

List of Experiments

1. Analyze and visualize IP address allocation and subnetting using a Python script with Scapy and Matplotlib.

```
#!/usr/bin/env python3
from scapy.all import sniff, IP
from collections import Counter
import matplotlib.pyplot as plt
import ipaddress

# Counter to store IP occurrence
ip_count = Counter()

def capture_packets(packet):
    if packet.haslayer(IP):
        # Count both source and destination IPs
        ip_count[packet[IP].src] += 1
        ip_count[packet[IP].dst] += 1

print("Sniffing 50 packets. (Make sure you have appropriate privileges!)")
sniff(prn=capture_packets, count=50)

# Plot the IP frequency distribution
ips = list(ip_count.keys())
counts = list(ip_count.values())

plt.figure(figsize=(10, 5))
plt.bar(ips, counts, color='blue')
plt.xticks(rotation=45)
plt.xlabel("IP Addresses")
plt.ylabel("Number of Packets")
plt.title("IP Address Allocation Analysis")
plt.tight_layout()
plt.show()

# Classify IPs as private or public
def classify_ip(ip):
    try:
        ip_obj = ipaddress.ip_address(ip)
        return "Private" if ip_obj.is_private else "Public"
    except ValueError:
        return "Invalid IP"

print("\nIP Classification:")
for ip in ips:
    print(f"{ip}: {classify_ip(ip)}")
```

2. Develop a Python program to read and modify Linux network configuration files such as `/etc/hosts`, `/etc/resolv.conf`, and `/etc/network/interfaces`.

```
#!/usr/bin/env python3
import os
import shutil

# Path to Windows hosts file
HOSTS_FILE = r"C:\Windows\System32\drivers\etc\hosts"

def read_hosts():
    if not os.path.exists(HOSTS_FILE):
        print("[ERROR] Hosts file not found!")
        return
    with open(HOSTS_FILE, 'r') as file:
        data = file.read()
        print('=== Hosts File Contents ===')
        print(data)

def modify_hosts():
    if not os.path.exists(HOSTS_FILE):
        print("[ERROR] Hosts file not found!")
        return
    try:
        # Backup the original hosts file
        backup = HOSTS_FILE + ".bak"
        shutil.copy(HOSTS_FILE, backup)
        with open(HOSTS_FILE, 'a') as file:
            file.write("\n192.168.1.100 mycustomhost\n")
        print("[SUCCESS] Hosts file updated. Backup created at:", backup)
    except Exception as e:
        print("[ERROR] Failed to update hosts file:", e)

def restart_network():
    try:
        # Flush DNS cache to apply changes
        os.system("ipconfig /flushdns")
        print("[SUCCESS] DNS cache flushed. Changes applied.")
    except Exception as e:
        print("[ERROR] Failed to flush DNS cache:", e)

if __name__ == "__main__":
    read_hosts()
    modify_hosts()
    restart_network()
```

3. Write a Python script to capture and analyze active TCP/IP daemons running on a system using `psutil` and `netstat`.

```
#!/usr/bin/env python3
```

```
import psutil
```

```
def list_tcp_connections():
```

```
    for conn in psutil.net_connections(kind='tcp'):
```

```
        laddr = f"{conn.laddr.ip}:{conn.laddr.port}" if conn.laddr else "N/A"
```

```
        raddr = f"{conn.raddr.ip}:{conn.raddr.port}" if conn.raddr else "N/A"
```

```
        print(f"PID: {conn.pid}, Local Address: {laddr}, Remote Address: {raddr},
```

```
Status: {conn.status}")
```

```
if __name__ == "__main__":
```

```
    print("Listing active TCP connections:")
```

```
    list_tcp_connections()
```

4. Create a simple Python-based network daemon that listens on a specified port and logs incoming connections.

```
#!/usr/bin/env python3
```

```
import socket
```

```
def start_daemon(host='0.0.0.0', port=5000):
```

```
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    server_socket.bind((host, port))
```

```
    server_socket.listen(5)
```

```
    print(f"Network daemon listening on {host}:{port}...")
```

```
    while True:
```

```
        client_socket, client_address = server_socket.accept()
```

```
        print(f"Connection received from {client_address}")
```

```
        # Send a welcome message and then close the connection
```

```
        client_socket.sendall(b"Hello! You are connected to the network daemon.\n")
```

```
        client_socket.close()
```

```
if __name__ == "__main__":
```

```
    start_daemon()
```

