IS Practical

Note ~

> Some experiments require a Linux VM or Linux machine only.

> If Linux is not present, install `WSL` (Windows Subsystem for Linux) from the Microsoft Store and proceed with the experiment.

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.txt

Windows: Open Notepad, type "Confidential Data: Usernames and Pass words", 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 permissi ons →
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
Windows: Switch user accounts or create a new user, then try opening t he 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 logfi le.txt

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
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.

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

1. Set Up a Simple Web Server (Optional)

Linux: Use Python to start a basic web server: python3 -m http.server 8080
Windows: Use IIS or WAMP/XAMPP to set up a local server.

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

* Use a tool to overload the server with requests:
Linux: Install and use ab (Apache Benchmark):
ab -n 1000 -c 100 http://localhost:8080/
[-n: Total number of requests, -c: Number of concurrent requests.]

Windows: Use a custom PowerShell script or any load-testing tool like JMeter

3. Monitor Server Behavior

* Observe the server response time during the attack:
Linux: Check server logs or terminal output for delays or errors.
Windows: Use Task Manager or Resource Monitor to track CPU and netw
ork usage.

* Note any timeouts or connection refusals.

4. Restore Normal Operations

* Stop the attack and ensure normal availability:
Linux: Terminate the ab command or server process (Ctrl+C)
Windows: Stop the server or restart it via IIS Manager.

* Verify that the server is responsive again.

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:
mkdir dac_demo && cd dac_demo
touch confidential.txt

b) Set Permissions Using chmod:
chmod 600 confidential.txt

(for WSL) ~ sudo chmod o-rx /home/username/dac_demo

c) Change File Ownership Using chown
sudo adduser alice
sudo chown alice:alice confidential.txt

d) Log in as Alice:
su alice
cat confidential.txt

(if su doesn’t work on WSL) (FOR WSL)
sudo -i -u alice

sudo su - alice

ls /home/atharv/dac_demo ~ Output :
ls: cannot access '/home/usename/dac_demo': Permission denied

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

6. To configure and understand Mandatory Access Control (MAC) mechanisms in a Linux environment.

> `SELinux won't work.` ~ You would need a full Linux VM (like Fedora, CentOS, or RHEL) or a real Linux machine

> Requirement ~ Linux VM or Linux Machine

1. Mandatory Access Control (MAC)

   Create Files and Directories:
   mkdir mac_demo && cd mac_demo
   touch confidential.txt

   a) Enable SELinux:
   getenforce

   If SELinux is not enabled:
   sudo setenforce 1

   b) Apply Security Context to a File: (Create A file name : confidential.txt)
   ls -Z confidential.txt

   c) Test Policy Enforcement:
   Try accessing the file through an unauthorized process.
   sudo cat /var/log/audit/audit.log

   d) Results:
   SELinux denies access to unauthorized processes and logs the event in au
dit.log.
   Unauthorized processes will be denied access, even if they are run by 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):
sudo groupadd managers
sudo usermod -aG managers alice

b) Set Permissions on a File:
touch manager_notes.txt
sudo chown :managers manager_notes.txt
sudo chmod 770 manager_notes.txt

c) Test Access:
* As Alice ~
su alice
cd

To Exit alice to add new user ~ command : exit
To add a new user ~ command : sudo adduser <username>
To test with <uother_user> ~ command : su <other_user/username>

* As <other_User>
cat manager_notes.txt

d) Results:
Access to manager_notes.txt is controlled by group membership (role), not individual ownership.

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
   * Enter a PEM pass phrase ~ [Enter anything, but remember it]

3. Generate the public key from the private key:
   openssl rsa -pubout -in private.key -out public.key
   * Enter the pass phrase.

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

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 encryptio
n):
   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
   * Fill out the Distinguished Name (DN) fields:

8. Sign the user CSR with the CA key to generate the signed user certificate:
   openssl x509 -req -in user.csr -CA ca.crt -CAkey ca.key -CAcreateserial -o
ut user_signed_cert.crt -days 365
   * Enter the pass phrase for the CA key 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
    ip a

    nmap -sV -p- 10.10.10.6 (Check for your system IP address here) (eth0)
```

10. Perform AES Encryption & Decryptio

```
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 algorith
m.
    a. Generate an AES Key : (Create 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 : python3 aes_key.py
```

```
# Encrypt a Message (Create 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 :  python3 aes_encrypt.py
```

```
# Decrypt the Message (Create 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'

# Decrypt the message
```

```
decrypted_message = cipher.decrypt(encrypted_message).decode()
print(f"Decrypted Message: {decrypted_message}")


"""
[Note : ENCRYPTED_MESSAGE_HERE :  replace it with the encrypted message you received
from aes_encrypt.py ]
"""

# run : python3 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"
```

```
# 1. Generate RSA Key Pair (rsa_key.py)
# 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.")
```

```
# Encrypt Data Using RSA Public Key (rsa_encrypt.py)
# 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}")
```

```
# Decrypt Data Using RSA Private Key (rsa_decrypt.py)
# 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 (use the result from rsa_encrypt.py)
encrypted_message = b'ENCRYPTED_MESSAGE_HERE'  # Replace with the a
ctual encrypted message

# Decrypt the message using RSA private key
decrypted = private_key.decrypt(
    encrypted_message,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
```

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

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
```

```
# Masking Sensitive Data (Anonymization) (Create nano mask.py)
# 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=Tr
ue)

# Display the masked data
print(df)
```

```
# 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())
```