Swami Vivekānanda Yoga Anusandhāna Saṁsthāna(S-VYASA)
(Deemed to be University)

School of Advanced Studies

LAB RECORD

Student Name **Prince Kumar**

USN **: 2222408023**

Course Code **: MCCE347**

Course Name **: Fundamentals of Cybersecurity Lab**

Class/Semester **: MCA/ III Semester**

Department **: Computer Science and Applications**

Academic Year **: 2025 – 2026**

Swami Vivekānanda Yoga Anusandhāna Saṁsthāna(S-VYASA)
(Deemed to be University)

BONAFIDE CERTIFICATE

Certified that this is a bonafide record of the practical work done by Mr. **Prince Kumar** USN : **2222408023** of Semester **3**, for **MCA ComputerScience(Cybersecurity, Ethical Hacking and Cyber Forensics)** during the academic year **2025–2026** in the **MCCE347 Fundamentals of Cybersecurity** Laboratory.

Date: **FACULTY IN-CHARGE**

EXAMINER

INDEX

Experiment No: 01

Experiment Name: Develop a Python program that categorizes cyber threats based on severity levels.

Aim : To develop a Python program that categorizes different types of cyber threats into severity levels (Low, Medium, High, Critical) based on their impact and likelihood, and to generate a final report summarizing the categorization.

Algorithm /Procedure :

Step 1: Start the program

Step 2: Define a class CyberThreat

Initialize attributes: `name`, `impact`, `likelihood`, `threat_type`, `severity`.

Define method `calculate_severity()`

1. Compute **risk score = impact × likelihood**.
2. Categorize severity:
   - If risk score ≤ 20 → `"Low"`.
   - If 21 ≤ risk score ≤ 40 → `"Medium"`.
   - If 41 ≤ risk score ≤ 70 → `"High"`.
   - Else → `"Critical"`.

Define method `display_info()` to print threat details.

Step 3: Define the main() function

1. Display program title.
2. Initialize an empty list `threats`.
3. **Repeat until user chooses to stop**:
   - Accept user input for `threat name` and `threat type`.
   - Validate input for `impact` (must be 1–10).
   - Validate input for `likelihood` (must be 1–10).
   - Create a `CyberThreat` object with given details.
   - Append the object to `threats` list.
   - Display categorized threat information.
   - Ask the user whether to add another threat.
4. After exiting the loop, display the **Final Categorization Report** by printing details of all threats stored in `threats` list.

Step 4: End the program.

Program

```
# Interactive Cyber Threat Categorization Tool

class CyberThreat:
    def __init__(self, name, impact, likelihood, threat_type):
        """
        :param name: Name of the threat (string)
        :param impact: Impact value (1-10)
        :param likelihood: Likelihood value (1-10)
        :param threat_type: Type of threat (string)
        """
        self.name = name
        self.impact = impact
        self.likelihood = likelihood
        self.threat_type = threat_type
        self.severity = self.calculate_severity()

    def calculate_severity(self):
        """
        Determine severity level based on risk score.
        Risk Score = Impact × Likelihood
        """
        risk_score = self.impact * self.likelihood

        if risk_score <= 20:
            return "Low"
        elif 21 <= risk_score <= 40:
            return "Medium"
        elif 41 <= risk_score <= 70:
            return "High"
        else:
            return "Critical"

    def display_info(self):
        print(f"\nThreat: {self.name}")
        print(f"  Type: {self.threat_type}")
        print(f"  Impact: {self.impact}")
        print(f"  Likelihood: {self.likelihood}")
        print(f"  Severity Level: {self.severity}")
        print("-" * 40)


def main():
    print("\n Cyber Threat Categorization Tool ")
    print("=" * 50)
```

```
    threats = []

    while True:
        name = input("\nEnter threat name: ")
        threat_type = input("Enter threat type (e.g., Malware, Phishing, DDoS): ")

        # Get valid impact and likelihood scores
        while True:
            try:
                impact = int(input("Enter impact (1-10): "))
                if 1 <= impact <= 10:
                    break
                else:
                    print("Impact must be between 1 and 10.")
            except ValueError:
                print("Please enter a valid number.")

        while True:
            try:
                likelihood = int(input("Enter likelihood (1-10): "))
                if 1 <= likelihood <= 10:
                    break
                else:
                    print("Likelihood must be between 1 and 10.")
            except ValueError:
                print("Please enter a valid number.")

        threat = CyberThreat(name, impact, likelihood, threat_type)
        threats.append(threat)
        threat.display_info()

        # Ask if user wants to enter more threats
        choice = input("\nDo you want to add another threat? (y/n): ").lower()
        if choice != "y":
            break

    # Final Summary
    print("\nFinal Categorization Report")
    print("=" * 50)
    for t in threats:
        t.display_info()


if __name__ == "__main__":
    main()
```