5. Develop a Python script to scan and list open ports on a target machine using `socket` and `nmap` libraries.

```
#!/usr/bin/env python3
```

```
import socket
```

```
def scan_ports_socket(target, ports):
```

```
    print("=== Scanning ports using socket ===")
```

```
    for port in ports:
```

```
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
```

```

    sock.settimeout(1) # Set timeout to 1 second
    result = sock.connect_ex((target, port))
    if result == 0:
        print(f'Port {port} is OPEN')
    else:
        print(f'Port {port} is CLOSED or FILTERED')

if __name__ == "__main__":
    # Set target IP and ports to scan
    target_ip = "127.0.0.1" # Change as needed
    ports_to_scan = [22, 80, 443, 8080]

    scan_ports_socket(target_ip, ports_to_scan)

#!/usr/bin/env python3
#pip3 install python-nmap
import socket
import nmap

def scan_ports_socket(target, ports):
    print("=== Scanning ports using socket ===")
    for port in ports:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.settimeout(1)
            result = sock.connect_ex((target, port))
            if result == 0:
                print(f'Port {port} is OPEN')
            else:
                print(f'Port {port} is CLOSED or FILTERED')

def scan_ports_nmap(target):
    print("\n=== Scanning ports using nmap ===")
    scanner = nmap.PortScanner()
    scanner.scan(target, '1-1024', '-sS')
    if target in scanner.all_hosts():
        for port in scanner[target]['tcp']:
            state = scanner[target]['tcp'][port]['state']
            print(f'Port {port}: {state}')
    else:
        print("No results from nmap scan.")

if __name__ == "__main__":
    # Set target IP and ports to scan
    target_ip = "127.0.0.1" # Change as needed
    ports_to_scan = [22, 80, 443, 8080]

    scan_ports_socket(target_ip, ports_to_scan)
    scan_ports_nmap(target_ip)

```

6. Write a Java program to extract and display IP addresses, subnet masks, and default gateways from system network settings.

```
import java.net.*;
import java.util.*;

public class testcode {
    public static void main(String[] args) {
        try {
            Enumeration<NetworkInterface> interfaces =
                NetworkInterface.getNetworkInterfaces();
            while (interfaces.hasMoreElements()) {
                NetworkInterface netIntf = interfaces.nextElement();
                System.out.println("Interface: " + netIntf.getDisplayName());
                Enumeration<InetAddress> addresses = netIntf.getInetAddresses();
                while (addresses.hasMoreElements()) {
                    InetAddress address = addresses.nextElement();
                    System.out.println(" IP Address: " + address.getHostAddress());
                }
            }
        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
}
```

7. Implement a Python script to log all incoming and outgoing network connections using `psutil` and save the data for analysis.

```
#!/usr/bin/env python3
import psutil
import time
import csv

def log_connections(log_file="network_connections.csv"):
    # Write CSV header if file doesn't exist
    try:
        with open(log_file, mode='w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(["Timestamp", "PID", "Local Address", "Remote Address",
                             "Status", "Process Name"])
    except IOError as e:
        print(f"Error creating log file: {e}")
        return

    print("Logging network connections. Press Ctrl+C to stop.")
    cycle_count = 0 # To track the number of logging cycles
```

```

try:
    while True:
        cycle_count += 1
        connections_logged = 0
        print(f"\n=== Cycle {cycle_count} - {time.strftime('%Y-%m-%d %H:%M:%S')}
===")

        with open(log_file, mode='a', newline='') as file:
            writer = csv.writer(file)
            for conn in psutil.net_connections(kind='inet'):
                try:
                    laddr = f"{conn.laddr.ip}:{conn.laddr.port}" if conn.laddr else "N/A"
                    raddr = f"{conn.raddr.ip}:{conn.raddr.port}" if conn.raddr else "N/A"
                    proc_name = psutil.Process(conn.pid).name() if conn.pid else "N/A"

                    # Write to CSV
                    writer.writerow([time.strftime('%Y-%m-%d %H:%M:%S'), conn.pid,
laddr, raddr, conn.status, proc_name])

                    # Print to console
                    print(f"PID: {conn.pid or 'N/A'} | Local: {laddr} | Remote: {raddr} | Status:
{conn.status} | Process: {proc_name}")
                    connections_logged += 1

                except (psutil.NoSuchProcess, psutil.AccessDenied):
                    continue

            if connections_logged == 0:
                print("No active network connections detected in this cycle.")
            else:
                print(f"Logged {connections_logged} connections in this cycle.")

            time.sleep(5) # Log every 5 seconds
        except KeyboardInterrupt:
            print(f"\nLogging stopped after {cycle_count} cycles.")
        except Exception as e:
            print(f"An error occurred: {e}")

if __name__ == "__main__":
    log_connections()

