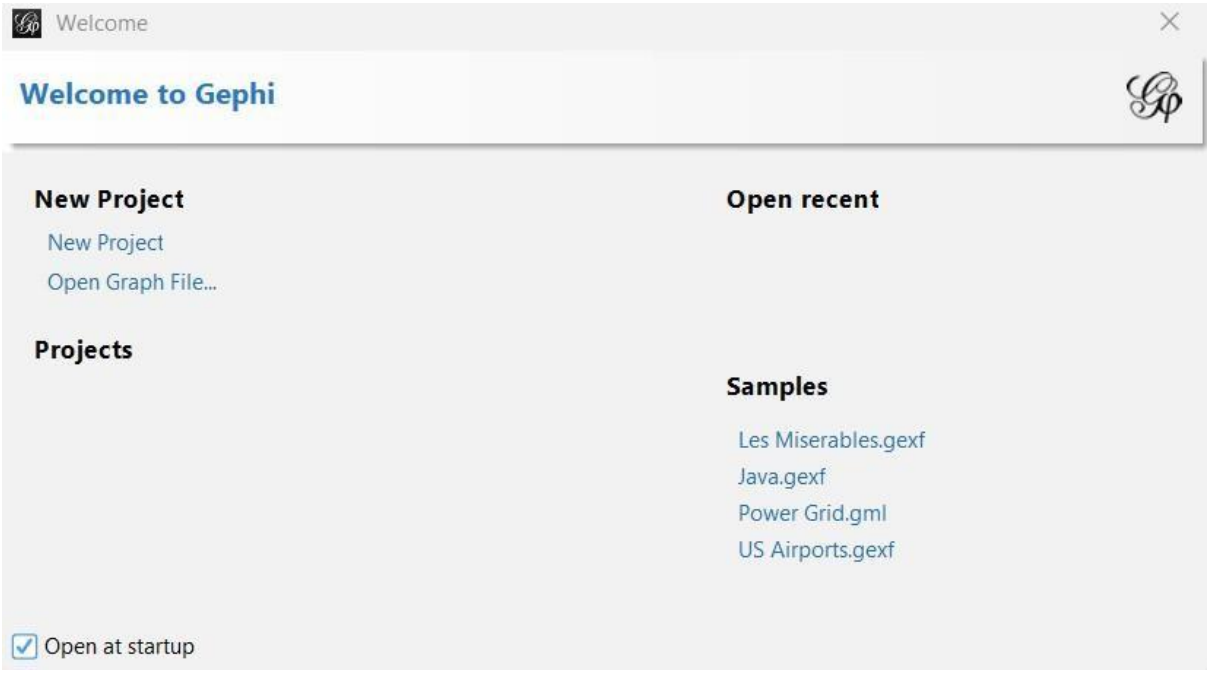


Step 4: You can download the dataset by creating **a New Project**: Go to File > New Project.

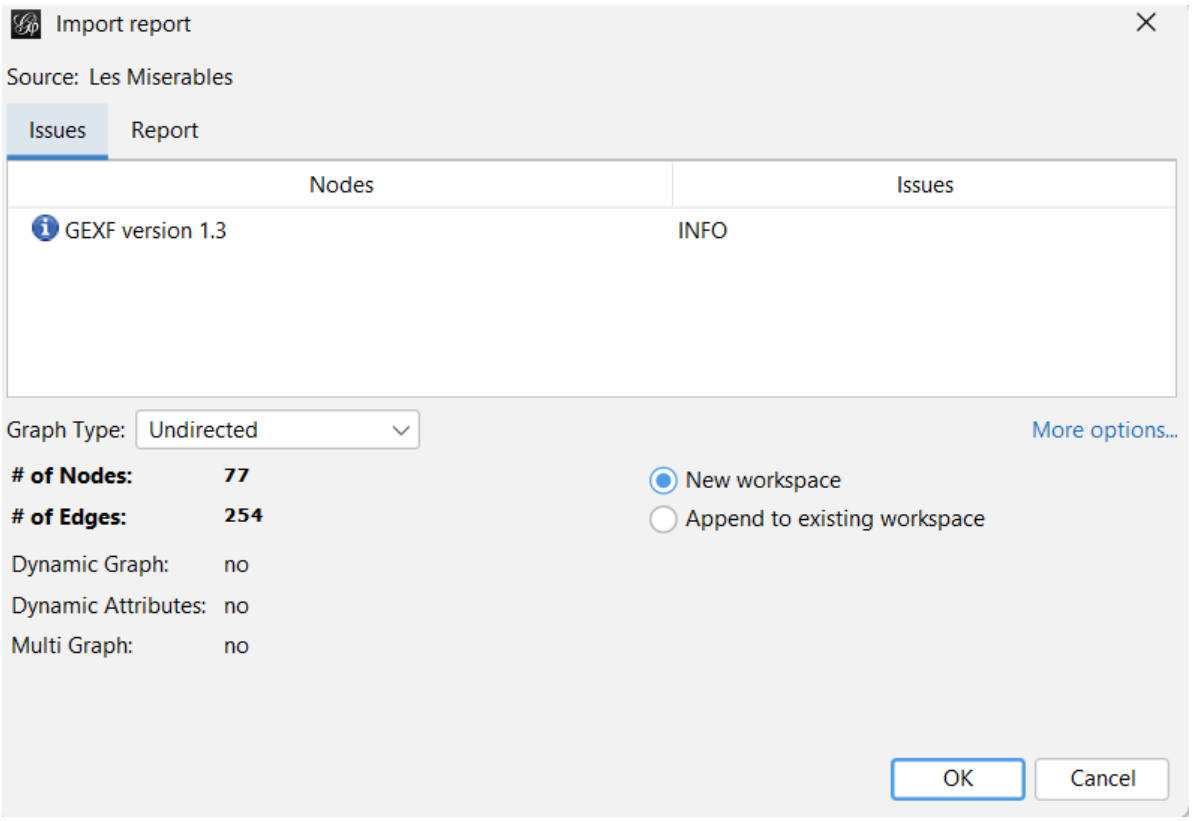
Import Dataset:

- Download a sample social network dataset, such as the Les Miserables co-occurrence network.
- In Gephi, go to File > Open and select the downloaded Les Miserables.gml file.
- In the import settings, ensure "Graph Type" is set to "Undirected" and click "OK".

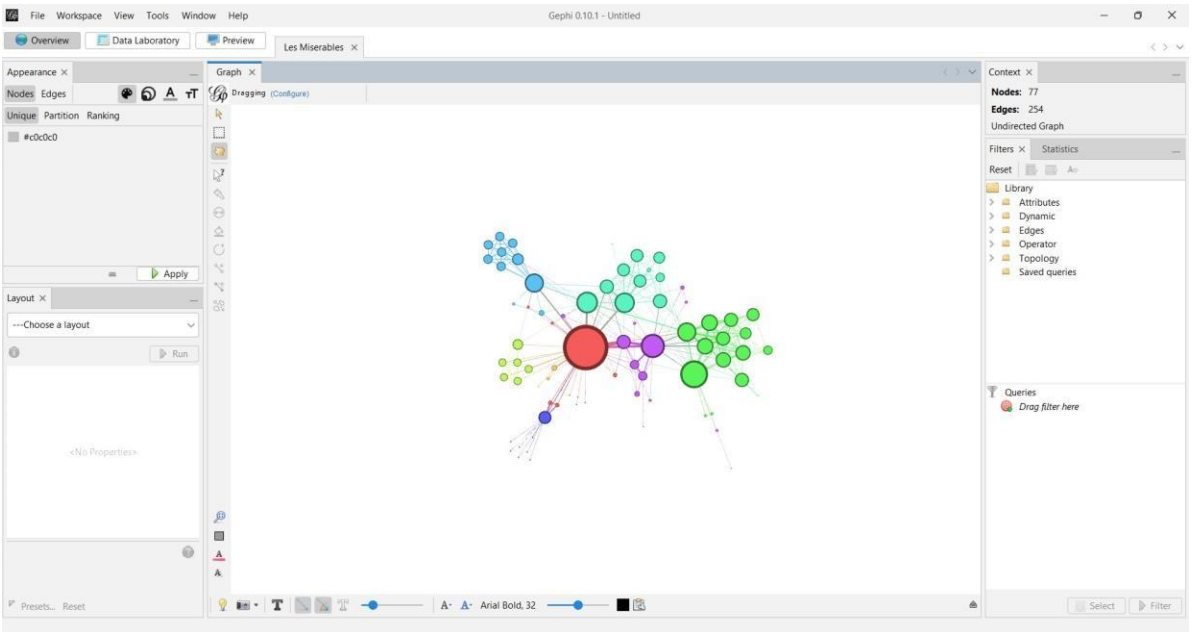
Step 5: Otherwise you can select the default dataset by the Gephi tool. As you can see in the samples.

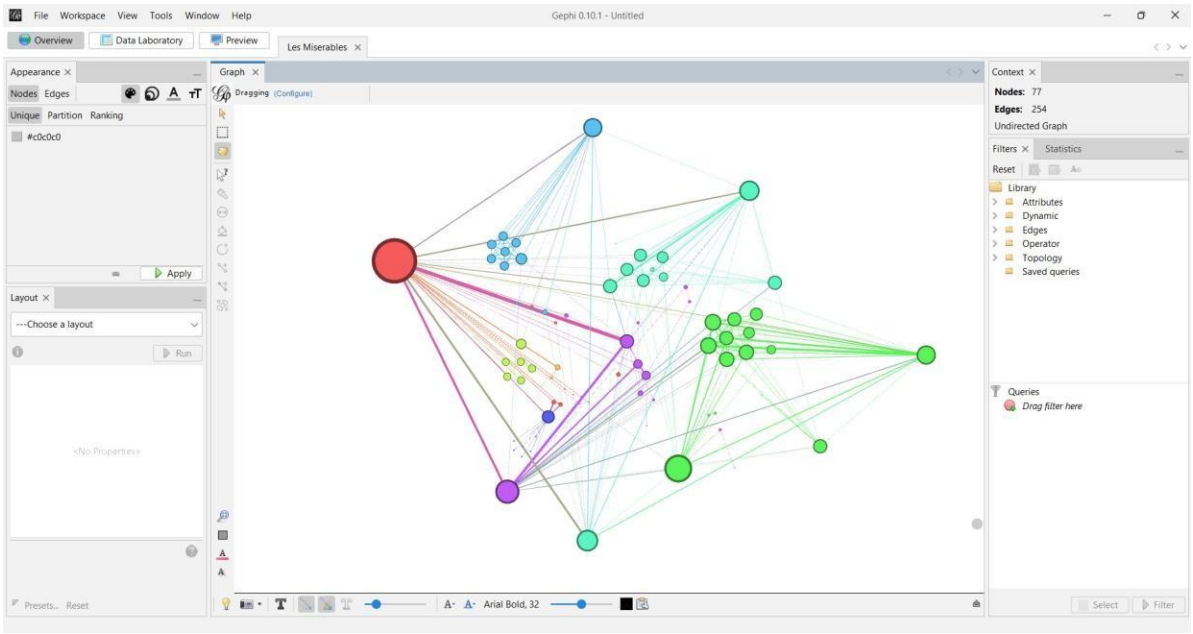


Step 6: Click ok


Import report
Source: Les Miserables
Issues
Report
Nodes
Issues
GEXF version 1.3
INFO
Graph Type:
Undirected
More options...
# of Nodes:
77
# of Edges:
254
Dynamic Graph:
no
Dynamic Attributes:
no
Multi Graph:
no
New workspace
Append to existing workspace
OK
Cancel

Output:


File
Workspace
View
Tools
Window
Help
Gephi 0.10.1 - Untitled
Overview
Data Laboratory
Preview
Les Miserables
Appearance
Nodes
Edges
Unique
Partition
Ranking
Apply
Layout
Choose a layout
Run
Graph
Dragging
Context
Nodes: 77
Edges: 254
Undirected Graph
Filters
Statistics
Reset
Library
Attributes
Dynamic
Edges
Operator
Topology
Saved queries
Queries
Drag filter here
Arial Bold, 32
Select
Filter


File Workspace View Tools Window Help
Gephi 0.10.1 - Untitled
Overview
Data Laboratory
Preview
Les Miserables
Appearance
Nodes Edges
Unique Partition Ranking
#c0c0c0
Apply
Layout
---Choose a layout
Run
<No Properties>
Presets... Reset
Graph
Dragging (Configure)
Arial Bold, 32
Context
Nodes: 77
Edges: 254
Undirected Graph
Filters
Statistics
Reset
Library
Attributes
Dynamic
Edges
Operator
Topology
Saved queries
Queries
Drag filter here
Select
Filter

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT

Thus the explore Social network analysis tools to learn about the users and networks has been successfully executed.