Output:

```
Cyber Threat Categorization Tool

Enter threat name: Ransomware
Enter threat type (e.g., Malware, Phishing, DDoS): Malware
Enter impact (1-10): 9
Enter likelihood (1-10): 8

Threat: Ransomware
Type: Malware
Impact: 9
Likelihood: 8
Severity Level: Critical
----------------------------------------

Do you want to add another threat? (y/n): y

Enter threat name: Phishing
Enter threat type (e.g., Malware, Phishing, DDoS): Social Engineering
Enter impact (1-10): 5
Enter likelihood (1-10): 6

Threat: Phishing
Type: Social Engineering
Impact: 5
Likelihood: 6
Severity Level: Medium
----------------------------------------

Do you want to add another threat? (y/n): n

Final Categorization Report
Threat: Ransomware
Type: Malware
Impact: 9
Likelihood: 8
Severity Level: Critical
----------------------------------------
Threat: Phishing
Type: Social Engineering
Impact: 5
Likelihood: 6
Severity Level: Medium
```

Experiment No: 02

Experiment Name: Write Python Script Using Scapy to Detect Open Ports and Vulnerabilities in Network

Aim: To develop a Python script using the Scapy library to scan a target system, detect open ports, and identify possible vulnerabilities associated with commonly exploited services.

Algorithm / Procedure:

Step 1: Start the Program

Step 2: Import Required Libraries

Import required components from Scapy (IP, TCP, sr1, conf).

Import socket module for resolving port names.

Disable Scapy verbose mode.

Step 3: Define a Dictionary of Common Vulnerable Ports

Map well-known ports to their typical vulnerabilities.

Step 4: Define scan_host() Function

1. Print scanning header.
2. Initialize an empty list to store open ports.
3. Loop through each port in the given range:
    - Create a TCP SYN packet.
    - Send packet and wait for response.
    - If response is SYN-ACK → port is open.
    - Send RST packet to close connection.
    - Display open ports and check if port appears in vulnerability list.
4. After scanning, print summary of open ports.

Step 5: Main Execution

1. Ask the user for a target IP address.
2. Call scan_host() with port range 1–1025.

Step 6: End the Program

Program:

```
from scapy.all import IP, TCP, sr1, conf
import socket

# Disable verbose output
conf.verb = 0

# Dictionary of common vulnerable ports and issues
vuln_ports = {
    21: "FTP - Anonymous login or cleartext credentials",
    22: "SSH - Weak passwords / outdated versions",
    23: "Telnet - Insecure (cleartext communication)",
    25: "SMTP - Open relay or weak authentication",
    53: "DNS - Misconfiguration or cache poisoning",
    80: "HTTP - Outdated web server, injection flaws",
    110: "POP3 - Cleartext credentials",
    139: "NetBIOS - File sharing vulnerabilities",
    143: "IMAP - Cleartext credentials",
    443: "HTTPS - SSL/TLS vulnerabilities",
    445: "SMB - EternalBlue or weak access control",
    3306: "MySQL - Default credentials",
    3389: "RDP - Remote Desktop vulnerabilities",
    8080: "HTTP Proxy - Weak authentication"
}

def scan_host(target_ip, port_range=(1, 1025)):
    print(f"\nScanning host: {target_ip}\n{'=' * 40}")

    open_ports = []

    for port in range(port_range[0], port_range[1] + 1):
        pkt = IP(dst=target_ip) / TCP(dport=port, flags="S")  # SYN packet
        resp = sr1(pkt, timeout=0.5)

        if resp is None:
            continue

        if resp.haslayer(TCP) and resp.getlayer(TCP).flags == 0x12:  # SYN-ACK
            # Send RST to close connection
            sr1(IP(dst=target_ip) / TCP(dport=port, flags="R"), timeout=0.5)
            open_ports.append(port)
```

```
            try:
                service = socket.getservbyport(port, "tcp") if port < 1024 else "unknown"
            except:
                service = "unknown"

            print(f"[OPEN] Port {port} ({service})")

            if port in vuln_ports:
                print(f"  ⚠ Potential Vulnerability: {vuln_ports[port]}")

    if not open_ports:
        print("No open ports detected in the given range.")
    else:
        print(f"\nScan complete. Open ports: {open_ports}\n")

if __name__ == "__main__":
    target = input("Enter target IP address: ")
    scan_host(target, port_range=(1, 1025))
```