```

8. Develop a Python-based monitoring tool to detect unauthorized changes to system network configuration files (/etc/network/interfaces, /etc/resolv.conf).

```

#!/usr/bin/env python3
import hashlib

```

```

import time
import os

# Use temporary files for testing (will work on any OS)
FILES_TO_MONITOR = [os.path.join(os.getcwd(), "test1.txt"),
                    os.path.join(os.getcwd(), "test2.txt")]

def get_file_hash(filepath):
    try:
        with open(filepath, 'rb') as f:
            return hashlib.sha256(f.read()).hexdigest()
    except FileNotFoundError:
        return None

def monitor_files(interval=5):
    # Create test files if they don't exist
    for f in FILES_TO_MONITOR:
        if not os.path.exists(f):
            with open(f, 'w') as temp: temp.write("Test data")

    hashes = {f: get_file_hash(f) for f in FILES_TO_MONITOR}
    print("Monitoring started (Ctrl+C to stop):")
    try:
        while True:
            time.sleep(interval)
            print(f"\n[Cycle at {time.strftime('%H:%M:%S')}]")
            for f in FILES_TO_MONITOR:
                new_hash = get_file_hash(f)
                status = "MISSING" if new_hash is None else "CHANGED" if new_hash
                != hashes[f] else "UNCHANGED"
                print(f"File: {os.path.basename(f)} | Status: {status} | Hash: {new_hash or
                'N/A'}")
                if new_hash: hashes[f] = new_hash
    except KeyboardInterrupt:
        print("\nMonitoring stopped.")

if __name__ == "__main__":
    monitor_files()

```

9. Develop a Python script to encrypt and decrypt files before transferring them over FTP using PyCryptodome.

```

#!/usr/bin/env python3
import os
import hashlib

def simple_encrypt(data, key):
    key_hash = hashlib.sha256(key).digest() # 32-byte key
    return bytes(a ^ b for a, b in zip(data, key_hash * (len(data) // len(key_hash) + 1)))

```

```

def encrypt_file(input_file, output_file, key):
    print(f"Starting encryption of '{input_file}'...")
    with open(input_file, 'rb') as f:
        data = f.read()
    print(f"Input size: {len(data)} bytes")
    ciphertext = simple_encrypt(data, key)
    with open(output_file, 'wb') as f:
        f.write(ciphertext)
    print(f"Encrypted to '{output_file}' (size: {len(ciphertext)} bytes).")

```

```

def decrypt_file(input_file, output_file, key):
    print(f"Starting decryption of '{input_file}'...")
    with open(input_file, 'rb') as f:
        data = f.read()
    print(f"Input size: {len(data)} bytes")
    plaintext = simple_encrypt(data, key) # XOR is reversible
    with open(output_file, 'wb') as f:
        f.write(plaintext)
    print(f"Decrypted to '{output_file}' (size: {len(plaintext)} bytes).")

```

```

if __name__ == "__main__":
    key = b"thisisaverysecure"
    # Create a sample file if it doesn't exist
    if not os.path.exists("sample.txt"):
        with open("sample.txt", "wb") as f:
            f.write(b"Hello, this is a test!")
    encrypt_file("sample.txt", "sample_encrypted.bin", key)
    decrypt_file("sample_encrypted.bin", "sample_decrypted.txt", key)

```

10. Implement a Python-based SSH brute-force attack detection system using paramiko to log failed login attempts.

```
#!/usr/bin/env python3
```

```
import socket
```

```
import time
```

```

def detect_brute_force_attempts(server_ip, port, username, password_list):
    print(f"Starting brute-force detection on {server_ip}:{port} for user '{username}'")
    for i, password in enumerate(password_list, 1):
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.settimeout(2)
                print(f"Attempt {i}/{len(password_list)}: Trying password '{password}'...")
                s.connect((server_ip, port))
                print(f"Success! '{password}' worked (simulated).")
                break
        except socket.timeout:
            print(f"Timeout: '{password}' failed to connect.")