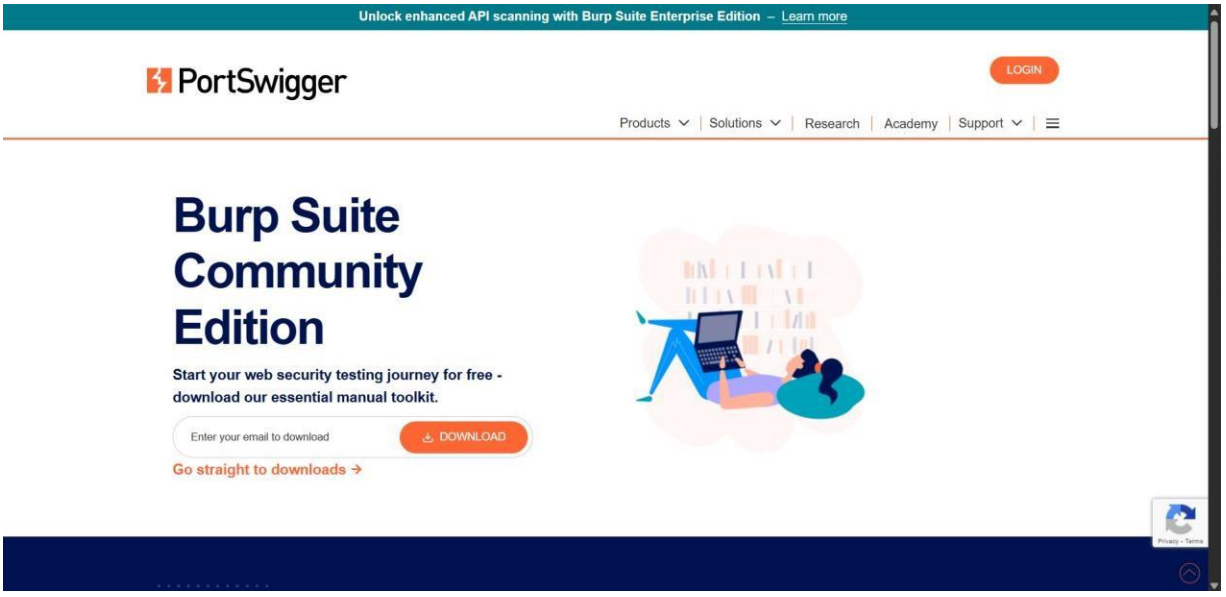
EX:2 Learn a program/tool to illustrate information leakage

Aim: To learn a program to illustrate information leakage

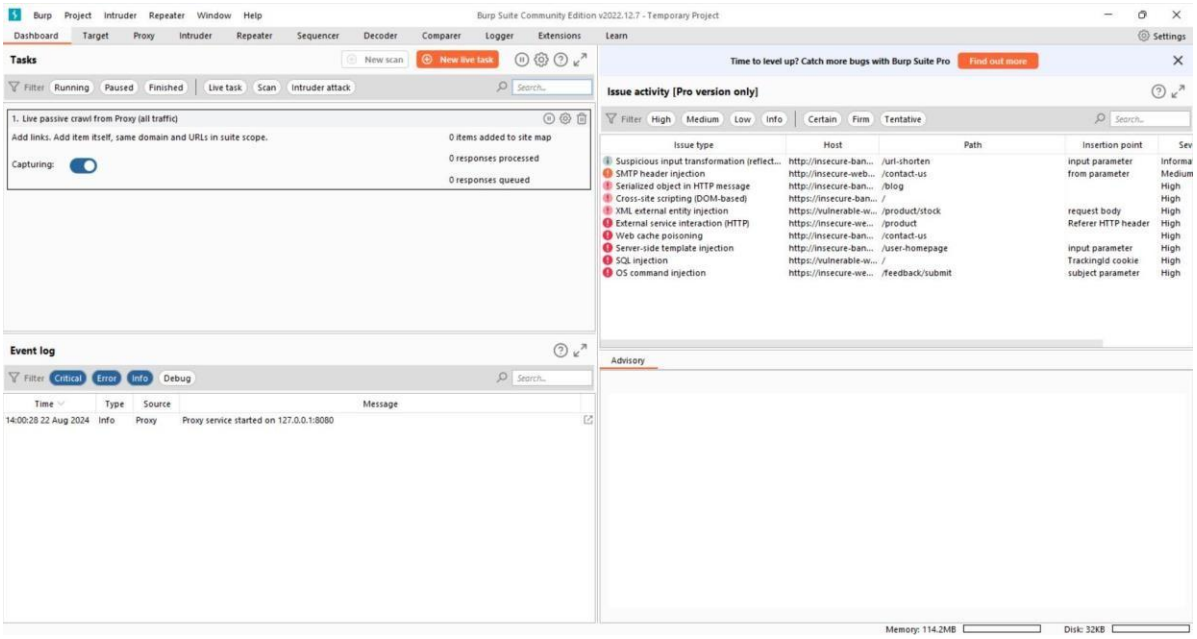
Procedure:

Step 1: Download the tool called Burp Suit using any browser

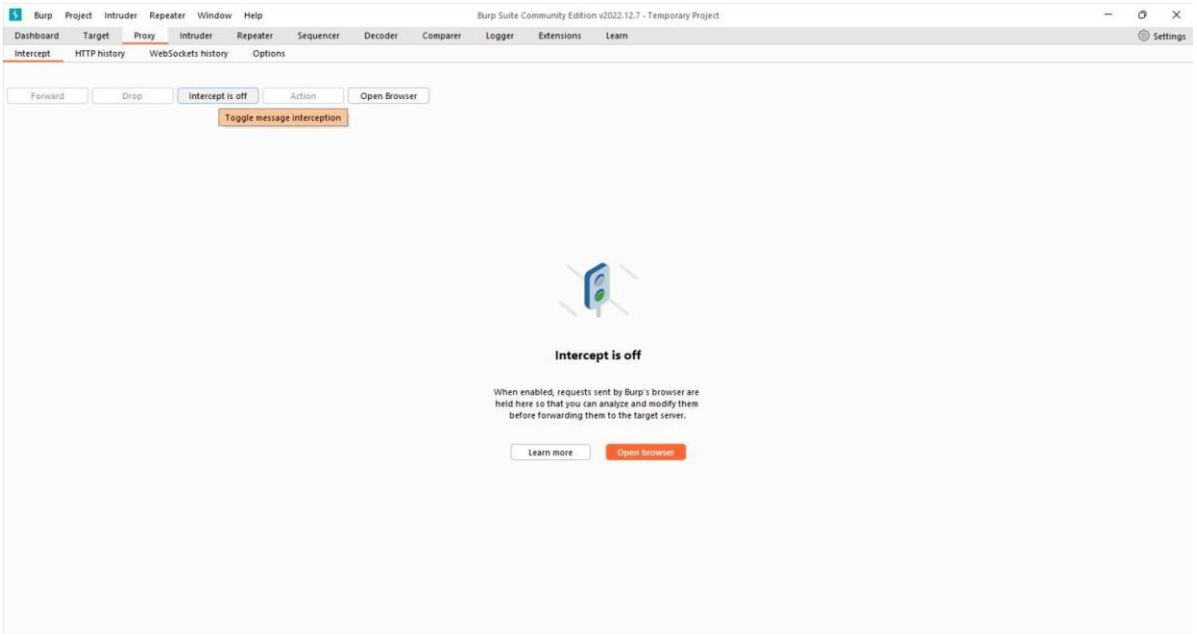
Step 2: Install the Burp Suit Community Edition



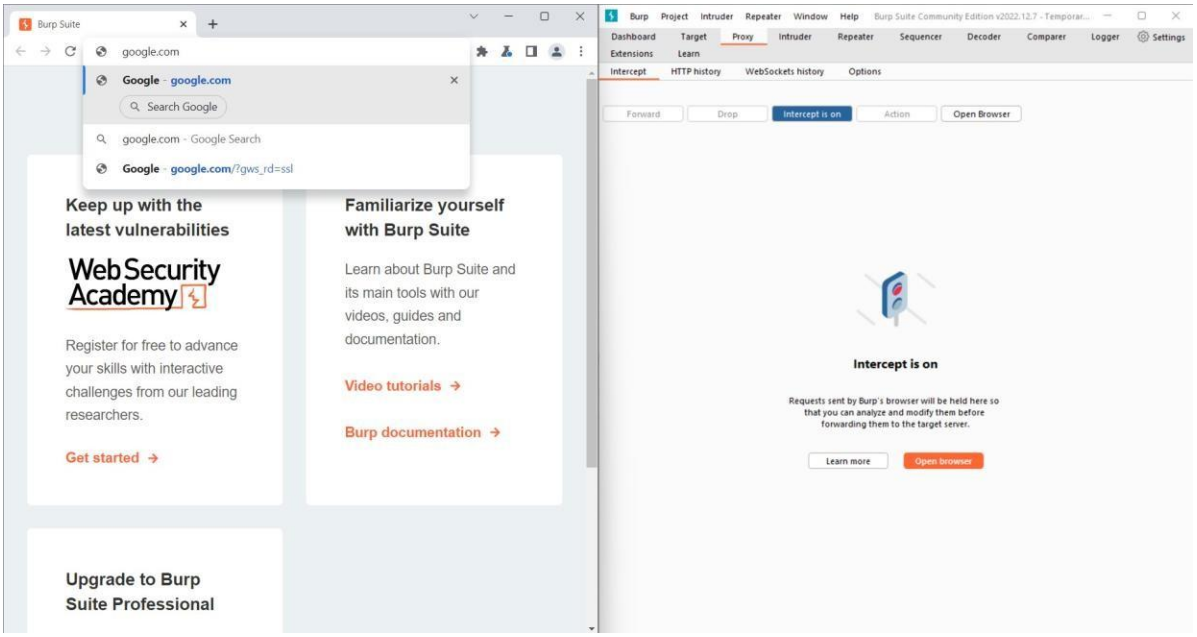
Step 3: Open the Burp Suit Community Edition



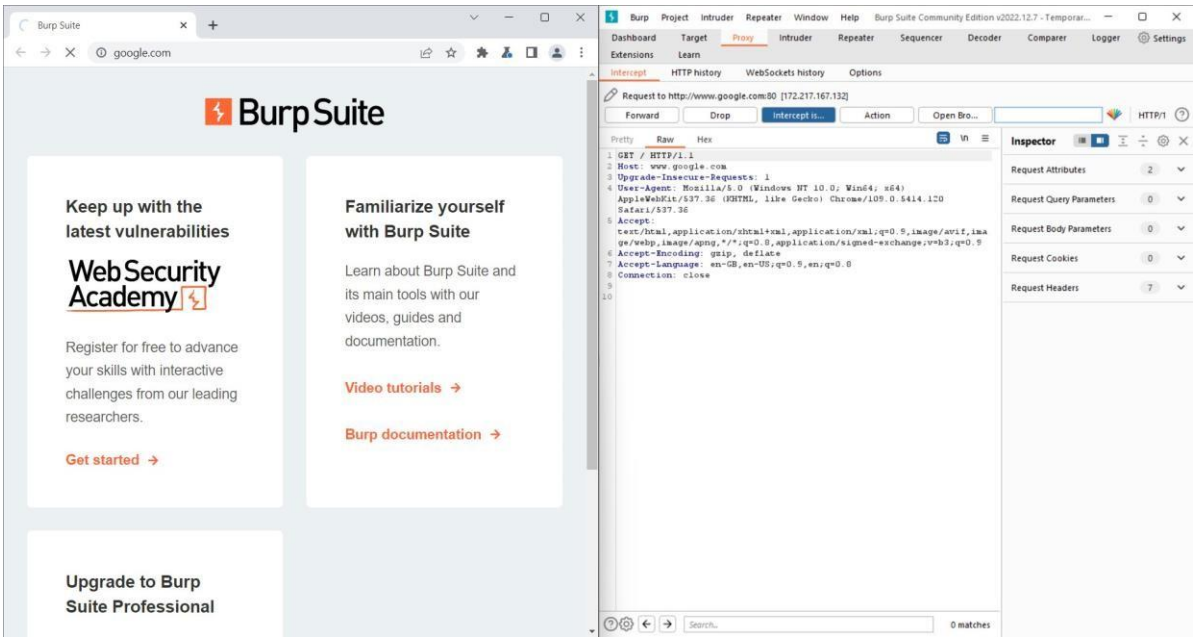
Step 4: Open the Proxy tab in the Burp Suit turn on the Interception and also click the open browser.

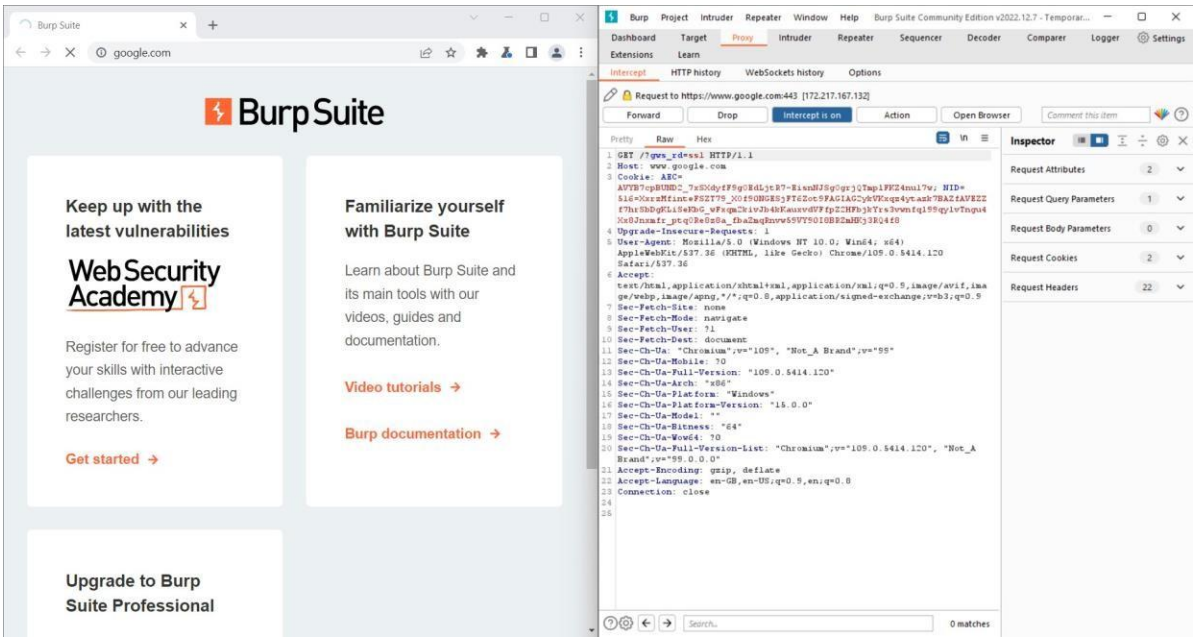


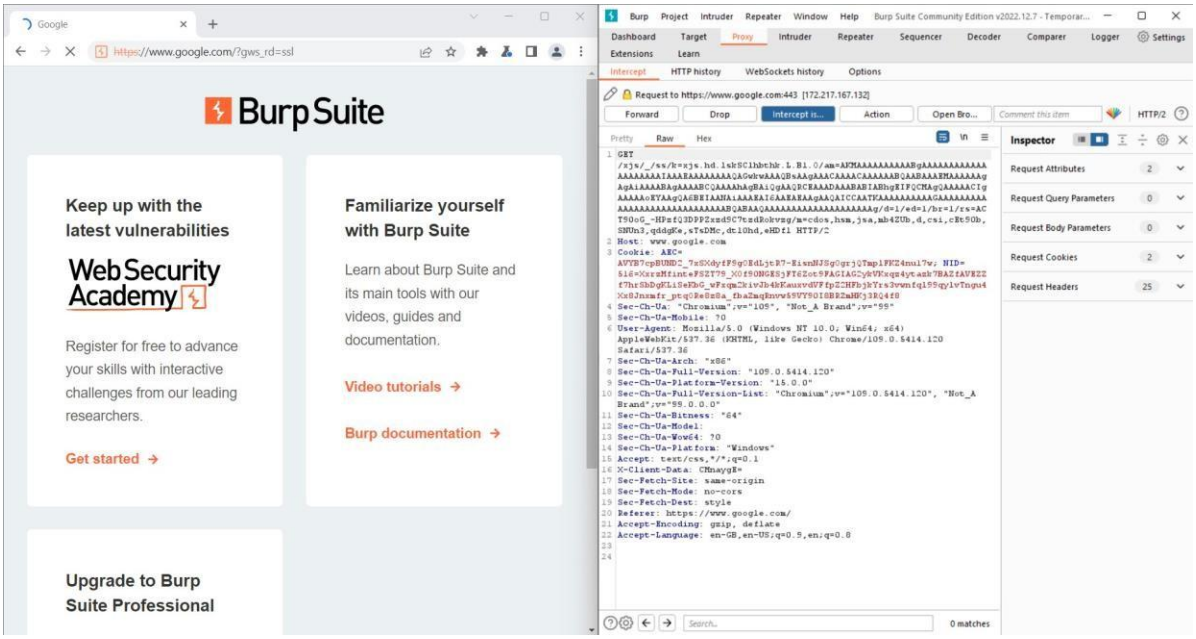
Step 5: Then in the browser open search for google.com