Output:

```
Enter target IP address: 10.102.31.226
□□ Scanning host: 10.102.31.226
==========================================
[OPEN] Port 135 (epmap)
[OPEN] Port 139 (netbios-ssn)
⚠️ Potential Vulnerability: NetBIOS - File sharing vulnerabilities
[OPEN] Port 445 (microsoft-ds)
⚠️ Potential Vulnerability: SMB - EternalBlue or weak access control
✅ Scan complete. Open ports: [135, 139, 445]
```

Experiment No: 03

Experiment Name: Write a Python Script to Generate Fake Phishing Emails for Educational Purposes

Aim: To develop a Python script that generates mock phishing-style emails for cybersecurity awareness and educational training, without using any real or harmful phishing domains.

Algorithm / Procedure (Step-by-Step):

Step 1: Start the Program

Step 2: Import Required Modules

- Import the random module for selecting random email components.

Step 3: Define Email Components

1. Create lists for:
    - Fake sender names (training labels)
    - Subject lines
    - Greetings
    - Email body messages
    - Fake placeholder links (non-real, non-actionable)
    - Sign-off lines

Step 4: Define generate_phishing_email() Function

1. Randomly select:
    - Sender
    - Subject
    - Greeting
    - Body
    - Fake link placeholder
    - Sign-off
2. Format them into an email template.
3. Return the final mock phishing email.

Step 5: Main Execution

1. Display program title.
2. Generate a sample phishing email.
3. Print the generated fake email for demonstration.

Step 6: End the Program

Program:

```
import random
# Training-safe fake sender addresses
senders = [
    "Admin <admin@training-example.com>",
    "IT Helpdesk <helpdesk@secure-training.net>",
    "Security Team <security@education-lab.org>",
    "Bank Alert <alerts@bank-training.com>",
    "Customer Care <support@mock-service.org>"
]
# Phishing-style subject lines (for awareness)
subjects = [
    "URGENT: Verify your account immediately!",
    "Password Expiration Notice - Action Required",
    "Unusual Login Attempt Detected",
    "Congratulations! You've won a reward!",
    "Payment Failed: Update your billing info now"
]
# Greetings
greetings = [
    "Dear Customer,",
    "Hello User,",
    "Attention Account Holder,",
    "Dear Valued Member,"
]
# Suspicious-style body messages
bodies = [
    "We noticed unusual activity on your account. Please review it immediately.",
    "Your password will expire soon. Reset it to avoid service interruption.",
```

```
    "You are eligible for a limited-time prize. Claim it before it expires!",
    "We could not process your recent payment. Update your billing details.",
    "Your account may be suspended unless you verify your information."
]


# SAFE placeholder links (non-functional)
fake_links = [
    "http://training-link-example.com",
    "http://secure-education.net/reset",
    "http://awareness-lab.org/verify",
    "http://mock-notification.edu/action",
    "http://cyber-training-demo.com/update"
]
# Email sign-offs
signoffs = [
    "Best regards,\nIT Security Team",
    "Thank you,\nAccount Support",
    "Sincerely,\nCustomer Service",
    "Yours truly,\nSecurity Department"
]
def generate_phishing_email():
    """Generate a safe, fake phishing email for training."""
    sender = random.choice(senders)
    subject = random.choice(subjects)
    greeting = random.choice(greetings)
    body = random.choice(bodies)
    link = random.choice(fake_links)
    signoff = random.choice(signoffs)

    email = f"""
From: {sender}
Subject: {subject}
```

```
{greeting}
{body}

[Click here to proceed]({link})

{signoff}
"""
    return email

if __name__ == "__main__":
    print("\n=== Fake Phishing Emails (Training Purposes Only) ===\n")

    print(generate_phishing_email())
    print("=" * 60)
```

Output:

=== Fake Phishing Emails (For Training Purposes Only) ===

From: IT Helpdesk <helpdesk@account-verify.net>

Subject: Congratulations! You've won a reward □□

Dear Customer,

We could not process your recent payment. Update your billing details immediately.