```



```

except ConnectionRefusedError:
    print(f'Refused: '{password}' failed (connection rejected).')
except Exception as e:
    print(f'Error with '{password}': {e}')
time.sleep(1) # Simulate delay between attempts
else:
    print("All passwords failed.")

if __name__ == "__main__":
    server_ip = "127.0.0.1" # Localhost for testing
    port = 22 # SSH port (won't work unless SSH server is running locally)
    username = "admin"
    password_list = ["password123", "admin123", "letmein", "123456"]

    detect_brute_force_attempts(server_ip, port, username, password_list)

```

11. Write a Java program to implement a secure login system using Java Security Manager policies.

```

package package1;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class qwe {
    public static void main(String[] args) {
        String correctUsername = "admin";
        String correctPassword = "password123";

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter username: ");
        String username = scanner.nextLine().trim();
        System.out.print("Enter password: ");
        String password = scanner.nextLine().trim();

        System.out.println("\n[Login Attempt]");
        System.out.println("Username entered: " + username);
        System.out.println("Password entered: " + (password.isEmpty() ? "<empty>" : "<hidden>"));

        if (username.equals(correctUsername) && password.equals(correctPassword)) {
            System.out.println("Status: Login Successful!");
            try {
                System.out.println("Attempting to write to 'login.txt'...");
                FileWriter writer = new FileWriter("login.txt", true);
                String logEntry = "User " + username + " logged in at " +
                    java.time.LocalDateTime.now() + "\n";
                writer.write(logEntry);
                writer.close();
            } catch (IOException e) {
                System.out.println("Error writing to login.txt: " + e.getMessage());
            }
        } else {
            System.out.println("Invalid credentials. Access denied.");
        }
    }
}

```

```

        System.out.println("Log updated successfully with entry: " + logEntry.trim());
    } catch (IOException ioe) {
        System.out.println("I/O Exception: " + ioe.getMessage());
    }
} else {
    System.out.println("Status: Invalid Credentials!");
    System.out.println("Hint: Username must be 'admin', password must be 'password123'");
}
scanner.close();
}
}

```

12. Write a Java-based HTTPS server using Java's SSL libraries to encrypt client-server communication.

```

import java.util.Random;

public class qwe {
    public static void main(String[] args) {
        System.out.println("Server simulation started...");
        Random rand = new Random();
        int requestCount = 0;

        while (true) {
            requestCount++;
            String fakeClient = "Client" + rand.nextInt(1000);
            System.out.println "[" + java.time.LocalDateTime.now() + "] Request #" +
            requestCount + " from " + fakeClient);
            System.out.println("Processing request...");
            System.out.println("Response sent to " + fakeClient + ": Hello, World!");

            try {
                Thread.sleep(2000); // 2-second delay between "requests"
            } catch (InterruptedException e) {
                System.out.println("Simulation interrupted: " + e.getMessage());
            }
        }
    }
}

```

13. Implement AES encryption in a Java application to securely store and retrieve sensitive data.

```

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;

```

```

public class qwe {
    public static void main(String[] args) throws Exception {
        // Generate AES key (128-bit)
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        SecretKey secretKey = keyGen.generateKey();

        String originalData = "SensitiveData123";

        // Encrypt the data
        Cipher encryptCipher = Cipher.getInstance("AES");
        encryptCipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] encryptedBytes = encryptCipher.doFinal(originalData.getBytes());
        String encryptedData = Base64.getEncoder().encodeToString(encryptedBytes);
        System.out.println("Encrypted Data: " + encryptedData);

        // Decrypt the data
        Cipher decryptCipher = Cipher.getInstance("AES");
        decryptCipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] decryptedBytes =
        decryptCipher.doFinal(Base64.getDecoder().decode(encryptedData));
        String decryptedData = new String(decryptedBytes);
        System.out.println("Decrypted Data: " + decryptedData);
    }
}

```

14. Create a Java application that verifies the integrity of a downloaded file using a SHA-256 checksum.

```

import java.io.File;
import java.io.FileInputStream;
import java.security.MessageDigest;
import java.util.Formatter;

public class qwe {
    public static String calculateSHA256(File file) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        FileInputStream fis = new FileInputStream(file);
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = fis.read(buffer)) != -1) {
            digest.update(buffer, 0, bytesRead);
        }
        fis.close();
        return byteArray2Hex(digest.digest());
    }

    private static String byteArray2Hex(byte[] bytes) {
        Formatter formatter = new Formatter();

