1. To explore the Confidentiality identify potential security violations through practical examples and system analysis.

1. Create a File Containing Sensitive Information
* Open a terminal or command prompt.
* Create a text file named sensitive_file.txt:
Linux: echo "Confidential Data: Usernames and Passwords" > sensitive_file.txtIS Practical1
Windows: Open Notepad, type "Confidential Data: Usernames and Passwords", and save
the file as sensitive_file.txt.

2. Restrict File Permissions
* Set file permissions so only the owner can access it:
Linux: Run chmod 600 sensitive_file.txt
Windows: Right-click the file Properties Security Edit permissions → Deny access for all users except the owner.

3. Simulate Unauthorized Access
* Switch to another user or simulate unauthorized access:
Linux: Use su or create a new user, then try accessing the file:
cat sensitive_file.txt
To create new user, use this command: sudo adduser random_intruder
Windows: Switch user accounts or create a new user, then try opening the file.
* Observe the error message (e.g., "Permission denied").

2. To explore the Integrity identify potential security violations through practical examples and system analysis.

1. Create or Access a Log File
* Use an existing log file or create a simulated one:
Linux: sudo nano /var/log/syslog (requires root access).
Windows: Open Event Viewer (eventvwr) or create a text file named logfile.

2. Modify the Log File Simulate Unauthorized Changes)
* Add or change log entries to simulate a security violation:
Linux: Edit the file: sudo nano /var/log/syslog
Add a fake entry:
“Jan 1 12:00:00 UnauthorizedAccess: Admin login”
Add in any Random Line
Ctrl+x
Shift+Y
Enter
Windows: Open logfile.txt in Notepad and add UnauthorizedAccess: Admin login.

3. Verify Integrity with Hashing
* Calculate the file's hash before and after modification
Linux: Use sha256sum logfile.txt and note the hash
Windows: Use PowerShell: Get-FileHash .\logfile.txt Algorithm SHA256
* Observe the hash difference.

IF “NO SUCH DIRECTORY ERROR”:
echo "Test log entry" > logfile.txt
Then run this again:
sha256sum logfile.txt

3. To explore Availability and identify potential security violations through practical examples and system analysis:

1. Set Up a Simple Web Server (Optional)

Linux:
- Open a terminal and run:
  ```bash
  python3 -m http.server 8080
  ```

Windows:
- Set up a server using IIS or WAMP/XAMPP.

2. Simulate Denial-of-Service (DoS) Attack

Linux:
1. Install Apache Benchmark (ab tool):
   ```bash
   sudo apt update
   sudo apt install apache2-utils
   ```
2. Run the benchmark tool in a new terminal window:
   ```bash
   ab -n 1000 -c 100 http://localhost:8080/
   ```

Windows:
- Use PowerShell or JMeter to simulate the attack.

3. Monitor Server Behavior

Linux:
- Check the server logs or terminal for delays, errors, and timeouts.

Windows:
- Use Task Manager or Resource Monitor to observe CPU and network usage.

4. Restore Normal Operations

Linux:
- Terminate the attack using Ctrl+C and test the server response with:
  ```bash
  curl http://localhost:8080
  ```

Windows:
- Stop or restart the server via IIS Manager, then verify server availability.

4. To explore the Confidentiality and Availability and identify potential security violations through practical examples and system analysis.

Perform Experiments 1 and 3 from above.

5. To Configure and Understand Discretionary Access Control (DAC) Mechanisms in a Linux Environment

1. Discretionary Access Control (DAC)

a) Create Files and Directories:
1. Open a terminal and create a directory for testing DAC mechanisms:
   mkdir dac_demo && cd dac_demo
2. Create a file named confidential.txt within the directory:
   touch confidential.txt

b) Set Permissions Using chmod:
1. Use chmod to set permissions for the file. For example, restrict read and write access to the file for the owner only:
   chmod 600 confidential.txt
   - This makes the file readable and writable only by the file's owner.

For WSL (Windows Subsystem for Linux), you can run:
   sudo chmod o-rx /home/username/dac_demo

c) Change File Ownership Using chown:
1. Add a new user (e.g., alice) and change the ownership of the file to that user:
   sudo adduser alice
   sudo chown alice:alice confidential.txt

d) Log in as Alice:
1. Switch to the user alice and try to read the confidential.txt file:
   su alice
   cat confidential.txt

For WSL, if su does not work, use:
  sudo -i -u alice
  sudo su - alice

2. Verify that other users (such as the default user) cannot access the file:
  ls /home/atharv/dac_demo
Output:
  ls: cannot access '/home/username/dac_demo': Permission denied

e) Results:
- Alice (the file owner) can access confidential.txt.
- Other users (e.g., the default user) cannot read or write to the file due to the restrictive permissions (600).