□□ [Click here to proceed](http://amaz0n-secure-billing.ru)

Thank you,

Account Support

Experiment No: 04

Experiment Name: Develop a Python Script Using YARA Rules to Detect Malware Signatures in Files

Aim: To develop a Python script that uses YARA rules to scan files and detect known malware signatures based on text and hexadecimal patterns.

Algorithm / Procedure:

Step 1: Start the Program

Step 2: Import Required Libraries

- Import the yara module to compile and apply YARA rules.

Step 3: Define a YARA Rule

1. Create a rule as a multi-line string.
2. Add two signature patterns inside the rule:
    - A text string pattern ("malicious_code").
    - A hexadecimal byte pattern ({ E8 ?? ?? ?? ?? 83 C4 04 }) with wildcards.
3. Set the condition to trigger if either pattern is found in the file.

Step 4: Compile the YARA Rule

- Use yara.compile(source=rule) to convert the rule into a form that can be applied to files.

Step 5: Get File Path from User

- Prompt the user to enter the path of the file they want to scan.

Step 6: Scan the File for Malware Signatures

1. Apply the compiled rules to the file using rules.match(file_path).
2. Store the results in a variable called matches.

Step 7: Display Results

- If matches are found:
    - Print alert message.
    - Loop through all matched rules and display their names.
- If no matches:
    - Print a clean file message.

Step 8: End the Program

Program

```
# Import the yara module
import yara
# Step 1: Define a simple YARA rule as a string
# The rule has two string patterns ($a and $b) and a condition
# It will trigger if either of them is found in the file
rule = """
rule MalwareExample
{
   strings:
      $a = "malicious_code"        // A text string pattern
      $b = { E8 ?? ?? ?? ?? 83 C4 04 }   // A hex pattern with wildcards (??)

   condition:
      $a or $b                     // Rule matches if either string is found
}
"""

# Step 2: Compile the YARA rule
rules = yara.compile(source=rule)

# Step 3: Ask user for file path
file_path = input("Enter the file path to scan: ")

# Step 4: Scan the file with the compiled rules
matches = rules.match(file_path)

# Step 5: Display results
if matches:
   print("\n⚠️ Malware detected! Signature(s) matched:")

   # Loop through all matches and display their rule names
   for match in matches:
      print(" -", match.rule)
else:
   print("\n✅ No malware signature detected in the file.")
```

Output:

Enter the file path to scan: testfile.exe
⚠️ Malware detected! Signature(s) matched:
- MalwareExample
Or, if no malicious content is found:

Enter the file path to scan: safe_document.txt
✅ No malware signature detected in the file.

Experiment No: 05

Experiment Name: Write a Python Script Using Scapy to Capture and Analyze Network Traffic

Aim: To develop a Python script that uses the Scapy library to capture live network traffic, analyze packets based on protocol (TCP/UDP/Other), and save the captured packets into a .pcap file.

Algorithm / Procedure

Step 1: Start the program

Step 2: Import required libraries

- Import Scapy modules: sniff, IP, TCP, UDP, wrpcap.

Step 3: Initialize storage

- Create an empty list captured_packets to store packets.

Step 4: Define packet_callback()

- Check if the packet contains an IP layer.
- Extract source/destination IP and protocol.
- If TCP → extract ports and print details.
- If UDP → extract ports and print details.
- Else → print protocol number.
- Append packet to storage list.

Step 5: Define main()

- Display "Starting packet capture…".
- Start sniffing packets using sniff(prn=packet_callback).
- Stop when user presses Ctrl + C.
- Save all captured packets to a .pcap file using wrpcap().

Step 6: Execute main()

- Use if __name__ == "__main__": to run the program.