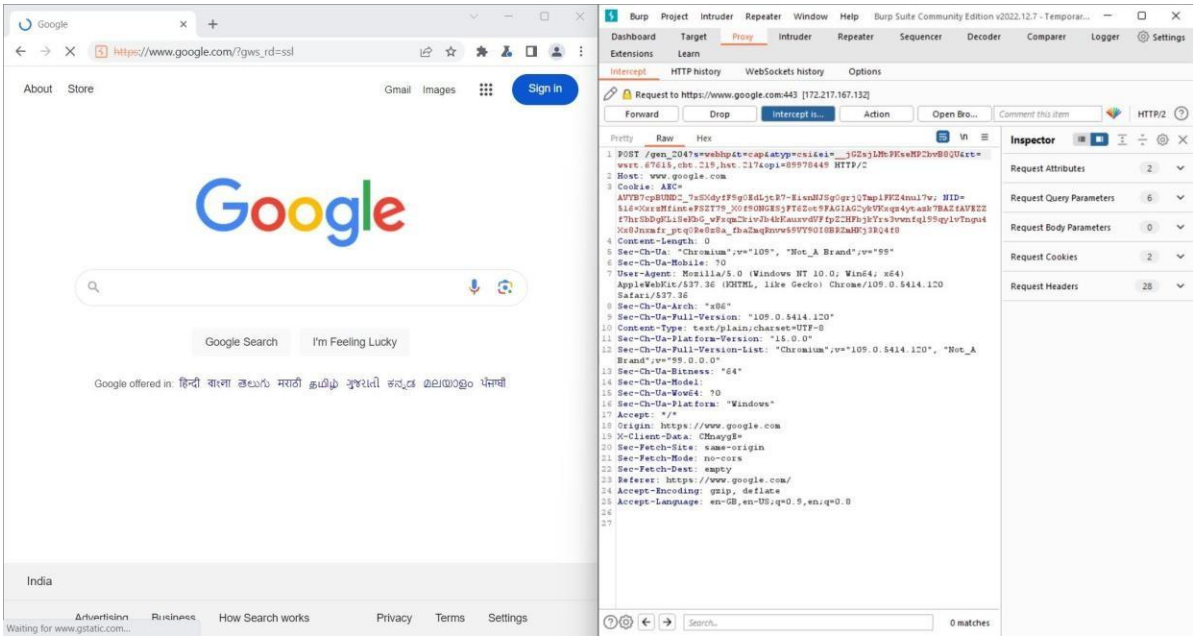


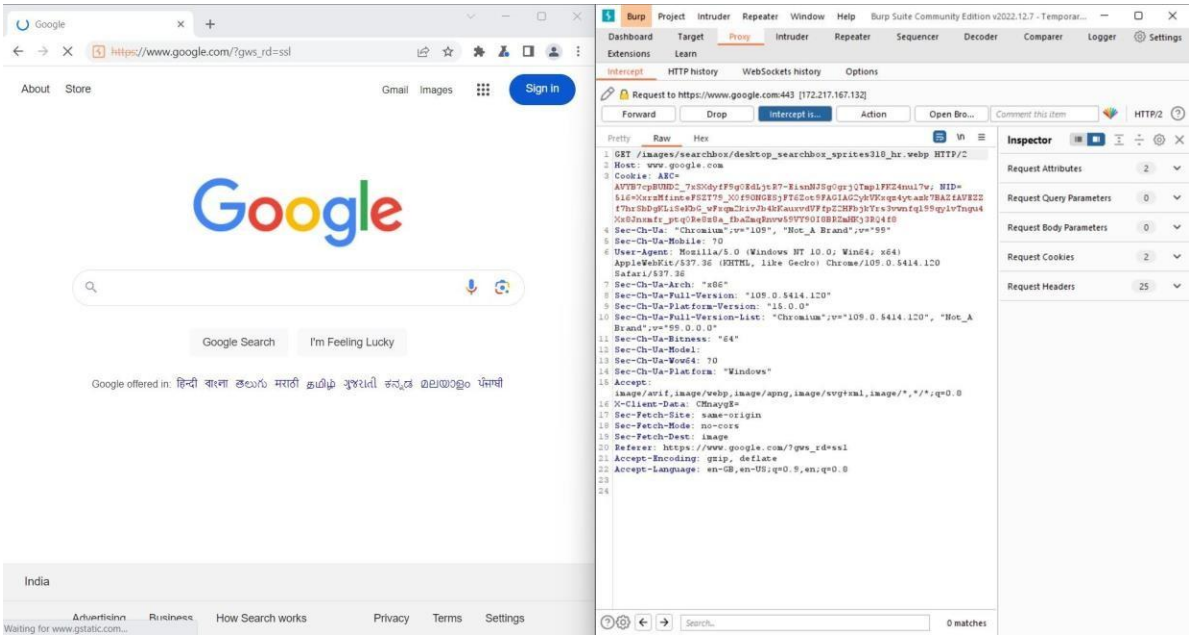
Step 6: After entering it will not load Google. Instead, the tool will stop the methods and it will display all the get, post methods.


Burp Suite
google.com
Burp Suite
Keep up with the latest vulnerabilities
Web Security Academy
Register for free to advance your skills with interactive challenges from our leading researchers.
Get started →
Familiarize yourself with Burp Suite
Learn about Burp Suite and its main tools with our videos, guides and documentation.
Video tutorials →
Burp documentation →
Upgrade to Burp Suite Professional
Burp Project Intruder Repeater Window Help Burp Suite Community Edition v2022.12.7 - Temporar...
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Settings
Extensions Learn
Intercept HTTP history WebSockets history Options
Request to http://www.google.com:80 [172.217.167.132]
Forward Drop Intercept is... Action Open Bro...
Pretty Raw Hex
GET / HTTP/1.1
Host: www.google.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.120 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Connection: close
Inspector
Request Attributes 2
Request Query Parameters 0
Request Body Parameters 0
Request Cookies 0
Request Headers 7
Search...
0 matches


Burp Suite
google.com
Burp Suite
Keep up with the latest vulnerabilities
Web Security Academy
Register for free to advance your skills with interactive challenges from our leading researchers.
Get started →
Familiarize yourself with Burp Suite
Learn about Burp Suite and its main tools with our videos, guides and documentation.
Video tutorials →
Burp documentation →
Upgrade to Burp Suite Professional
Burp Project Intruder Repeater Window Help Burp Suite Community Edition v2022.12.7 - Temporar...
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Settings
Extensions Learn
Intercept HTTP history WebSockets history Options
Request to https://www.google.com:443 [172.217.167.132]
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex
GET /?gws_rd=ssl HTTP/1.1
Host: www.google.com
Cookie: ABC=
AVYB7cpBUHD2_7xSXdyfF9g0EdLjtR7-EisnNJSgOgrjQTmp1FK24nul7w; NID=511=XxrzMfintePSZT79_X0f5ONGE5jFf6Zot9FAGIAG2ykVKrqz4ytaxk7BAZfAVEZZf7hr5hDgKLiSeKbG_wFxqmZkivJb4bKauxvdV9fpZCMPbjkYrs3vwnfq159qylvTngu4Xz0Jnxmfr_ptqOBe0z0a_fbaZaqRnvv65VY5010B2ZmHGj3RQ4f8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.120 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "Chromium";v="109", "Not_A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Full-Version: "109.0.5414.120"
Sec-Ch-Ua-Arch: "x86"
Sec-Ch-Ua-Platform: "Windows"
Sec-Ch-Ua-Platform-Version: "15.0.0"
Sec-Ch-Ua-Model: ""
Sec-Ch-Ua-Bitness: "64"
Sec-Ch-Ua-Wow64: ?0
Sec-Ch-Ua-Full-Version-List: "Chromium";v="109.0.5414.120", "Not_A Brand";v="99.0.0.0"
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Connection: close
Inspector
Request Attributes 2
Request Query Parameters 1
Request Body Parameters 0
Request Cookies 2
Request Headers 22
Search...
0 matches


Burp Suite
Keep up with the latest vulnerabilities
Web Security Academy
Register for free to advance your skills with interactive challenges from our leading researchers.
Get started →
Familiarize yourself with Burp Suite
Learn about Burp Suite and its main tools with our videos, guides and documentation.
Video tutorials →
Burp documentation →
Upgrade to Burp Suite Professional


Google
About
Store
Gmail
Images
Sign in
Google Search
I'm Feeling Lucky
India
How Search works
Privacy
Terms
Settings


Google
https://www.google.com/?gws_rd=ssl
About
Store
Gmail
Images
Sign in
Google
Google Search
I'm Feeling Lucky
Google offered in: हिन्दी বাংলা తెలుగు मराठी தமிழ் ગુજરાતી ಕನ್ನಡ മലയാളം ਪੰਜਾਬੀ
India
Advertising
Business
How Search works
Privacy
Terms
Settings
Waiting for www.gstatic.com...
Burp
Project
Intruder
Repeater
Window
Help
Burp Suite Community Edition v2022.12.7 - Temporar...
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Settings
Extensions
Learn
Intercept
HTTP history
WebSockets history
Options
Request to https://www.google.com:443 [172.217.167.132]
Forward
Drop
Intercept is...
Action
Open Bro...
Comment this item
HTTP/2
Pretty
Raw
Hex
GET /images/searchbox/desktop_searchbox_sprites318_hr.webp HTTP/2
Host: www.google.com
Cookie: ABC=
AVTB7cpBUMD2_7sSXdyfF9gGEdLjtR7-EisnNJSgGgrjGTmp1FKZ4nul7w; NID=
511=XxraMfimeFSZT75_XOf9ONGE8jFT6Zot9FAG1AGCykVKxg4ytaskYBAZfAVE2Z
f7hrSbDgKLiSeXbG_wFnqmJkivJb4kKauxvdVFfpZCHFbjkYrs3vvnfq199qylvTnqu4
Xx8Jnxmfr_ptqORe8z8a_fbaZaqRnvw59VT9OI8BEZmNKj3EQ4i8
Sec-Ch-Ua: "Chromium";v="109", "Not_A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.120
Safari/537.36
Sec-Ch-Ua-Arch: "x86"
Sec-Ch-Ua-Full-Version: "109.0.5414.120"
Sec-Ch-Ua-Platform-Version: "15.0.0"
Sec-Ch-Ua-Full-Version-List: "Chromium";v="109.0.5414.120", "Not_A
Brand";v="99.0.0.0"
Sec-Ch-Ua-Bitness: "64"
Sec-Ch-Ua-Model:
Sec-Ch-Ua-Wow64: ?0
Sec-Ch-Ua-Platform: "Windows"
Accept:
image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
X-Client-Data: CMnaygE=
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: https://www.google.com/?gws_rd=ssl
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
Inspector
Request Attributes
2
Request Query Parameters
0
Request Body Parameters
0
Request Cookies
2
Request Headers
25
Search...
0 matches

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT

Thus to illustrate information leakage has been successfully executed

Ex:3 Experiment a link anonymization technique

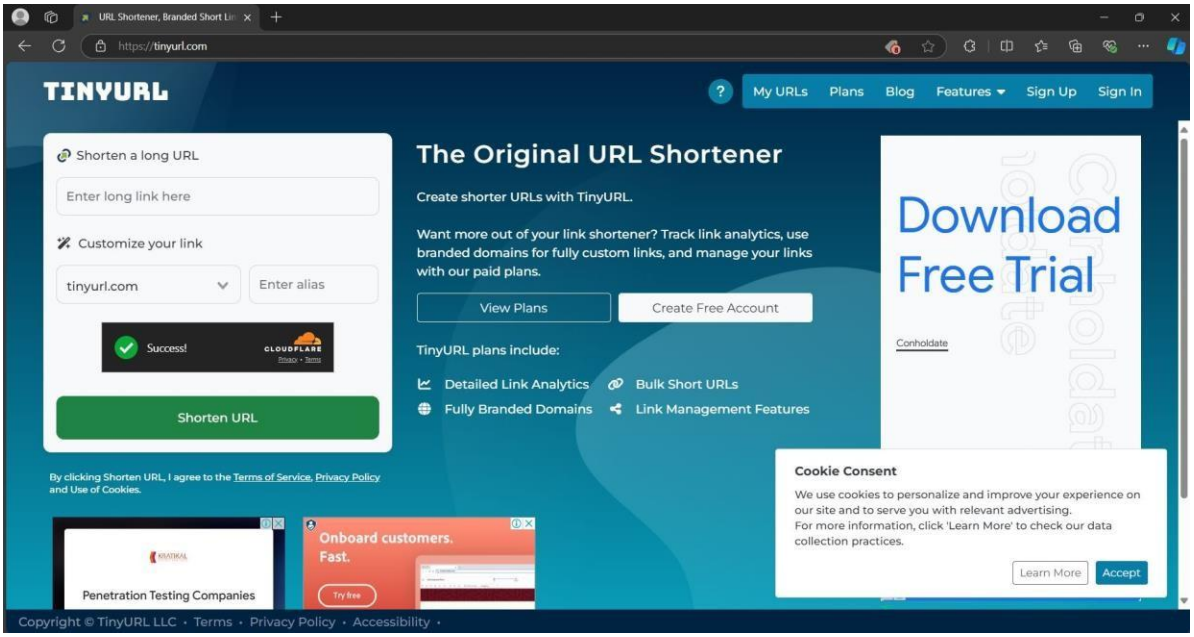
Aim: To Experiment a link anonymization technique using various methods.