6. To Configure and Understand Mandatory Access Control (MAC) Mechanisms in a Linux Environment

1. Mandatory Access Control (MAC)

a) Create Files and Directories:
1. Open a terminal and create a directory for testing MAC mechanisms:
   mkdir mac_demo && cd mac_demo
2. Create a file named confidential.txt:
   touch confidential.txt

b) Enable SELinux:
1. Check if SELinux is enabled by running:
   getenforce
2. If SELinux is not enabled, you can enable it by running:
   sudo setenforce 1

c) Apply Security Context to a File:
1. Verify the security context of the file using the ls -Z command:
   ls -Z confidential.txt

d) Test Policy Enforcement:
1. Simulate unauthorized access by trying to access the file through a user or process that lacks the required permissions. For example, you might attempt to cat the file as a non-root user:
   sudo cat confidential.txt
2. Check the SELinux logs to verify that the event was logged:
   sudo cat /var/log/audit/audit.log

e) Results:

- SELinux will deny access to unauthorized processes or users and will log the event in the audit.log.
- Unauthorized processes or users will be denied access even if they are the file owner.

7. To Configure and Understand Role-Based Access Control (RBAC) Mechanisms in a Linux Environment

1. Role-Based Access Control (RBAC)

a) Create Roles (Groups):

First, create a new group called `managers`:

```
sudo groupadd managers
```

1.

Then, add the user `alice` to the `managers` group:

```
sudo usermod -aG managers alice
```

2.

b) Set Permissions on a File:

Create a file named `manager_notes.txt`:

```
touch manager_notes.txt
```

1.

Change the file's group to `managers`:

```
sudo chown :managers manager_notes.txt
```

2.

Set the permissions on the file so that only the owner and group members can read and write to it:

sudo chmod 770 manager_notes.txt

3.

c) Test Access:

1. **As Alice:**

Switch to the `alice` user:

su alice

   ○

After testing access, exit from the `alice` user and return to the main user:

exit

   ○

2. **To Add a New User:**

Add a new user (e.g., `other_user`):

sudo adduser <username>

   ○

3. **As Another User:**

Switch to the newly created user:

su <other_user/username>

-

Try to access `manager_notes.txt`:

```
cat manager_notes.txt
```

-

d) Results:

- Access to `manager_notes.txt` is controlled by group membership (role), not individual ownership. Only users in the `managers` group will be able to read and write to the file.

8. Generation of Public/Private Key Pairs, Creation of a User Certificate, and Digital Signing Using a Certificate Authority

1. Create a directory for keys and navigate to it:

mkdir key && cd key

2. Generate a private key (RSA algorithm) with AES256 encryption:

openssl genpkey -algorithm RSA -out private.key -aes256

- You will be prompted to **Enter a PEM pass phrase**. Enter a passphrase and remember it.

3. Generate the public key from the private key:

openssl rsa -pubout -in private.key -out public.key

- Enter the pass phrase for the private key when prompted.

4. Create a Certificate Signing Request (CSR) using the private key:

openssl req -new -key private.key -out user.csr

- Enter the pass phrase for the private key.

- Fill out the **Distinguished Name (DN)** fields when prompted.

5. Create a self-signed certificate using the private key and CSR:

openssl req -x509 -key private.key -in user.csr -out user_cert.crt -days 365

- Enter the pass phrase for the private key.
- A warning may appear—ignore it.

6. Generate the Certificate Authority (CA) private key (with AES256 encryption):

openssl genpkey -algorithm RSA -out ca.key -aes256

- Enter a **PEM pass phrase** for the CA key and verify it.

7. Create the CA self-signed certificate:

openssl req -x509 -key ca.key -out ca.crt -days 3650

- Enter the pass phrase for the CA key when prompted.
- Fill out the **Distinguished Name (DN)** fields when prompted.

8. Generation of Public/Private Key Pairs, Creation of a User Certificate, and Digital Signing Using a Certificate Authority

1. Create a directory for keys and navigate to it:

- mkdir key && cd key

2. Generate a private key (RSA algorithm) with AES256 encryption:

- openssl genpkey -algorithm RSA -out private.key -aes256

- You will be prompted to **Enter a PEM pass phrase**. Enter a passphrase and remember it.

3. Generate the public key from the private key:

- openssl rsa -pubout -in private.key -out public.key

- Enter the pass phrase for the private key when prompted.