Step 7: End the program

Program:

```
from scapy.all import sniff, IP, TCP, UDP, wrpcap
# List to store captured packets
captured_packets = []
def packet_callback(packet):
    """Callback function to process each captured packet."""
    if IP in packet:  # Ensure it's an IP packet
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        proto = packet[IP].proto
        if TCP in packet:
            sport = packet[TCP].sport
            dport = packet[TCP].dport
            print(f"[TCP] {ip_src}:{sport} -> {ip_dst}:{dport}")
        elif UDP in packet:
            sport = packet[UDP].sport
            dport = packet[UDP].dport
            print(f"[UDP] {ip_src}:{sport} -> {ip_dst}:{dport}")
        else:
            print(f"[OTHER] {ip_src} -> {ip_dst} (Protocol: {proto})")
        # Store packet for saving later
        captured_packets.append(packet)
def main():
    print("Starting packet capture... Press Ctrl+C to stop.")
    try:
        # Capture packets (filter can be added, e.g., filter='tcp')
        sniff(prn=packet_callback, store=False)
    except KeyboardInterrupt:
        print("\nCapture stopped.")
    # Save captured packets to file
    pcap_file = "captured_traffic.pcap"
    wrpcap(pcap_file, captured_packets)
    print(f"Saved {len(captured_packets)} packets to {pcap_file}")
```

```
if __name__ == "__main__":
    main()
```

Output:

```
Starting packet capture... Press Ctrl+C to stop.
[UDP] 10.101.39.99:1900 -> 239.255.255.250:1900
[OTHER] 10.101.45.254 -> 224.0.0.22 (Protocol: 2)
[UDP] 10.101.33.215:53628 -> 224.0.0.252:5355
[OTHER] 10.101.33.215 -> 224.0.0.22 (Protocol: 2)
[OTHER] 10.101.33.215 -> 224.0.0.22 (Protocol: 2)
[UDP] 10.101.41.17:5353 -> 224.0.0.251:5353
[UDP] 10.101.52.24:37345 -> 239.255.255.250:1900
[UDP] 10.101.51.149:137 -> 10.101.127.255:137
[UDP] 10.101.31.16:53125 -> 239.255.255.250:1900
[UDP] 10.101.39.99:1900 -> 239.255.255.250:1900
[TCP] 10.101.36.80:60885 -> 93.123.17.254:80
[TCP] 93.123.17.254:80 -> 10.101.36.80:60885
[TCP] 10.101.36.80:60885 -> 93.123.17.254:80
[TCP] 10.101.36.80:60885 -> 93.123.17.254:80
[TCP] 93.123.17.254:80 -> 10.101.36.80:60885
[TCP] 93.123.17.254:80 -> 10.101.36.80:60885
[UDP] 10.101.36.82:138 -> 10.101.127.255:138
[UDP] 10.101.31.228:5353 -> 224.0.0.251:5353
```

Experiment No: 06

Experiment Name: Develop a Python Program Using PyCryptodome to Encrypt and Decrypt a File Using AES

Aim: To write a Python program that uses the PyCryptodome library to perform AES encryption and decryption on a file using AES-GCM mode, ensuring both confidentiality and integrity.

Algorithm / Procedure

Step 1: Start the Program

Step 2: Import Required Libraries

- Import AES from Crypto.Cipher
- Import get_random_bytes for key generation

Step 3: Generate AES Key

- Create a 32-byte (256-bit) random AES key.

Step 4: Define encrypt_file() Function

1. Read input file in binary.
2. Create AES cipher in GCM mode.
3. Encrypt data and generate authentication tag.
4. Write nonce + tag + ciphertext to output file.

Step 5: Define decrypt_file() Function

1. Read nonce, tag, and ciphertext from encrypted file.
2. Decrypt using AES-GCM with the stored nonce.
3. Verify tag (ensures integrity).
4. Write decrypted data to output file.

Step 6: Main Execution

1. Create a sample text file.
2. Call encryption function.
3. Call decryption function.

Step 7: End the program

Program:

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
# AES requires a 16, 24, or 32-byte key
KEY = get_random_bytes(32)  # 256-bit AES key
def encrypt_file(input_file, output_file):
    with open(input_file, "rb") as f:
        data = f.read()
    cipher = AES.new(KEY, AES.MODE_GCM)
    ciphertext, tag = cipher.encrypt_and_digest(data)
    with open(output_file, "wb") as f:
        # Store nonce, tag, and ciphertext together
        f.write(cipher.nonce)
        f.write(tag)
        f.write(ciphertext)
    print(f"File '{input_file}' encrypted successfully as '{output_file}'")
def decrypt_file(input_file, output_file):
    with open(input_file, "rb") as f:
        nonce = f.read(16)        # AES-GCM nonce is 16 bytes
        tag = f.read(16)          # Authentication tag
        ciphertext = f.read()
    cipher = AES.new(KEY, AES.MODE_GCM, nonce=nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
    with open(output_file, "wb") as f:
        f.write(data)
    print(f"File '{input_file}' decrypted successfully as '{output_file}'")
```

```
if __name__ == "__main__":
    # Create a sample file
    with open("sample.txt", "w") as f:
        f.write("This is a secret message!")
    # Encrypt and decrypt
    encrypt_file("sample.txt", "encrypted.bin")
    decrypt_file("encrypted.bin", "decrypted.txt")
```

Output:

```
File 'sample.txt' encrypted successfully as 'encrypted.bin'
This is a secret message!
File 'encrypted.bin' decrypted successfully as 'decrypted.txt'
```

Experiment No: 07

Experiment Name: Write a Python Script Using bcrypt to Hash Passwords and Verify Them Securely

Aim: To develop a Python script that uses the bcrypt library to securely hash passwords and verify user-entered passwords by comparing them with stored hashed values.

Algorithm / Procedure:

Step 1: Start the Program

Step 2: Import Required Library

- Import the bcrypt module for hashing and verifying passwords.

Step 3: Define hash_password() Function

1. Convert plaintext password to bytes.
2. Generate a salt using bcrypt.gensalt().
3. Hash the password with bcrypt.hashpw().
4. Return hashed password.

Step 4: Define verify_password() Function

1. Convert plaintext password to bytes.
2. Compare with stored hashed password using bcrypt.checkpw().
3. Return verification result (True/False).

Step 5: Main Execution

1. Ask the user for a password.
2. Hash and display the hashed password.
3. Ask user to re-enter password.
4. Verify password and print success/failure message.

Step 6: End the Program

Program:

```
import bcrypt
def hash_password(plain_password: str) -> bytes:
    """
    Hash a plaintext password using bcrypt.
    """
    # Convert the password to bytes
    password_bytes = plain_password.encode('utf-8')
    # Generate a salt and hash the password
    salt = bcrypt.gensalt()
    hashed_password = bcrypt.hashpw(password_bytes, salt)
    return hashed_password
def verify_password(plain_password: str, hashed_password: bytes) -> bool:
    """
    Verify a plaintext password against a hashed password.
    """
    # Convert plain password to bytes
    password_bytes = plain_password.encode('utf-8')
    # Check if the password matches
    return bcrypt.checkpw(password_bytes, hashed_password)
if __name__ == "__main__":
    # Example usage
    user_password = input("Enter your password: ")
    # Hash the password
    hashed = hash_password(user_password)
    print(f"Hashed password (store this safely): {hashed.decode()}")
    # Verify password
    check_password = input("Re-enter password to verify: ")
    if verify_password(check_password, hashed):
        print("Password match! Access granted.")
    else:
        print("Password does not match. Access denied.")
```

Output:

Enter your password: hello

Hashed password (store this safely):

$2b$12$7162AUXEs1CbwfEgpBJ3UuDC1EQMIqe0cFWvpqJ/rQxTfKklSH9VS

Re-enter password to verify: hello

Password match! Access granted.

Experiment No: 08

Experiment Name: Write a Python Program Using Cryptography to Sign and Verify Messages

Aim: To write a Python program that uses RSA cryptography to digitally sign a message using a private key and verify the signature using the corresponding public key.

Algorithm / Procedure:

Step 1: Start the Program

Step 2: Import Required Libraries

- Import RSA, padding, and hashing modules from cryptography.

Step 3: Generate RSA Key Pair

- Generate a **private key** using rsa.generate_private_key().
- Extract the **public key** using private_key.public_key().

Step 4: Prepare Message

- Define the message to be signed in bytes format.

Step 5: Sign the Message

- Use private_key.sign() with:
  - **PSS padding**
  - **SHA256 hash function**
- Store the resulting digital signature.

Step 6: Display Message and Signature

- Print the message and generated signature.

Step 7: Verify the Signature

- Use public_key.verify() with the same padding and hashing scheme.
- If verification succeeds → print success message.
- If verification fails → print error message.

Step 8: End the Program

Program:

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
# Generate RSA private and public keys
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048
)
public_key = private_key.public_key()
# Message to be signed
message = b"This is a secret message."
# ---- Signing ----
signature = private_key.sign(
    message,
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
print("Message:", message)
print("Digital Signature:", signature)
# ---- Verification ----
try:
    public_key.verify(
        signature,
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
```

```
    )
    print("Signature Verified Successfully")
except Exception as e:
    print("Verification Failed:", str(e))
```