Theory: Link anonymization is a technique used to hide the actual destination of a URL, making it difficult for websites or tracking systems to track the origin of clicks.

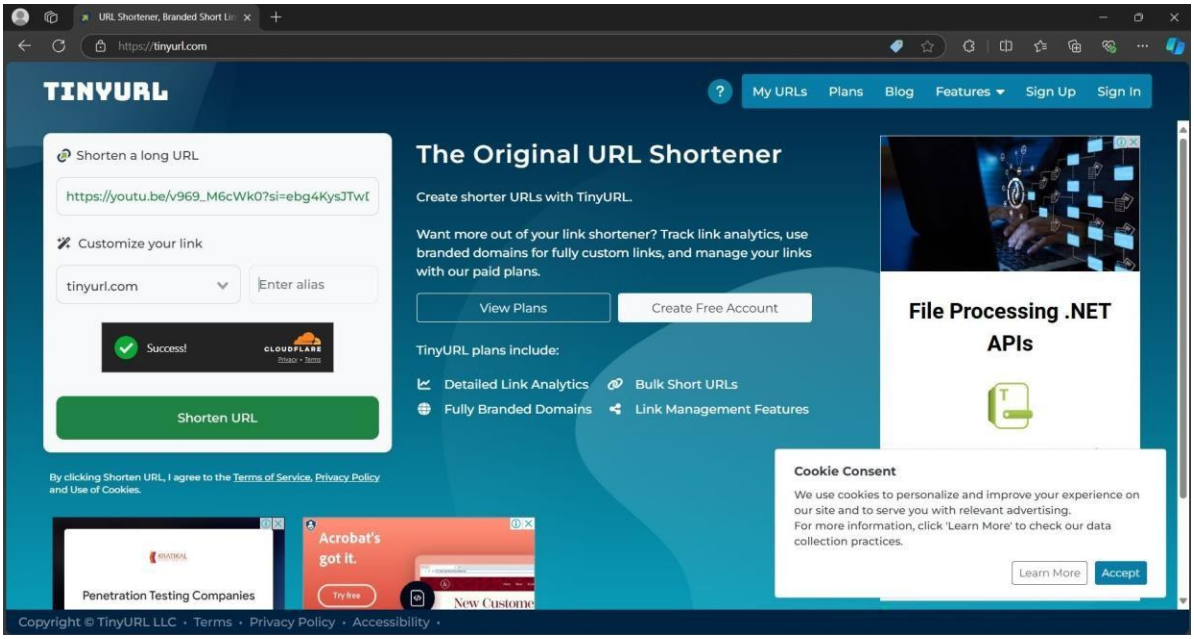
Procedure:

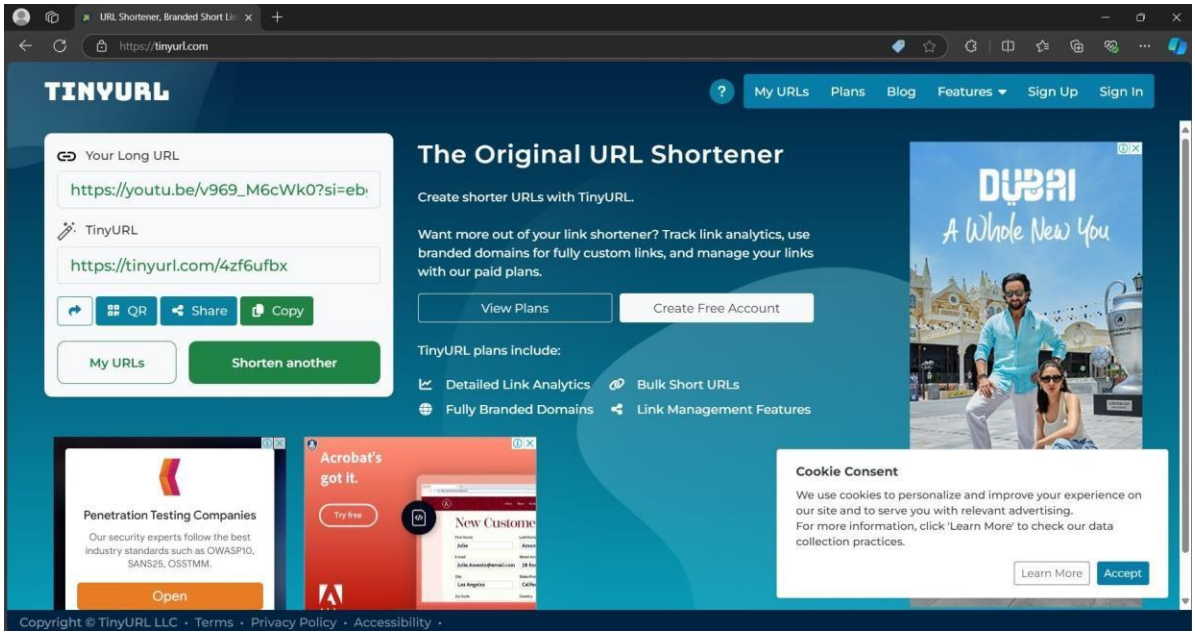
Step 1: Shortening a link using services like Bit.ly, TinyURL, or is. gd can anonymize the URL since the shortened URL hides the destination.

Step 2: For example let us consider the tinyurl for that visit tinyurl.com.



Step 3: Enter the long URL customize the link as you want and click shorten URL.





4: Then we can copy the URL and use it.

Other Method:

Using HTML using href tag. For example, Instagram.com here the link says Instagram.com but when we click, it goes for YouTube.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

RESULT:

Thus, experimenting a link anonymization technique has been successfully executed.

Ex:4 Explore ARX anonymization tool

Aim: To explore the ARX anonymization tool

Theory: ARX (Anonymization Release and Exchange) is a powerful open-source tool designed for anonymizing sensitive personal data. It supports various privacy-preserving techniques, including k-anonymity, l-diversity, t-closeness, and differential privacy. ARX is commonly used for de-identifying data while maintaining analytical utility.

Procedure:

Step 1: ARX is a Java-based application and can be downloaded from its official website. It comes with both a GUI and a command-line interface.

Step 2: After launching ARX, you can load a dataset in CSV format or other formats supported by the tool.

Step 3: Mark sensitive attributes, quasi-identifiers, and other fields in your dataset.

Step 4: Choose the desired anonymization technique and configure the parameters like the *k* value for k-anonymity.

Step 5: Use ARX's visual tools to analyze the trade-offs between privacy and utility before finalizing the anonymization.

Step 6: Once satisfied with the results, export the anonymized dataset for use.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, the Exploration ARX anonymization tool has been successfully executed.

Ex:5 Implement a program for network access control to illustrate malware attacks

Aim: To Implement a program for network access control to illustrate malware attacks.

Theory: To illustrate malware attacks in the context of network access control, you can implement a basic network simulation where you demonstrate how malicious entities could attempt to infiltrate a network. The program will include:

1. **Simulated Network Access Control (NAC)**: A system that monitors and controls access to the network based on authentication.
2. **Malware Simulation**: A basic simulation of how malware might behave to compromise a network (like a denial-of-service attack or phishing).

Procedure:

Network Structure:

- A few simulated devices connected to the network.
- Access control is based on device authentication (like MAC address).

Malware Attack Simulation:

- Malware could either try to overload the network (DoS attack) or spoof an authenticated device to bypass access control.

Program :

```
import random
import time
# Simulated Devices in the network with MAC addresses
devices = {
   "Device1": "00:0a:95:9d:68:16",
   "Device2": "00:0a:95:9d:68:17",
   "Device3": "00:0a:95:9d:68:18",
}
# Network Access Control List (ACL)
authorized_devices = {
   "00:0a:95:9d:68:16", # Device1
   "00:0a:95:9d:68:17",  # Device2
}
# Simulated malware MAC address trying to access the network
```

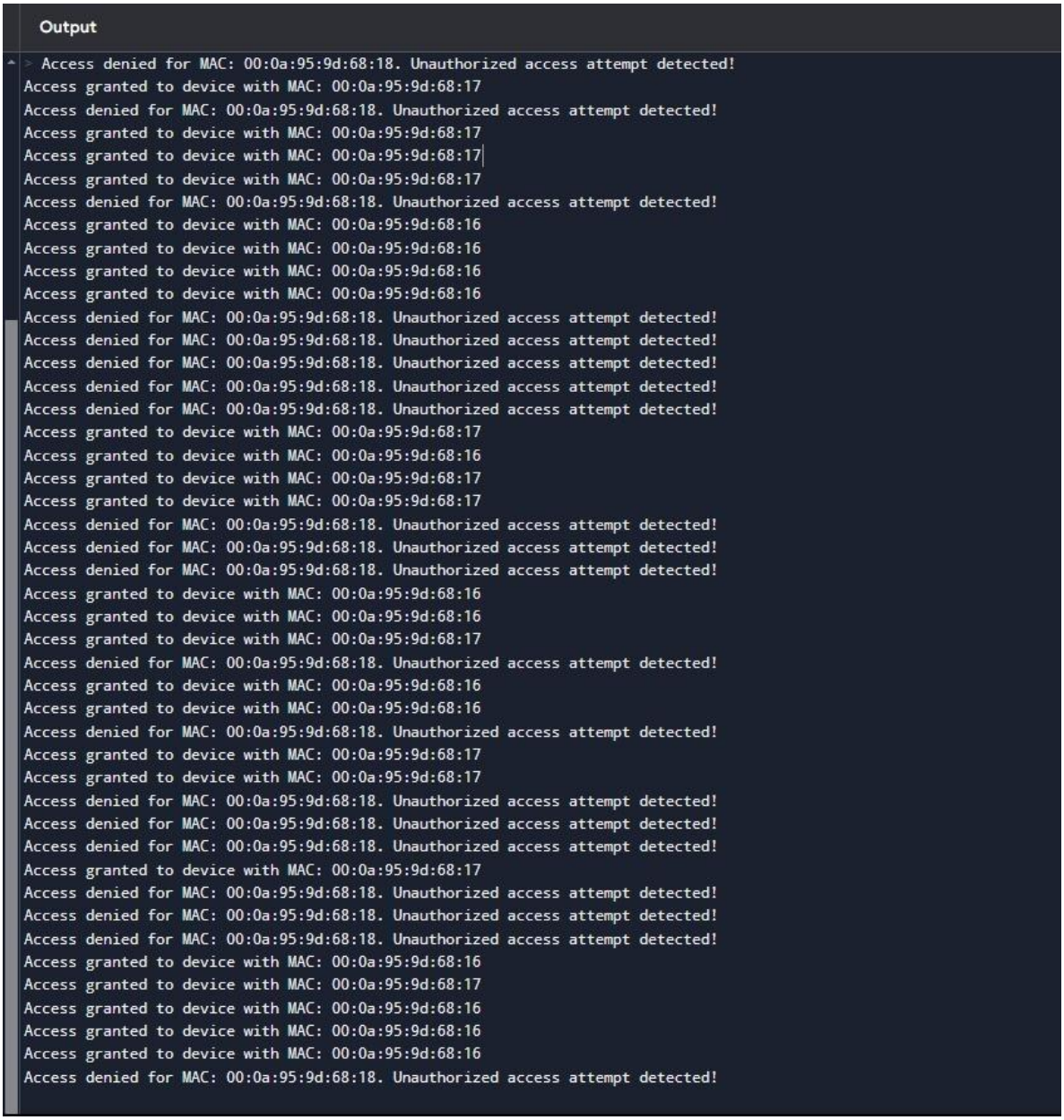
```
malware_device = "00:0a:95:9d:68:99"  # Spoofed MAC address
def access_network(device_mac):
    """Simulates a device trying to access the network."""
    if device_mac in authorized_devices:
        print(f"Access granted to device with MAC: {device_mac}")
        return True
    else:
        print(f"Access denied for MAC: {device_mac}. Unauthorized access attempt detected!")
        return False
def simulate_normal_device_access():
    """Simulates legitimate devices accessing the network."""
    while True:
        device = random.choice(list(devices.values()))
        access_network(device)
        time.sleep(random.randint(1, 3))  # Delay between accesses
def simulate_malware_attack():
    """Simulates malware trying to bypass NAC."""
    print("\n--- Malware Attack Simulation Started ---")
    for _ in range(5):  # Malware tries multiple times
        access_granted = access_network(malware_device)
        if not access_granted:
            print("Malware trying again...")
        time.sleep(1)
def simulate_dos_attack():
    """Simulates a Denial of Service attack."""
    print("\n--- DoS Attack Simulation Started ---")
    for _ in range(10):  # Rapid access attempts
        access_network(malware_device)
        time.sleep(0.5)
if __name__ == "__main__":
```

```
print("Simulating normal network access:")
simulate_normal_device_access()
# Simulate a malware attack after a short delay time.sleep(5)
simulate_malware_attack()
# Simulate a DoS attack after another short delay time.sleep(5)
simulate_dos_attack()
```

Network Access Control: The access_network() function simulates access control, where only authorized devices can connect. Each device has a MAC address.

Malware Simulation:

- The simulate_malware_attack() function simulates an attacker using a spoofed MAC address (malware) trying to gain unauthorized access to the network.
- The simulate_dos_attack() function demonstrates a denial-of-service (DoS) attack where the malware continuously bombards the network with requests to overload it.


Output
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:17
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:17
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access granted to device with MAC: 00:0a:95:9d:68:16
Access denied for MAC: 00:0a:95:9d:68:18. Unauthorized access attempt detected!

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, the program to implement network access control to illustrate malware attacks has been successfully executed.

Ex:6 Create a simple social network application to show authentication mechanisms

Aim: To create a simple social network application to show authentication mechanisms

Steps to Implement:

1. **User Registration**: Store user details securely (passwords should be hashed before storing in the database).
2. **User Login**: Authenticate using stored credentials.
3. **Session Management**: Use Flask's session management to keep track of logged-in users.
4. **Post Feature**: Allow users to post and display posts on the social feed.