4. Create a Certificate Signing Request (CSR) using the private key:

- openssl req -new -key private.key -out user.csr

- Enter the pass phrase for the private key.

- Fill out the **Distinguished Name (DN)** fields when prompted.

5. Create a self-signed certificate using the private key and CSR:

- openssl req -x509 -key private.key -in user.csr -out user_cert.crt -days 365

- Enter the pass phrase for the private key.
- A warning may appear—ignore it.

6. Generate the Certificate Authority (CA) private key (with AES256 encryption):

- openssl genpkey -algorithm RSA -out ca.key -aes256

- Enter a **PEM pass phrase** for the CA key and verify it.

7. Create the CA self-signed certificate:

- openssl req -x509 -key ca.key -out ca.crt -days 3650

- Enter the pass phrase for the CA key when prompted.
- Fill out the **Distinguished Name (DN)** fields when prompted.

9. Perform Scan for Open Ports Using Nmap

1. Update Ubuntu and Install Required Packages:

sudo apt update && sudo apt upgrade -y

sudo apt install nmap -y

2. Scan for Open Ports Using Nmap:

- First, check for your system's IP address:

ip a

- Then, perform the port scan on the target IP (in this case, `10.10.10.6`):

nmap -sV -p- 10.10.10.6 # Replace with your target IP address (e.g., eth0)

- The `-sV` option attempts to determine the version of the services running on open ports.
- The `-p-` option tells Nmap to scan all ports.
- **NOTE**: REMOVE /24 FROM IP ADDRESS, /24 MEANS ALL DEVICES.

10. Perform AES Encryption & Decryption

1. Update Ubuntu and Install Required Packages:

sudo apt update && sudo apt upgrade -y

sudo apt install python3 python3-pip -y

pip3 install cryptography pandas faker

python3 --version

pip3 list | grep -E "cryptography|pandas|faker"

2. AES Encryption & Decryption

- **AES (Advanced Encryption Standard)** is a symmetric encryption algorithm.

a. Generate an AES Key:

- Create a Python script `aes_key.py` to generate the AES key:
  Terminal: touch aes_key.py && nano aes_key.py

```
from cryptography.fernet import Fernet


# Generate AES key

key = Fernet.generate_key()


# Save the key in a file
```

```
with open("aes_key.key", "wb") as key_file:
    key_file.write(key)

print(f"Generated AES Key: {key.decode()}")
```

- Run the script:

```
python3 aes_key.py
```

This will generate a key and save it in the `aes_key.key` file.

b. Encrypt a Message:

- Create a Python script `aes_encrypt.py` to encrypt a message:
  Terminal: touch aes_encrypt.py && nano aes_encrypt.py

```
from cryptography.fernet import Fernet

# Load AES key
key = open("aes_key.key", "rb").read()
cipher = Fernet(key)

# Message to encrypt
message = "Confidential Data: Do not share!"
```

Encrypt the message

encrypted_message = cipher.encrypt(message.encode())

print(f"Encrypted Message: {encrypted_message.decode()}")

- Run the script:

python3 aes_encrypt.py

This will encrypt the message and print the encrypted result.

c. Decrypt the Message:

- Create a Python script `aes_decrypt.py` to decrypt the message:
  Terminal: touch aes_decrypt.py && nano aes_decrypt.py

from cryptography.fernet import Fernet

Load AES key

key = open("aes_key.key", "rb").read()

cipher = Fernet(key)

Encrypted message (replace with actual encrypted message)

encrypted_message = b”ENCRYPTED_MESSAGE_HERE” # Replace with your encrypted message

Decrypt the message

decrypted_message = cipher.decrypt(encrypted_message).decode()

print(f"Decrypted Message: {decrypted_message}")

- Replace `ENCRYPTED_MESSAGE_HERE` with the actual encrypted message you got from `aes_encrypt.py`.
- Run the script:

python3 aes_decrypt.py

This will decrypt the message and print the original message.