Output:

```
Message: b'This is a secret message.'
Digital Signature:
b'\xb2\xb9\x97)8\rx\x81\x1e\xc4\xed.\xb94\xc6P\xfa\x14\xdb\xa8\x86pF\xc7\x81\rv\x15\x0e\xe4h\x
83\xef\xc8x\x13\x1f7\xb9\xf0\xea7I\xc0\xb6<6\x95M\x19\xe6SY%\xd3f\xa5\x9b\xf1\xe8\xd3L\xd
d\xd6\x95\xc0\'*F\x12:]\xd3]\xf0\xf4Y\xda\xa1?\xa8\xdf\xc9\xbf\xf4\xdeYc\xf3\'*\xd4Vc\x1cd"{
\xa5|\xc4Tc\xedqN\xf6P\xdc\x7f\x1eg\xea\xc6\x9b\x80\x01\x9dI\xaa\xa4(\xf7o\xa0h%p<\xf0\x9
1R$\xf6\xdd\xdcW\x85Z\xa9A\x1e:\r(k\xd3\x0e\xce\xf4\xe8\x84\xe3r\xae\x85\\\xc2}}\xf1?o\xf76
8Ib\xb3.\x1c\x05\x17d\x1c\x88\xfe\x8fA=L\xa6\xcb:K\xf3\\\xa1\xb8\xb8\x1c\xb7\x8c\xe8o\xad\x
c8E\xcdz\xe07z\x9d\xce6\xe9$-
*\xad\xe5/\x8d@,\x8a\xfa\xea:\xd5\xc1]a\xe7\x16y\xa9\xde\xfb\x18C\x81W_)\xc9\xed\x0e\t\x967
X\x01\x9dT\xe1#\xb8\'\x87e\xf4\x90G\x02'
Signature Verified Successfully
```

Experiment No: 9

Experiment Name: Develop a Python Script Using Pystegano to Hide and Extract Messages in Images

Aim: To create a Python script that uses the Pystegano (Stegano) library to securely hide secret messages inside images using LSB (Least Significant Bit) steganography and extract them back when needed.

Algorithm / Procedure:

Step 1: Start the Program

Step 2: Import Required Library

- Import lsb module from stegano.

Step 3: Define hide_message() Function

1. Take input image, output image, and secret message.
2. Hide the message using lsb.hide().
3. Save the resulting stego-image.
4. Print confirmation.

Step 4: Define reveal_message() Function

1. Take stego-image as input.
2. Extract hidden message using lsb.reveal().
3. If message exists → print it.
4. Else print "No hidden message found".

Step 5: Main Execution

1. Call hide_message() to embed a secret message.
2. Call reveal_message() to extract the hidden message.

Step 6: End the Program

Program:

```
from stegano import lsb
# ---- Hide a secret message in an image ----
def hide_message(input_image, output_image, secret_message):
    secret = lsb.hide(input_image, secret_message)
    secret.save(output_image)
    print(f"Message hidden successfully in '{output_image}'")
# ---- Reveal the hidden message from an image ----
def reveal_message(stego_image):
    message = lsb.reveal(stego_image)
    if message:
        print("Extracted Message:", message)
    else:
        print("No hidden message found!")
# ---- Example Usage ----
if __name__ == "__main__":
    # Hide a message
    hide_message("input.png", "stego_image.png", "This is a top-secret message!")
    # Extract the message
    reveal_message("stego_image.png")
```

Output:

Message hidden successfully in stego_image.png;

Extracted Message: This is a top-secret message!

Experiment No: 10

Experiment Name: Create a simple web application that is vulnerable to SQL Injection and demonstrate exploitation.Application

Aim: To study how SQL Injection vulnerabilities occur in insecure web applications by analyzing an intentionally vulnerable Flask-based user lookup system and understanding how unsanitized input can lead to database exploitation.