Procedure:

Step 1: **Install Required Libraries**

First, install the necessary dependencies:

pip install Flask Flask-Session bcrypt

Step 2: **Create the Application Structure**

Your project structure should look like this:

/social_network_app

/static

/templates

app.py

database.db

Step 3: Flask App (app.py)

```
from flask import Flask, render_template, request, redirect, session, flash
from flask_session import Session
import sqlite3
from bcrypt import hashpw, checkpw, gensalt

app = Flask(__name__)
app.secret_key = 'supersecretkey'  # Secret for session management
app.config['SESSION_TYPE'] = 'filesystem'
```

```
Session(app)

# Database setup (use SQLite for simplicity)
def init_db():
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS users
                      (id INTEGER PRIMARY KEY, username TEXT, email TEXT, password
TEXT)''')
    cursor.execute('''CREATE TABLE IF NOT EXISTS posts
                      (id INTEGER PRIMARY KEY, user_id INTEGER, content TEXT)''')
    conn.commit()
    conn.close()

# Register new user
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        email = request.form['email']
        password = request.form['password']
        hashed_password = hashpw(password.encode('utf-8'), gensalt())

        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO users (username, email, password) VALUES (?, ?, ?)",
                       (username, email, hashed_password))
        conn.commit()
        conn.close()
```



```
        flash('Registration successful. Please log in.')
```

```
        return redirect('/login')
    return render_template('register.html')

# User login
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE email=?", (email,))
        user = cursor.fetchone()
        conn.close()

        if user and checkpw(password.encode('utf-8'), user[3].encode('utf-8')):
            session['user_id'] = user[0]
            session['username'] = user[1]
            flash('Login successful!')
            return redirect('/dashboard')
        else:
            flash('Invalid login credentials.')
    return render_template('login.html')

# User dashboard (protected route)
@app.route('/dashboard', methods=['GET', 'POST'])
def dashboard():
    if 'user_id' not in session:
        flash('You need to log in first!')
```

```
        return redirect('/login')

    if request.method == 'POST':
        content = request.form['content']
        conn = sqlite3.connect('database.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO posts (user_id, content) VALUES (?, ?)",
                       (session['user_id'], content))
        conn.commit()
        conn.close()

    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute('''SELECT users.username, posts.content FROM posts
                      JOIN users ON users.id = posts.user_id''')
    posts = cursor.fetchall()
    conn.close()

    return render_template('dashboard.html', posts=posts)

# Logout user
@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.')
    return redirect('/login')

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Step 4: **HTML Templates**

register.html

Place the following HTML files in the /templates folder:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
</head>
<body>
    <h2>Register</h2>
    <form method="POST" action="/register">
        <label>Username:</label>
        <input type="text" name="username" required><br>
        <label>Email:</label>
        <input type="email" name="email" required><br>
        <label>Password:</label>
        <input type="password" name="password" required><br>
        <button type="submit">Register</button>
    </form>
    <a href="/login">Already have an account? Login here.</a>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <form method="POST" action="/login">
        <label>Email:</label>
        <input type="email" name="email" required><br>
        <label>Password:</label>
        <input type="password" name="password" required><br>
        <button type="submit">Login</button>
    </form>
    <a href="/register">Don't have an account? Register here.</a>
</body>
</html>
```

dashboard.html

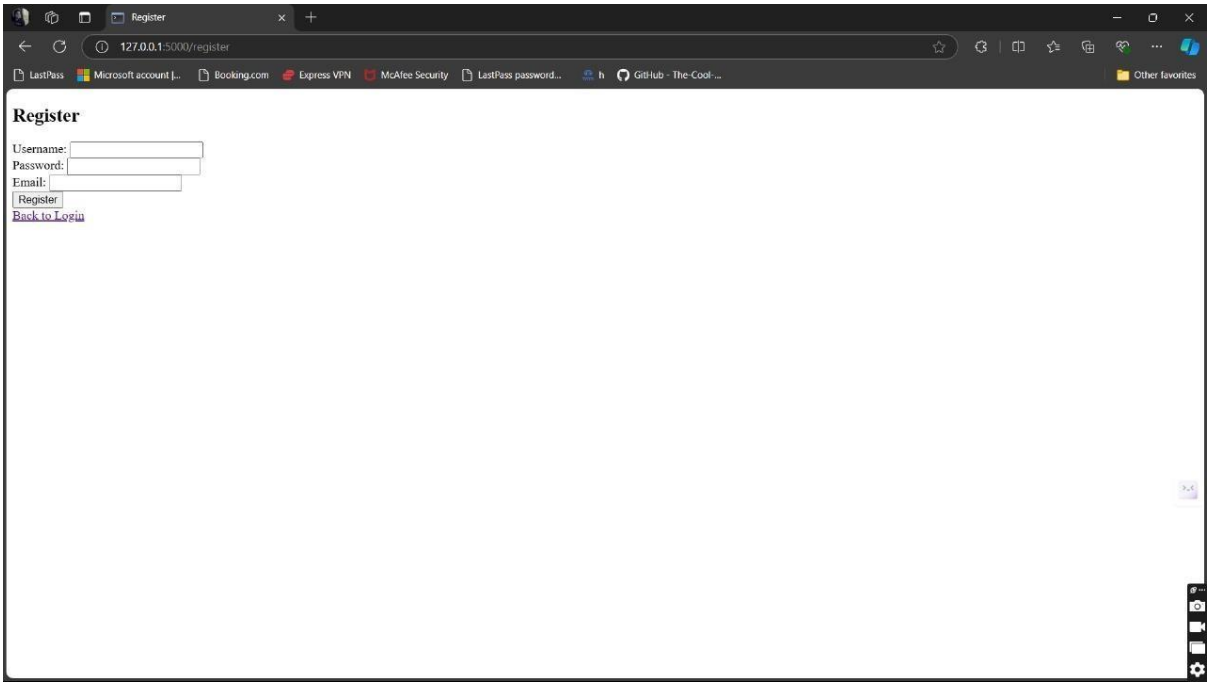
```
        <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard</title>
</head>
<body>
    <h2>Welcome, {{ session['username'] }}!</h2>
    <a href="/logout">Logout</a>

    <h3>Create a new post:</h3>
    <form method="POST" action="/dashboard">
        <textarea name="content" rows="4" cols="50" required></textarea><br>
        <button type="submit">Post</button>
```

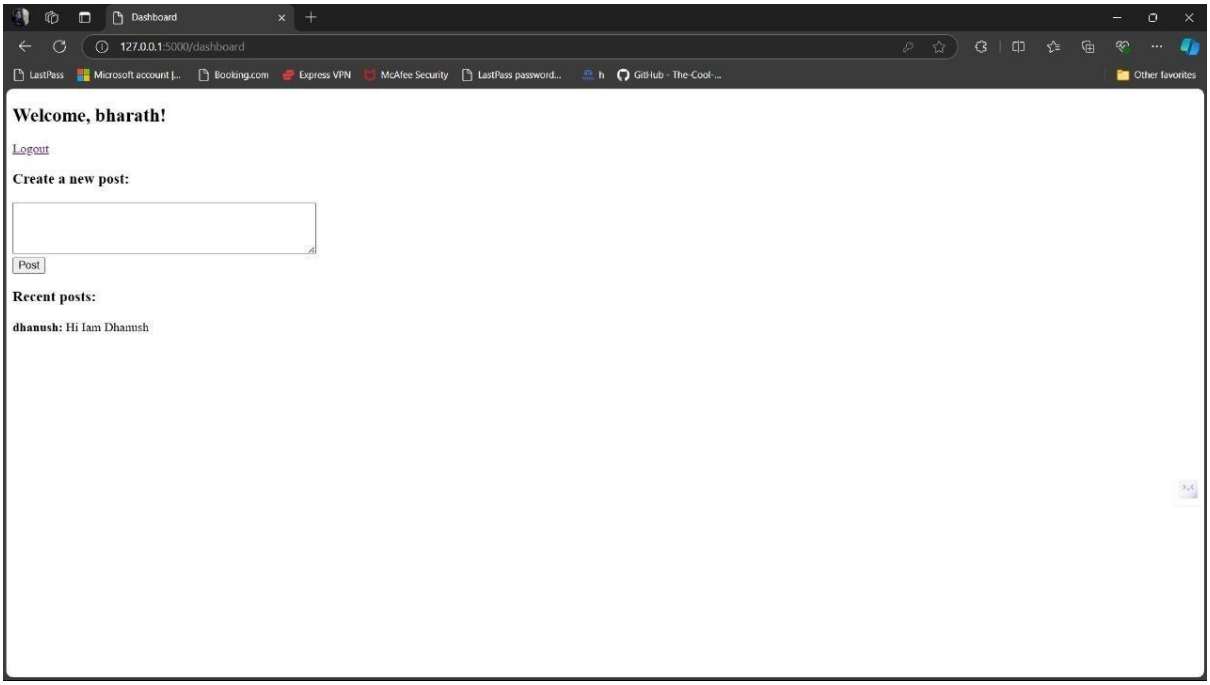
```
    </form>

    <h3>Recent posts:</h3>
    {% for post in posts %}
        <p><strong>{{ post[0] }}:</strong> {{ post[1] }}</p>
    {% endfor %}
</body>
</html>
```

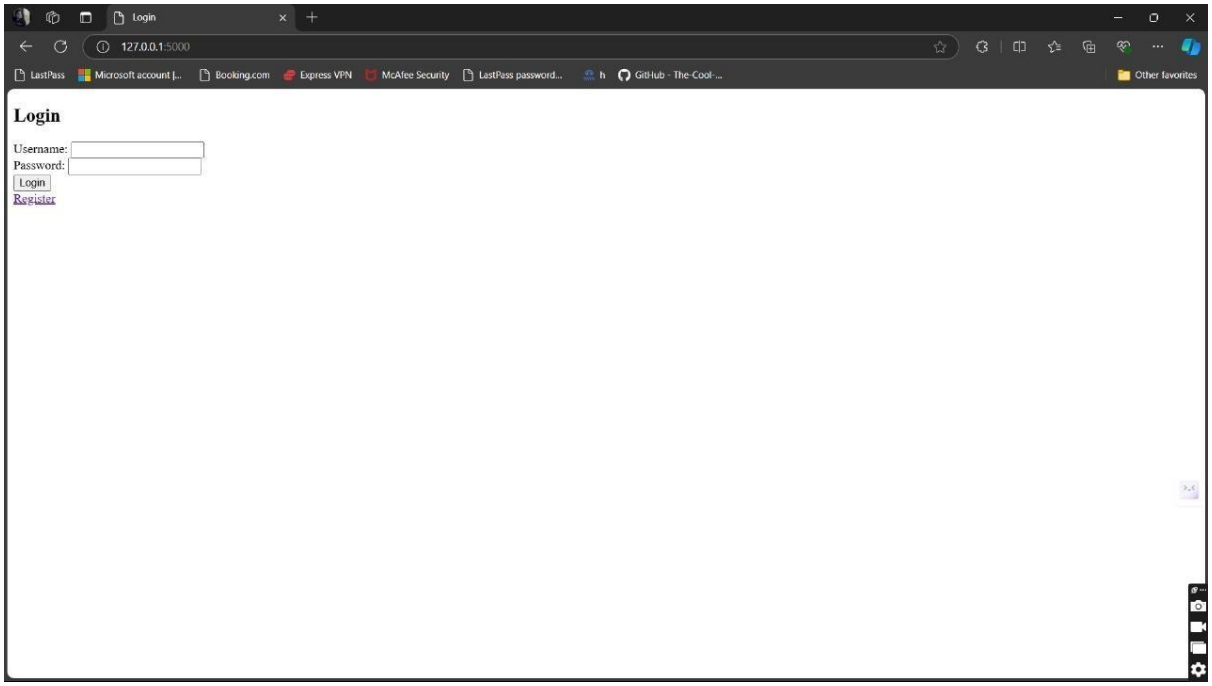
Step 5: **Run the Application**

To run the Flask application, navigate to the project folder and start the server: python app.py

Visit the app at http://127.0.0.1:5000/register to test the registration and authentication mechanisms




Dashboard
127.0.0.1:5000/dashboard
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
GitHub - The-Cool-...
Other favorites
Welcome, bharath!
Logout
Create a new post:
Post
Recent posts:
dhanush: Hi Iam Dhanush


Login
127.0.0.1:5000
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
GitHub - The-Cool-...
Other favorites
Login
Username:
Password:
Login
Register

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, to create a simple social network application to show authentication mechanisms has been successfully executed.

Ex:7 Create an application for the following scenario: "Social networking users are presented with two apparently similar emails or websites. They must first identify the differences between them and then decide which one is a scam attempting to steal their information or money.

Theory: To create an application where social networking users can identify differences between two similar emails or websites to determine which one is a scam, we can develop a simple web application. This application will present two versions of an email or website and prompt the user to identify the differences, eventually revealing which one is the scam.

Implementation Steps

1. **Create the Backend**: Set up a Flask application to serve the content.
2. **Prepare Sample Emails/Websites**: Create JSON data to simulate different emails or website layouts.
3. **Build the Frontend**: Create a simple interface to display the comparison and allow user interaction.