NOTE: If indent error in aes_decrypt.py: sed -i '1s/^ //' aes_decrypt.py

And check with: cat -A aes_decrypt.py

11. Perform RSA Encryption & Decryption

1. Update Ubuntu and Install Required Packages:

sudo apt update && sudo apt upgrade -y

sudo apt install python3 python3-pip -y

pip3 install cryptography pandas faker

python3 --version

pip3 list | grep -E "cryptography|pandas|faker"

2. RSA Encryption & Decryption

a. Generate RSA Key Pair:

Create a Python script `rsa_key.py` to generate the RSA key pair:
Terminal: touch rsa_key.py && nano rsa_key.py

```
from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.hazmat.primitives import serialization


# Generate private key

private_key = rsa.generate_private_key(

    public_exponent=65537,

    key_size=2048

)
```

```
# Save private key
with open("rsa_private.pem", "wb") as f:
    f.write(private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.TraditionalOpenSSL,
        encryption_algorithm=serialization.NoEncryption()
    ))

# Generate public key
public_key = private_key.public_key()

# Save public key
with open("rsa_public.pem", "wb") as f:
    f.write(public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ))

print("RSA key pair generated and saved.")
```

- Run the script:

python3 rsa_key.py

This will generate a private key `rsa_private.pem` and a public key `rsa_public.pem`.

b. Encrypt Data Using RSA Public Key:

Create a Python script `rsa_encrypt.py` to encrypt the message:
Terminal: touch rsa_encrypt.py && nano rsa_encrypt.py

```
from cryptography.hazmat.primitives.asymmetric import padding

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives import serialization


# Load the public key

with open("rsa_public.pem", "rb") as f:

    public_key = serialization.load_pem_public_key(f.read())


# The message to encrypt

message = b"Secure Data Transfer"


# Encrypt the message using RSA public key
```

```
encrypted = public_key.encrypt(
        message,
        padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),  # <-- ()
        algorithm=hashes.SHA256(),                    # <-- ()
        label=None
        )
)

print(f"Encrypted Data: {encrypted}")
```

- Run the script:

```
python3 rsa_encrypt.py
```

This will encrypt the message and print the encrypted result.

c. Decrypt Data Using RSA Private Key:

Create a Python script `rsa_decrypt.py` to decrypt the message:
Terminal: touch rsa_decrypt.py && nano rsa_decrypt.py

```
from cryptography.hazmat.primitives.asymmetric import padding

from cryptography.hazmat.primitives import hashes
```

```
from cryptography.hazmat.primitives import serialization


# Load the private key

with open("rsa_private.pem", "rb") as f:

    private_key = serialization.load_pem_private_key(f.read(),
password=None)


# The encrypted message (replace with the result from
rsa_encrypt.py)

encrypted_message = b”ENCRYPTED_MESSAGE_HERE”  #
Replace with your encrypted message


# Decrypt the message using RSA private key

decrypted = private_key.decrypt(

        encrypted_message,

        padding.OAEP(

        mgf=padding.MGF1(algorithm=hashes.SHA256()),  # <-- ADD ()
!!!

        algorithm=hashes.SHA256(),                    # <-- ADD () !!!

        label=None

        )

)
```

print(f"Decrypted Data: {decrypted.decode()}")

- Replace `ENCRYPTED_MESSAGE_HERE` with the actual encrypted message you got from `rsa_encrypt.py`.
- Run the script:

python3 rsa_decrypt.py

This will decrypt the message and print the original message.

12. Perform Data Anonymization

1. Update Ubuntu and Install Required Packages:

sudo apt update && sudo apt upgrade -y

sudo apt install python3 python3-pip -y

pip3 install cryptography pandas faker

python3 --version

pip3 list | grep -E "cryptography|pandas|faker"

Install Faker Library:

sudo apt install faker

2. Masking Sensitive Data (Anonymization)

a. Create a script to mask sensitive data, such as SSNs:

Create a Python script `mask.py` to mask the SSN data:
Terminal: touch mask.py && nano mask.py

```
import pandas as pd

# Data containing SSNs
data = {"SSN": ["123-45-6789", "987-65-4321", "555-44-3333"]}
df = pd.DataFrame(data)
```

Masking SSN (showing only the last 4 digits)

df["Masked_SSN"] = df["SSN"].str.replace(r"\d{3}-\d{2}", "***-**-", regex=True)

Display the masked data

print(df)

- Run the script:

python3 mask.py

This will replace the first five digits of the SSN with `***-**-`, effectively masking the sensitive information.

NOTE: if pandas(I'll fucking kms) not working, put in terminal: sudo apt install python3-pandas

3. Generating Fake Data Using Faker:

Create another script `generate_fake_data.py` to generate fake data:
Terminal: touch generate_fake_data.py && nano generate_fake_data.py

from faker import Faker

```
# Create a Faker instance
fake = Faker()


# Generate and print 5 fake records
for _ in range(5):
    print(fake.name(), "-", fake.email(), "-", fake.phone_number())
```

- Run the script:

```
python3 generate_fake_data.py
```

This will generate 5 fake records with random names, email addresses, and phone numbers.