Algorithm / Procedure

Step 1: Start the Program

- Launch the Flask application and initialize/reset the SQLite database.

Step 2: Import Required Libraries

- Import Flask for routing,
- sqlite3 for database handling,
- os for file operations.

Step 3: Initialize the Database

- Delete old database file.
- Create a new users table.
- Insert sample user records.

Step 4: Define the Home Route (/)

- Display a username input form.
- Show example SQL injection payloads for learning.

Step 5: Define the Vulnerable Lookup Route (/find)

- Read user input from the form.
- Build an SQL query using string concatenation (**unsafe**).
- Execute the query and display results.

Step 6: Demonstrate SQL Injection

- Enter payloads like ' OR '1'='1 to view unauthorized data.
- Observe how the vulnerable query leaks information.

Step 7: Run the Server

- Application runs on http://127.0.0.1:5000.
- Console shows executed SQL queries for analysis.

Step 8: End the Program

- Stop the server manually using **Ctrl + C**.

Program:

```
from flask import Flask, request, render_template_string, g
import sqlite3
import os
DATABASE = 'vulnerable.db'
app = Flask(__name__)
def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DATABASE)
        db.row_factory = sqlite3.Row
    return db
@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db:
        db.close()
def init_db():
    if os.path.exists(DATABASE):
        os.remove(DATABASE)  # Start fresh
    con = sqlite3.connect(DATABASE)
    cur = con.cursor()
    cur.execute('''
      CREATE TABLE users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL
      );
    ''')
```

```
    cur.execute("INSERT INTO users (username, password) VALUES ('admin', 'SuperSecret123!');")
    cur.execute("INSERT INTO users (username, password) VALUES ('alice', 'AlicePassword456');")
    cur.execute("INSERT INTO users (username, password) VALUES ('bob', 'BobsPassword789');")
    cur.execute("INSERT INTO users (username, password) VALUES ('eve', 'EvesSecret!@#');")
    con.commit()

    con.close()
@app.route('/')
def index():
    return render_template_string('''
      <h2>Vulnerable User Lookup</h2>
      <form action="/find" method="get">
        <input name="username" placeholder="username" style="width: 300px;">
        <button type="submit">Find User</button>
      </form>
      <br>
      <h3>Try these SQL Injection payloads:</h3>
      <ul>
        <li><code>' OR '1'='1</code> - Show all users</li>
        <li><code>' UNION SELECT id, password FROM users --</code> - Get passwords</li>
        <li><code>' UNION SELECT username, password FROM users --</code> - Get usernames &
passwords</li>
      </ul>
    ''')
@app.route('/find')
def find_user():
    username = request.args.get('username', '')
    # VULNERABLE CODE - DO NOT USE IN PRODUCTION
    query = f"SELECT id, username FROM users WHERE username = '{username}'"
    print(f"[VULNERABLE] Executing: {query}")  # Debug output
    try:
        cur = get_db().execute(query)  # UNSAFE!
        rows = cur.fetchall(
```

```
        if rows:
            result = "<h3>Results:</h3><table
border='1'><tr><th>ID</th><th>Username/Data</th></tr>"
            for row in rows:
                result += f"<tr><td>{row['id']}</td><td>{row['username']}</td></tr>"
            result += "</table>"
            return result + "<br><a href='/'>Back</a>"

        else:
            return "No results found.<br><a href='/'>Back</a>"
    except Exception as e:
        return f"Error: {str(e)}<br><a href='/'>Back</a>"
if __name__ == '__main__':
    init_db()
    print("Vulnerable app running on http://127.0.0.1:5000")
    print("WARNING: This app is intentionally vulnerable to SQL injection!")
    app.run(host='127.0.0.1', port=5000, debug=False)
```

Output:

Vulnerable User Lookup

' UNION SELECT username, password FROM us | Find User

Try these SQL Injection payloads:

- `' OR '1'='1` - Show all users
- `' UNION SELECT id, password FROM users --` - Get passwords
- `' UNION SELECT username, password FROM users --` - Get usernames & passwords

Results:

ID	Username/Data
admin	SuperSecret123!
alice	AlicePassword456
bob	BobsPassword789
eve	EvesSecret!@#

Back