Procedure:

Step 1: Set Up the Project Structure

/scam_identifier_app

/static

style.css

/templates

index.html

result.html

app.py

data.json

Step 2: Create Sample Data (data.json)

Here's a sample JSON file with two sets of similar emails or websites:

[

```
{
  "id": 1,
  "email_a": {
    "subject": "Your Account Has Been Compromised",
    "body": "Dear User, we noticed unusual activity in your account. Click here to verify31 your identity: http://fake-link.com",
    "scam": true
```

```
    },
    "email_b": {
      "subject": "Important Security Update",
      "body": "Dear User, please confirm your identity to ensure your account's security: http://legit-link.com",
      "scam": false
    }
  },
  {
    "id": 2,
    "email_a": {
      "subject": "Congratulations! You've Won a Prize!",
      "body": "Click here to claim your $1000 prize: http://scam-link.com",
      "scam": true
    },
    "email_b": {
      "subject": "Your Monthly Newsletter",
      "body": "Here's your latest newsletter with updates: http://trusted-news.com",
      "scam": false
    }
  }
]
```

Step 3: Create the Flask Application (app.py)

```
    from flask import Flask, render_template, request, redirect, url_for, flash
import json
app = Flask(_name_)
app.secret_key = 'supersecretkey'
# Load email data from JSON
def load_data():
```

```
    with open('data.json') as f:
        return json.load(f)

@app.route('/')
def index():
    data = load_data()
    return render_template('index.html', emails=data)

@app.route('/result', methods=['POST'])
def result():
    email_id = int(request.form['email_id'])
    user_choice = request.form['user_choice']
    data = load_data()
    email_data = data[email_id - 1]

    scam_email = email_data['email_a'] if email_data['email_a']['scam'] else
email_data['email_b']

    if user_choice == scam_email['subject']:
        result_message = "Correct! You identified the scam."
    else:
        result_message = f"Wrong! The scam email was: {scam_email['subject']}."

    return render_template('result.html', result_message=result_message)

if __name__ == '__main__':
    app.run(debug=True)
```

Step 4: **Create the HTML**

Templates index.html


```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scam Identifier</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <h1>Identify the Scam!</h1>
    <form method="POST" action="/result">
        {% for email in emails %}
            <h2>Email Comparison {{ loop.index }}</h2>
            <div class="email">
                <div class="email-box">
                    <h3>Email A</h3>
                    <p><strong>Subject:</strong> {{ email.email_a.subject }}</p>
                    <p><strong>Body:</strong> {{ email.email_a.body }}</p>
                </div>
                <div class="email-box">
                    <h3>Email B</h3>
                    <p><strong>Subject:</strong> {{ email.email_b.subject }}</p>
                    <p><strong>Body:</strong> {{ email.email_b.body }}</p>
                </div>
            </div>
            <label for="user_choice">Select the scam email:</label>
            <select name="user_choice" required>
                <option value="{{ email.email_a.subject }}">{{ email.email_a.subject
}}</option>
                <option value="{{ email.email_b.subject }}">{{ email.email_b.subject
}}</option>
            </select>
```

```
            <input type="hidden" name="email_id" value="{{ email.id }}">
            <button type="submit">Submit</button>
        {% endfor %}
    </form>
</body>
</html>
```

Step 4: **result.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Result</title>
</head>
<body>
    <h1>Result</h1>
    <p>{{ result_message }}</p>
    <a href="/">Try Again</a>
</body>
</html>
```

Step5: Create a Simple CSS File (style.css) body {

```
    font-family: Arial, sans-serif; margin: 20px;
    padding: 20px; background-color: #f4f4f4;
}
h1, h2, h3 { color: #333;
}
email {
    display: flex;
    justify-content: space-between; margin-bottom:
    20px;
}
```

```
.email-box { width: 45%;
    border: 1px solid #ccc; padding: 10px;
    background-color: #fff;
    box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.1);
}
button {
    margin-top: 10px; padding: 10px 15px;
    background-color: #28a745; color: white;
    border: none; cursor: pointer;
}
button:hover {
    background-color: #218838;
}
```

Step 6: To run the Flask application, navigate to the project folder and start the server python app.py

Visit the app at http://127.0.0.1:5000/ to test the email/website comparison functionality.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, the application for the following scenario: “Social networking users are presented with two similar emails or websites. They must first identify the differences between them and then decide which one is a scam attempting to steal their information or money has been successfully executed.

Ex:8 Implement a program in Python to estimate the trust of social network users' group

Aim: To implement a program in Python to estimate the trust of social network user groups.

Implementation:

```
import random

class User:
    def _init_(self, user_id):
        self.user_id = user_id
        self.trust_score = 1.0  # Trust score starts at 1.0 (neutral)
        self.interactions = []  # Store interactions

    def add_interaction(self, other_user, is_positive):
        """Add an interaction with another user."""
        self.interactions.append((other_user.user_id, is_positive))
        self.update_trust_score(other_user, is_positive)

    def update_trust_score(self, other_user, is_positive):
        """Update the trust score based on interactions."""
        if is_positive:
            self.trust_score += 0.1  # Increase trust score for positive interaction
            other_user.trust_score += 0.05  # Increase the other user's trust score slightly
        else:
            self.trust_score -= 0.1  # Decrease trust score for negative interaction
            other_user.trust_score -= 0.05  # Decrease the other user's trust score slightly

    def __repr__(self):
        return f"User({self.user_id}, Trust Score: {self.trust_score:.2f})"
```

```
class SocialNetwork:
    def __init__(self):
        self.users = {}
    def add_user(self, user_id):
        """Add a new user to the network."""
        if user_id not in self.users:
            self.users[user_id] = User(user_id)
    def simulate_interaction(self, user_id_a, user_id_b):
        """Simulate an interaction between two users."""
        if user_id_a in self.users and user_id_b in self.users:
            is_positive = random.choice([True, False])  # Randomly determine interaction type
            self.users[user_id_a].add_interaction(self.users[user_id_b], is_positive)
            interaction_type = "positive" if is_positive else "negative"
            print(f"{user_id_a} had a {interaction_type} interaction with {user_id_b}.")
    def display_trust_scores(self):
        """Display the trust scores of all users."""
        for user in self.users.values():
            print(user)
# Example usage
if __name__ == "__main__":
    social_network = SocialNetwork()
    # Add users
    for i in range(5):
        social_network.add_user(f"User{i + 1}")
    # Simulate interactions
    for _ in range(10):  # Simulate 10 random interactions
        user_a = random.choice(list(social_network.users.keys()))
        user_b = random.choice(list(social_network.users.keys()))
        while user_a == user_b:  # Ensure different users interact
            user_b = random.choice(list(social_network.users.keys()))
```

```
    social_network.simulate_interaction(user_a, user_b)
# Display trust scores
print("\nTrust Scores:")
social_network.display_trust_scores()
```

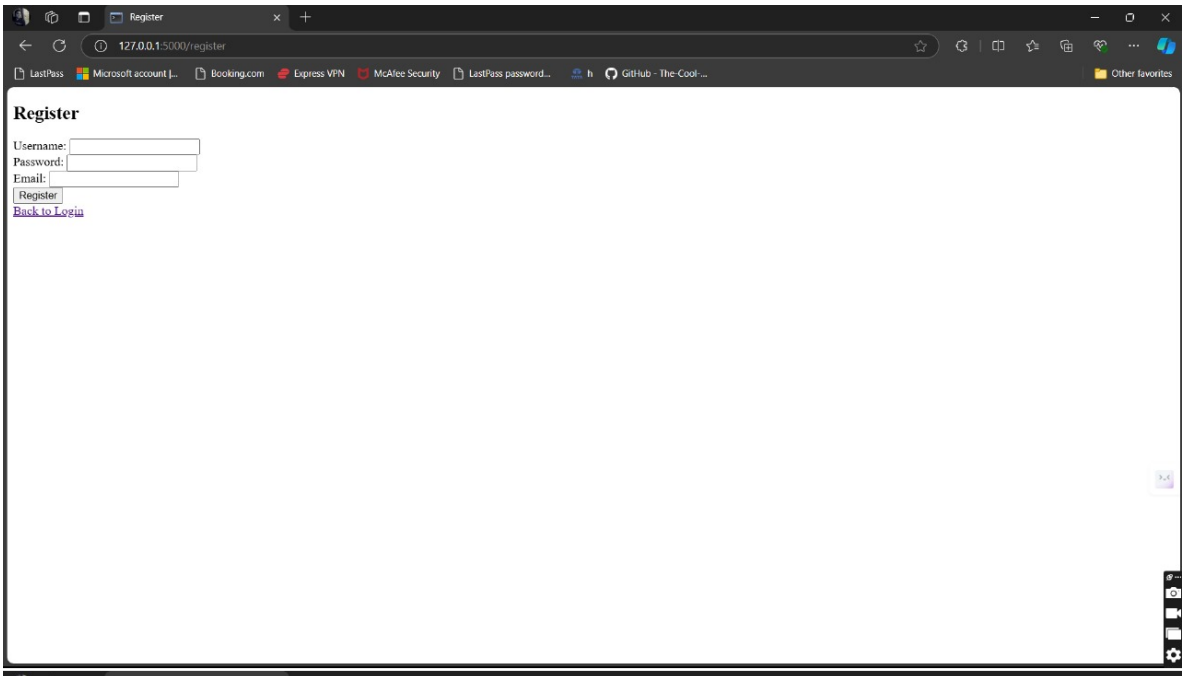
Explanation of the Code

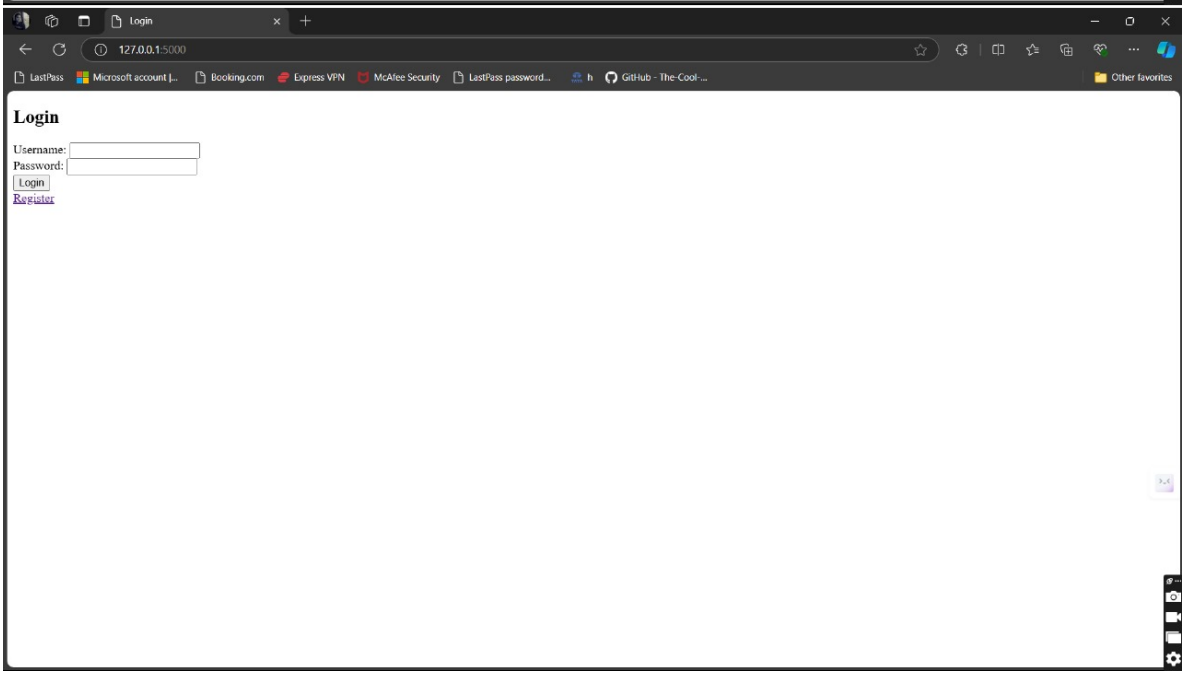
1. **User Class**: Represents each user, including a trust score and a list of interactions.
    - add_interaction: Adds an interaction with another user and updates trust scores based on whether the interaction was positive or negative.
    - update_trust_score: Adjusts the trust score based on the interaction type.
    - __repr__: Provides a string representation of the user for easy viewing.
2. **SocialNetwork Class**: Manages the users and interactions within the social network.
    - add_user: Adds a new user to the network.
    - simulate_interaction: Randomly simulates an interaction between two users, deciding whether it's positive or negative.
    - display_trust_scores: Displays the trust scores of all users in the network.
3. **Main Block**: Initializes a social network, adds users, simulates random interactions, and displays trust scores.

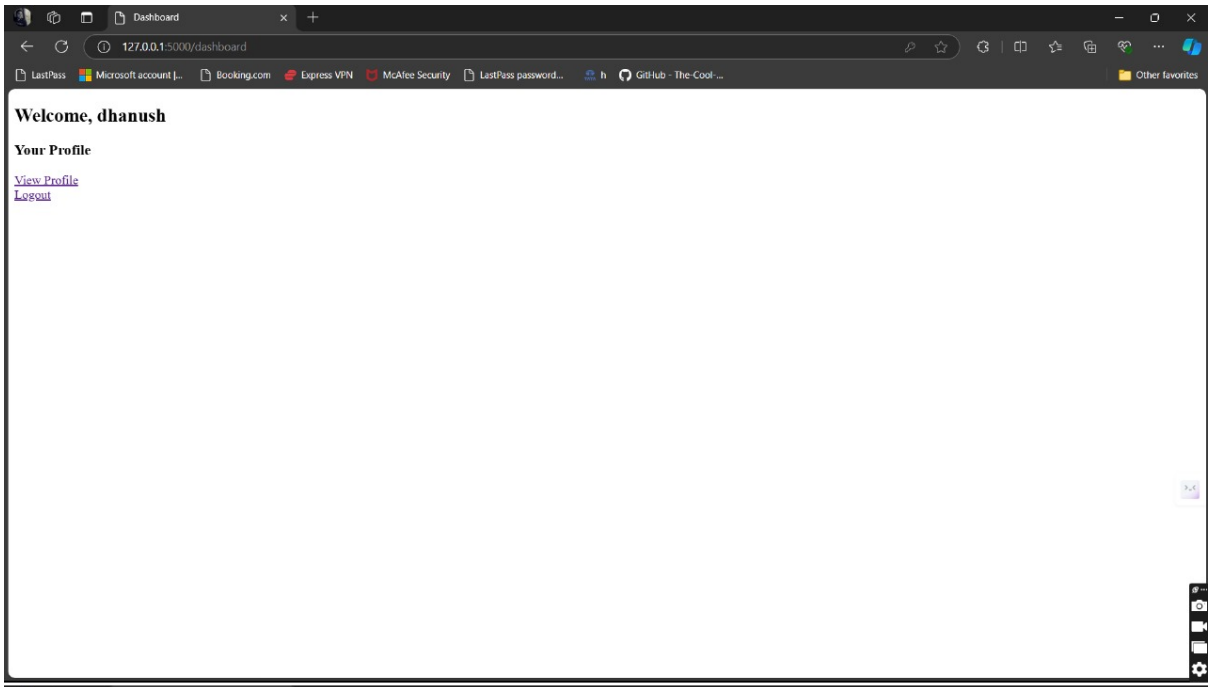
Running the Program

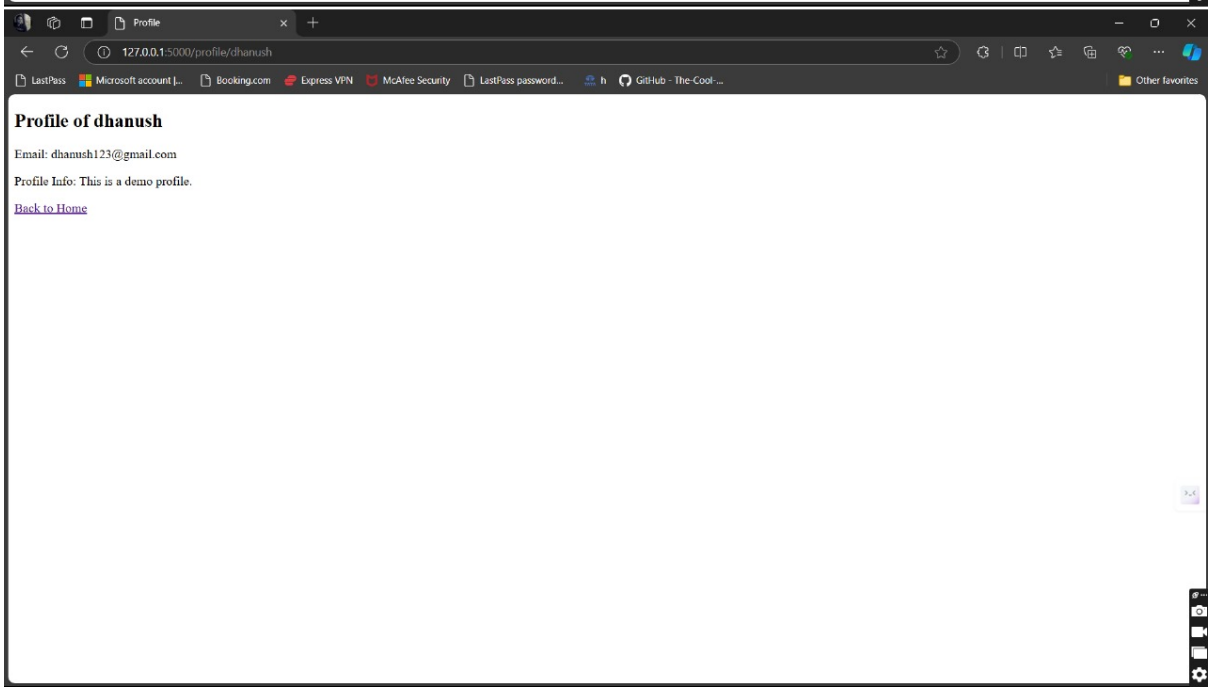
When you run this program, it will:

1. Create a social network with 5 users.
2. Simulate 10 random interactions between these users, with each interaction being either positive or negative.
3. Print the trust scores of all users after the interactions.


Register
127.0.0.1:5000/register
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
h
GitHub - The-Cool-...
Other favorites
Register
Username:
Password:
Email:
Register
Back to Login


Login
127.0.0.1:5000
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
h
GitHub - The-Cool-...
Other favorites
Login
Username:
Password:
Login
Register


Dashboard
127.0.0.1:5000/dashboard
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
h
GitHub - The-Cool-...
Other favorites
Welcome, dhanush
Your Profile
View Profile
Logout


Profile
127.0.0.1:5000/profile/dhanush
LastPass
Microsoft account |...
Booking.com
Express VPN
McAfee Security
LastPass password...
h
GitHub - The-Cool-...
Other favorites
Profile of dhanush
Email: dhanush123@gmail.com
Profile Info: This is a demo profile.
Back to Home

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus the implementation of a program in Python to estimate trust of the social network users group has been successfully executed.

Ex:9 Write a SQL injection program in python/JAVA to handle session hijacking

Aim: To write a SQL injection program in python/JAVA to handle session hijacking

Theory: Creating a SQL injection program for educational purposes requires a careful approach, especially since this topic involves vulnerabilities and security risks. Instead of providing a harmful example, I can guide you through a demonstration of how SQL injection occurs and how to prevent it, specifically focusing on session hijacking in web applications.

Procedure:

We'll create a simple web application using Flask in Python that demonstrates how SQL injection can occur and how to handle session management securely.

Step 1: **Setting Up the Environment**

Make sure you have Flask and SQLite installed. You can install them using pip:

pip install Flask

Step 2: **Simple Flask Application with Vulnerability**

Here's a basic example of a Flask application that is vulnerable to SQL injection. This example uses SQLite for simplicity.

app.py:

```
from flask import Flask, request, session, redirect, url_for, render_template
import sqlite3
app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Change this to a random secret key

# Initialize the database
def init_db():
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL,
```

```
            password TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()
# Function to check user credentials (vulnerable to SQL injection)
def check_user(username, password):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    # Vulnerable SQL query
    query = f"SELECT * FROM users WHERE username='{username}' AND
password='{password}'"
    cursor.execute(query)
    user = cursor.fetchone()
    conn.close()
    return user
@app.route('/')
def home():
    return render_template('login.html')
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    user = check_user(username, password)
    if user:
        session['user_id'] = user[0]  # Store user ID in session
        return redirect(url_for('dashboard'))
    return 'Login Failed'
@app.route('/dashboard')
def dashboard():
    if 'user_id' in session:
```

```
        return f'Welcome, User ID: {session["user_id"]}' return
    redirect(url_for('home'))
@app.route('/logout') def logout():
    session.pop('user_id', None)  # Remove user from session return
    redirect(url_for('home'))
if __name__ == '__main__': init_db()
    app.run(debug=True)
```

login.html:

```
<!doctype html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <form method="post" action="/login">
        Username: <input type="text" name="username"><br> Password: <input
        type="password" name="password"><br>
        <input type="submit" value="Login">
    </form>
</body>
</html>
```

Explanation of the Code

1. **Database Initialization**: The init_db function creates a simple SQLite database with a users table.
2. **Vulnerable Login Function**: The check_user function constructs a SQL query directly using user input, making it vulnerable to SQL injection.
3. **Session Management**: If the login is successful, the user ID is stored in the session, allowing for session hijacking if the application is compromised.
4. **Routes**: The application has routes for the home page, login processing, dashboard, and logout.

SQL Injection Example

To demonstrate SQL injection, you could try the following credentials when logging in:

- **Username**: admin' --
- **Password**: (leave this blank)

This input exploits the SQL query by commenting out the rest of the query, potentially allowing unauthorized access.

Prevention: Using Parameterized Queries

To prevent SQL injection, always use parameterized queries. Here's how you can modify the check_user function to be secure:

```
def check_user(username, password): conn =
    sqlite3.connect('users.db') cursor = conn.cursor()
    # Use parameterized query to prevent SQL injection
    cursor.execute("SELECT * FROM users WHERE username=? AND password=?", (username, password))
    user = cursor.fetchone() conn.close()
    return user
```

Conclusion

This example demonstrates how SQL injection can occur and emphasizes the importance of securing applications against such vulnerabilities. When handling sessions, always ensure you implement security best practices to prevent session hijacking, such as using HTTPS, secure session cookies, and proper session expiration policies.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, an SQL injection program in Python to handle session hijacking has been successfully executed.

Ex:10 Create an application using any social network platform to demonstrate the profile cloning concept

Aim: To create an application using any social network platform to demonstrate the profile cloning concept.

Theory: Creating an application that demonstrates the concept of profile cloning on a social network can be quite sensitive, especially considering ethical implications. Instead, I can guide you through developing an educational example that illustrates how profile cloning might occur in a hypothetical scenario. This will focus on understanding the mechanics without promoting any malicious behavior.

Procedure: We will create a simple web application using Flask that simulates user profiles and demonstrates how a malicious user might attempt to clone a profile. This example will include user registration, login, and an interface to view profiles, highlighting how easy it could be to impersonate someone if security measures aren't in place.

Step 1: Setting Up the Environment

Ensure you have Flask and SQLite installed. You can install them using pip:

pip install Flask

Step 2: **Creating the Flask Application**

Directory Structure:

profile_cloning/

```
├── app.py
├── templates/
│   ├── login.html
│   ├── register.html
│   ├── profile.html
│   └── dashboard.html
```

Step 3: app.py:

```
from flask import Flask, request, session, redirect, url_for, render_template, flash
import sqlite3
```

```
app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Change this to a random secret key

# Initialize the database
def init_db():
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL,
            email TEXT NOT NULL UNIQUE,
            profile_info TEXT
        )
    ''')
    conn.commit()
    conn.close()

# Function to register a new user
def register_user(username, password, email):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    try:
        cursor.execute("INSERT INTO users (username, password, email, profile_info)
VALUES (?, ?, ?, ?)",
                       (username, password, email, "This is a demo profile."))
        conn.commit()
    except sqlite3.IntegrityError:
        return False
    finally:
```

```
        conn.close()
    return True

# Function to check user credentials
def check_user(username, password):
   conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
   cursor.execute("SELECT * FROM users WHERE username=? AND password=?",
(username, password))
    user = cursor.fetchone()
    conn.close()
    return user

# Function to get user profile information
def get_user_profile(username):
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users WHERE username=?", (username,))
    profile = cursor.fetchone()
    conn.close()
    return profile

@app.route('/')
def home():
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
```

```
        email = request.form['email']
        if register_user(username, password, email):
            flash('User registered successfully! Please log in.', 'success')
            return redirect(url_for('home'))
        else:
            flash('Username or email already exists.', 'danger')
    return render_template('register.html')

@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    user = check_user(username, password)

    if user:
        session['user_id'] = user[0]  # Store user ID in session
        session['username'] = user[1]  # Store username in session
        return redirect(url_for('dashboard'))
    flash('Login Failed. Check your credentials.', 'danger')
    return redirect(url_for('home'))

@app.route('/dashboard')
def dashboard():
    if 'user_id' in session:
        return render_template('dashboard.html', username=session['username'])
    return redirect(url_for('home'))

@app.route('/profile/<username>')
def profile(username):
    profile = get_user_profile(username)
```

```
    if profile:
        return render_template('profile.html', profile=profile)
    return 'Profile not found', 404

@app.route('/logout')
def logout():
    session.pop('user_id', None)  # Remove user from session
    session.pop('username', None)
    return redirect(url_for('home'))

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

templates/login.html:

```
<!doctype html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h2>Login</h2>
    <form method="post" action="/login">
        Username: <input type="text" name="username"><br>
        Password: <input type="password" name="password"><br>
        <input type="submit" value="Login">
    </form>
    <a href="/register">Register</a>
</body>
</html>
```

templates/register.html:

```
<!doctype html>
<html>
<head>
   <title>Register</title>
</head>
<body>
   <h2>Register</h2>
   <form method="post" action="/register">
      Username: <input type="text" name="username"><br>
      Password: <input type="password" name="password"><br>
      Email: <input type="email" name="email"><br>
      <input type="submit" value="Register">
   </form>
   <a href="/">Back to Login</a>
</body>
</html>
```

templates/dashboard.html:

```
<!doctype html>
<html>
<head>
   <title>Dashboard</title>
</head>
<body>
   <h2>Welcome, {{ username }}</h2>
   <h3>Your Profile</h3>
   <a href="/profile/{{ username }}">View Profile</a><br>
   <a href="/logout">Logout</a>
</body>
</html>
```

templates/profile.html:

```
<!doctype html>
<html>
<head>
   <title>Profile</title>
</head>
<body>
   <h2>Profile of {{ profile[1] }}</h2>
   <p>Email: {{ profile[3] }}</p>
   <p>Profile Info: {{ profile[4] }}</p>
   <a href="/">Back to Home</a>
</body>
</html>
```

Explanation of the Code

1. **Database Initialization**: The init_db function sets up a SQLite database to store user information, including usernames, passwords, emails, and profile information.
2. **User Registration**: The register_user function allows new users to create an account. It checks for uniqueness of usernames and emails to avoid duplicates.
3. **User Login**: The check_user function validates user credentials during login.
4. **Profile Viewing**: The get_user_profile function retrieves profile information based on the username. Users can view their own profiles and other users' profiles.
5. **Routes**: The application includes routes for home (login), registration, dashboard, viewing profiles, and logging out.

Demonstration of Profile Cloning

1. **User Registration**: User A registers with their username and password.
2. **Profile Cloning Simulation**: A malicious user (User B) can attempt to clone User A's profile by registering with a similar username or accessing User A's profile directly.
3. **Security Awareness**: Highlight the importance of unique usernames, passwords, and security practices to prevent profile cloning and impersonation.

Running the Application

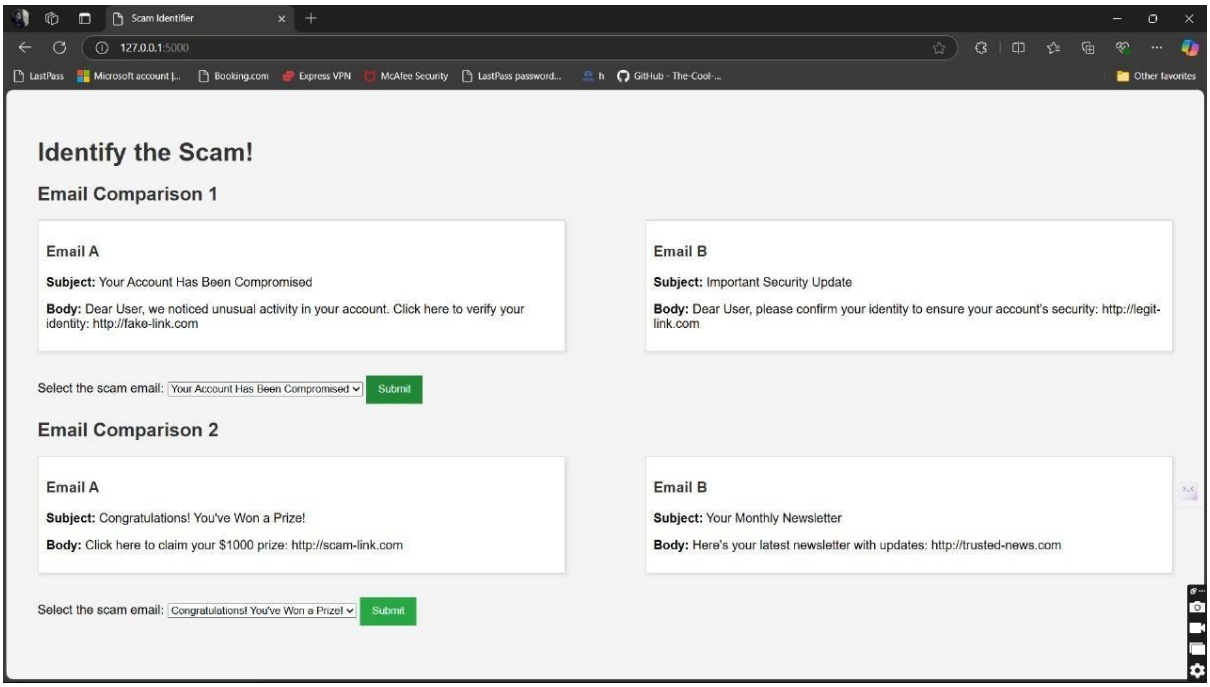
1. Save the above files in a directory called profile_cloning.
2. Run the application using the command:

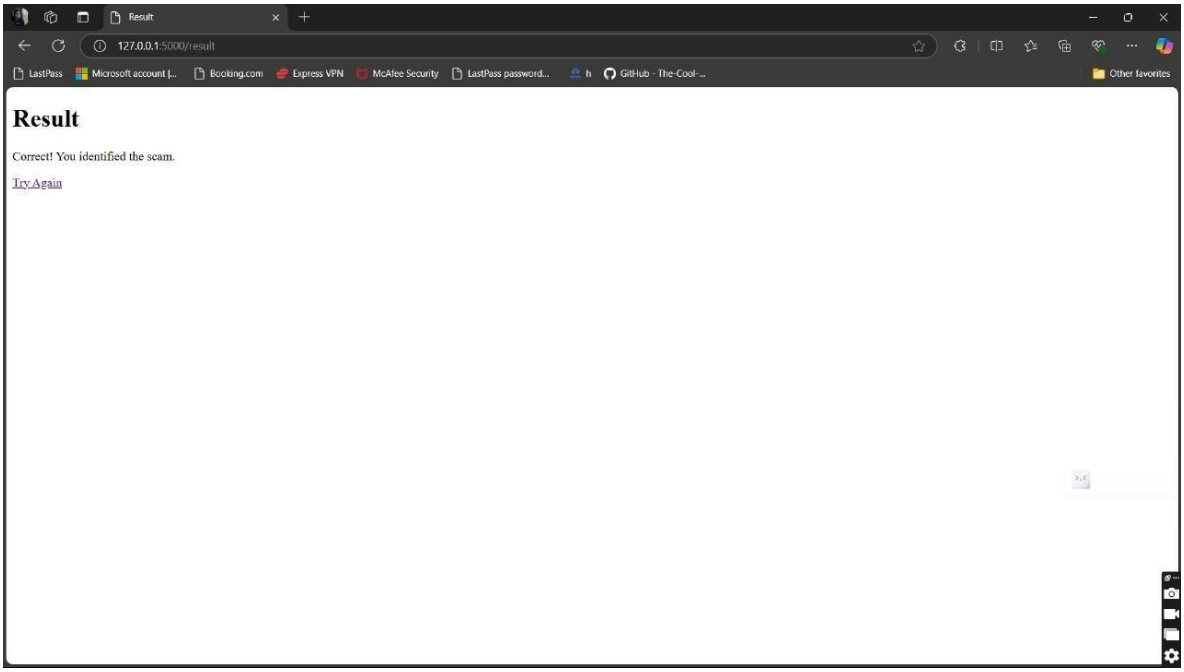
python app.py

Access the application in your web browser at http://127.0.0.1:5000/.

Conclusion

This application simulates the concept of profile cloning in a social network context, highlighting how easily profiles could be accessed and cloned if proper security measures are not in place. It serves as an educational tool to demonstrate the need for secure user management and awareness of potential impersonation risks.





Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus the queries to handle exception using procedure are executed and verified.

Ex:11 Perform fact-checking of social networking content using Google fact-checking tools

Aim: To perform fact-checking of social networking content using Google fact-checking tools

Theory: Fact-checking social networking content is essential to combat misinformation and ensure the reliability of information shared online. Google provides several tools and resources that can help you verify facts and validate content from social media platforms. Below are steps and methods for effectively using Google's fact-checking tools.

Procedure:

1. Google Fact Check Explorer

Google Fact Check Explorer is a tool that allows users to search for fact-checks that have been published by various organizations.

- **How to Use:**
    - Visit the Google Fact Check Explorer.
    - Enter keywords or phrases related to the content you want to verify.
    - Browse through the results to find relevant fact-checks from credible organizations.

2. Google Search

Using Google Search is another effective way to verify claims or information.

- **How to Use:**
    - Input the claim or specific information you want to fact-check in Google Search.
    - Look for results from reputable news sources, fact-checking organizations, or official statements.
    - Check the publication date to ensure the information is current.
    - Use search operators like "site:" to limit your search to specific domains (e.g., site:bbc.com).

3. Google Reverse Image Search

When verifying images that are circulated on social media, Google Reverse Image Search can help identify the source and context.

- **How to Use:**
    - Go to Google Images.

- Click on the camera icon to upload an image or paste an image URL.
- Review the results to see where else the image has appeared and whether it has been taken out of context.

4. Google News

Google News aggregates news stories from various sources, which can be useful for cross-referencing claims.

- **How to Use:**
  - Go to Google News.
  - Search for relevant topics or keywords.
  - Filter results by date to find the most recent and relevant articles.

5. Using Fact-Checking Websites

In addition to Google's tools, several dedicated fact-checking websites can provide valuable information:

- **Snopes**: A widely recognized resource for debunking rumors and myths.
- **FactCheck.org**: A project of the Annenberg Public Policy Center that monitors the factual accuracy of statements by political figures.
- **PolitiFact**: Focuses on political claims and provides a "Truth-O-Meter" rating.

6. Checking Social Media Content

When you encounter questionable content on social media, you can take the following steps:

- **Look for Citations**: Check if the post includes citations or links to reputable sources.
- **Investigate the Source**: Click on the user's profile to see their credibility and history of sharing information.
- **Use Hashtag Searches**: On platforms like Twitter, searching for specific hashtags related to the claim can help find discussions or fact-checks about it.

Example Scenario

Suppose you see a claim on social media that says, "Eating carrots improves your eyesight significantly."

1. **Google Fact Check Explorer**: Search for "carrots improve eyesight." Check if reputable organizations have published any fact-checks.
2. **Google Search**: Search for "carrots eyesight research." Look for studies from credible sources like medical journals or government health websites.
3. **Reverse Image Search**: If the claim is accompanied by an image (e.g., a graphic showing the benefits of carrots), use reverse image search to ensure it's not misleading or taken out of context.

4. **Cross-Referencing**: Look for articles or blog posts from credible health organizations to corroborate or debunk the claim.

Conclusion

Using Google's tools and other reliable resources can significantly enhance your ability to fact-check social networking content. Always cross-reference information and consider the credibility of sources before accepting or sharing claims.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, fact-checking of social networking content using Google fact-checking tools has been successfully executed

Ex:12 Explore a tool that helps protect websites from bot traffic and bot attacks.

Aim: To explore a tool that helps protect websites from bot traffic and bot attacks.

Theory: Protecting websites from bot traffic and attacks is crucial for maintaining security, performance, and the integrity of online services. There are several tools available that can help mitigate these risks by identifying and blocking malicious bot traffic.

Procedure:

1. **Cloudflare**

Overview: Cloudflare is a popular content delivery network (CDN) that provides various security features, including bot management.

- **Key Features**:
    - **Bot Management**: Identifies and mitigates malicious bots while allowing legitimate traffic.
    - **Web Application Firewall (WAF)**: Filters and monitors HTTP traffic between a web application and the Internet.
    - **Rate Limiting**: Limits the number of requests a user can make to the server in a given period.
    - **Bot Fight Mode**: Automatically challenges suspicious traffic with CAPTCHA or JavaScript challenges.
- **Usage**: Sign up for Cloudflare, configure your DNS settings, and enable bot management features through the dashboard.

Website: Cloudflare

2. Akamai Bot Manager

Overview: Akamai offers advanced bot management solutions designed to protect websites from bot attacks while ensuring legitimate traffic can access the site without friction.

- **Key Features**:
    - **Advanced Detection**: Uses machine learning to differentiate between good and bad bot traffic.
    - **Real-Time Analytics**: Provides insights into bot traffic patterns and threats.
    - **Custom Rules**: Allows you to set rules to block or challenge certain types of bot traffic.

- **Usage**: Integrate Akamai's solution with your web application and configure settings via their management console.

Website: Akamai Bot Manager

3. Imperva (Formerly Incapsula)

Overview: Imperva provides a comprehensive security solution that includes protection against bot attacks.

- **Key Features**:
    - **Bot Detection**: Identifies and blocks malicious bots using advanced algorithms.
    - **Traffic Analysis**: Monitors incoming traffic for suspicious activity.
    - **CAPTCHA Challenges**: Presents challenges to suspicious users to verify their identity.
- **Usage**: Sign up for Imperva services, implement the provided security measures, and monitor traffic via their dashboard.

Website: Imperva

4. Distil Networks (now part of Imperva)

Overview: Distil Networks specializes in bot detection and mitigation, focusing on protecting websites from various types of automated threats.

- **Key Features**:
    - **Real-Time Monitoring**: Continuously monitors traffic for bot activity.
    - **Behavioral Analysis**: Analyzes user behavior to identify bot-like patterns.
    - **Blocking and Mitigation**: Provides options to block or challenge suspicious traffic.
- **Usage**: Implement Distil's services on your site and configure settings to tailor the protection to your specific needs.

Website: Distil Networks

5. Bot Sentinel

Overview: Bot Sentinel is a tool designed to detect and analyze bot accounts, particularly on Twitter, but can also be useful for identifying suspicious behavior on websites.

- **Key Features**:
    - **Bot Detection**: Identifies and tracks bot accounts based on behavior.

- **Analytics**: Provides insights into bot activity and patterns.

- **Usage**: You can use the Bot Sentinel web app to input account information and analyze it for potential bot activity.

  Website: Bot Sentinel

6. Google reCAPTCHA

Overview: Google reCAPTCHA helps to differentiate between human users and bots on your website.

- **Key Features**:
  - **CAPTCHA Challenges**: Presents puzzles or challenges to verify users are human.
  - **Invisible reCAPTCHA**: Automatically analyzes user behavior in the background without user interaction.
- **Usage**: Integrate reCAPTCHA into your forms and pages. You can customize the difficulty of challenges based on your needs.

Website: Google reCAPTCHA

Conclusion

Using these tools, you can effectively protect your website from bot traffic and attacks. Implementing a combination of these solutions will enhance your security posture, helping you to block malicious bots while ensuring a seamless experience for legitimate users.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, the Exploration tool helps protect websites from bot traffic and bot attacks.

Ex:13 Create a fake news tracker program to collect, detect, and help visualize fake news data from any social network

Aim: To create a fake news tracker program to collect, detect, and help visualize fake news data from any social network

Theory: Creating a fake news tracker program involves multiple components, including data collection, detection of fake news, and visualization of the collected data.

Procedure:

Requirements

You'll need to install the following libraries:

Pip install tweepy pandas matplotlib seaborn sci-kit-learn

Step 1: Set Up Twitter API Access

To access Twitter data, you need to create a Twitter Developer account and create an application to obtain your API keys.

Step 2: Data Collection

Here's how to collect tweets using Tweepy:

data_collector.py:

```
import tweepy

import pandas as pd

import os

# Twitter API credentials

API_KEY = 'YOUR_API_KEY'

API_SECRET = 'YOUR_API_SECRET'

ACCESS_TOKEN = 'YOUR_ACCESS_TOKEN'

ACCESS_TOKEN_SECRET = 'YOUR_ACCESS_TOKEN_SECRET'

# Authenticate to Twitter

auth = tweepy.OAuth1UserHandler(API_KEY, API_SECRET, ACCESS_TOKEN,
ACCESS_TOKEN_SECRET)

api = tweepy.API(auth)
```

```
# Function to collect tweets
def collect_tweets(query, count=100):
    tweets = tweepy.Cursor(api.search_tweets, q=query, lang="en").items(count)
    tweet_data = [{'tweet': tweet.text, 'created_at': tweet.created_at} for tweet in tweets]
    return pd.DataFrame(tweet_data)

# Example usage
if _name_== "_main_":
    query = "#news"
    tweets_df = collect_tweets(query, count=200)
    tweets_df.to_csv('tweets.csv', index=False)
    print("Tweets collected and saved to tweets.csv.")
```

Step 3: Fake News Detection

We can create a simple rule-based detection for this example. A more sophisticated approach would involve using a machine learning model trained on labeled datasets.

fake_news_detector.py:

```
import pandas as pd

# Simple rule-based fake news detection function
def is_fake_news(tweet):
    fake_keywords = ['fake', 'scam', 'hoax', 'lie', 'deceptive']
    return any(keyword in tweet.lower() for keyword in fake_keywords)

# Function to detect fake news in the collected tweets
def detect_fake_news(tweets_df):
    tweets_df['is_fake'] = tweets_df['tweet'].apply(is_fake_news)
    return tweets_df
```



```
# Example usage
```

```
if __name__ == "__main__":
    tweets_df = pd.read_csv('tweets.csv')
    results_df = detect_fake_news(tweets_df)
    results_df.to_csv('tweets_with_detection.csv',  index=False)
    print("Fake news detection complete and results saved to tweets_with_detection.csv.")
```

Step 4: Visualization

Finally, we can visualize the results using Matplotlib and Seaborn.

visualization.py:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Function to visualize fake news detection results
def visualize_results(results_df):
    plt.figure(figsize=(8, 6))
    sns.countplot(x='is_fake', data=results_df)
    plt.title('Fake News Detection Results')
    plt.xlabel('Is Fake News?')
    plt.ylabel('Count')
    plt.xticks([0, 1], ['No', 'Yes'])
    plt.show()

# Example usage
if __name__ == "__main__":
    results_df = pd.read_csv('tweets_with_detection.csv')
    visualize_results(results_df)
```

Running the Program

1. **Collect Tweets**: Run data_collector.py to collect tweets containing a specific hashtag (e.g., #news). This will save the tweets to tweets.csv.

   ```
   python data_collector.py
   ```

2. **Detect Fake News**: Run fake_news_detector.py to analyze the collected tweets for fake news.
   python fake_news_detector.py

3. **Visualize Results**: Finally, run visualization.py to visualize the results of the fake news detection.
   python visualization.py

Conclusion

This simple fake news tracker program collects tweets, detects potentially fake news using a rule-based approach, and visualizes the results. For a production-level application, consider integrating more advanced machine learning models trained on comprehensive datasets, enhancing the detection accuracy.

Problem understanding and Design (3 marks)	
Procedure Implementation (3 marks)	
Output & Viva Explanation (4 marks)	
Total (10 marks)	

Result:

Thus, creating a fake news tracker program involves multiple components, including data collection, detection of fake news, and visualization of the collected data has been successfully executed.