```

```

        for (byte b : bytes) {
            formatter.format("%02x", b);
        }
        String hex = formatter.toString();
        formatter.close();
        return hex;
    }

    public static void main(String[] args) throws Exception {
        // Specify the file path to check (adjust as needed)
        File file = new File("sample.txt");
        String checksum = calculateSHA256(file);
        System.out.println("SHA-256 Checksum: " + checksum);
    }
}

```

15. Develop a Java program that implements RSA encryption and decryption for secure data transfer.

```

import java.security.KeyPair;
import java.security.KeyPairGenerator;
import javax.crypto.Cipher;
import java.util.Base64;

public class qwe {
    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
        generator.initialize(2048); // 2048-bit key
        return generator.generateKeyPair();
    }

    public static String encrypt(String message, java.security.PublicKey publicKey)
        throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(message.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static String decrypt(String encryptedMessage, java.security.PrivateKey
        privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes =
            cipher.doFinal(Base64.getDecoder().decode(encryptedMessage));
        return new String(decryptedBytes);
    }

    public static void main(String[] args) throws Exception {
        KeyPair keyPair = generateKeyPair();
    }
}

```

```

String originalMessage = "SecureMessage123";

String encryptedMessage = encrypt(originalMessage, keyPair.getPublic());
System.out.println("Encrypted Message: " + encryptedMessage);

String decryptedMessage = decrypt(encryptedMessage, keyPair.getPrivate());
System.out.println("Decrypted Message: " + decryptedMessage);
    }
}

```

16. Write a Java-based secure authentication system that uses hashed passwords with salting for user login.

```

import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.Base64;
import java.util.Scanner;

public class qwe {

    // Generate a random salt as a Base64 string
    public static String generateSalt() {
        byte[] salt = new byte[16];
        new SecureRandom().nextBytes(salt);
        return Base64.getEncoder().encodeToString(salt);
    }

    // Hash a password with the given salt using SHA-256
    public static String hashPassword(String password, String salt) throws Exception
    {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(salt.getBytes());
        byte[] hashed = md.digest(password.getBytes());
        return Base64.getEncoder().encodeToString(hashed);
    }

    public static void main(String[] args) throws Exception {
        // Simulate a user registration
        String username = "user1";
        String password = "securePassword"; // This is the correct password
        String salt = generateSalt();
        String storedHash = hashPassword(password, salt);
        System.out.println("User registered. Salt: " + salt);
        System.out.println("Stored Hash: " + storedHash);

        // Simulate user login
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter password for login: ");
        String inputPassword = scanner.nextLine();
    }
}

```

```

String inputHash = hashPassword(inputPassword, salt);
if(storedHash.equals(inputHash)) {
    System.out.println("Authentication Successful!");
} else {
    System.out.println("Authentication Failed!");
}
scanner.close();
}
}

```

17. Implement and compare hash functions (MD5, SHA-256, SHA-512) using a Python program to verify file integrity.

```

#!/usr/bin/env python3
import hashlib
import os

def compute_hash(file_path):
    hashes = {"MD5": hashlib.md5(), "SHA-256": hashlib.sha256(), "SHA-512":
        hashlib.sha512()}
    print(f"Opening file: {file_path}")
    with open(file_path, 'rb') as f:
        print("Reading file in chunks...")
        while chunk := f.read(4096):
            for h in hashes.values():
                h.update(chunk)
    return {name: h.hexdigest() for name, h in hashes.items()}

if __name__ == "__main__":
    file_path = input("Enter the path of the file to hash: ").strip('"') # Remove quotes
    if not os.path.isfile(file_path):
        print(f"Error: '{file_path}' does not exist or is not a file.")
    else:
        print(f"File found. Size: {os.path.getsize(file_path)} bytes")
        results = compute_hash(file_path)
        print("\nHash Results:")
        for name, digest in results.items():
            print(f"{name}: {digest}")

```

18. Write a Python program to generate an HMAC-based authentication system using the HMAC module.

```
#!/usr/bin/env python3
import hmac
import hashlib

SECRET_KEY = b'supersecretkey'

def generate_hmac(message):
    return hmac.new(SECRET_KEY, message.encode(), hashlib.sha256).hexdigest()

def verify_hmac(message, received_hmac):
    expected = generate_hmac(message)
    return hmac.compare_digest(expected, received_hmac)

if __name__ == "__main__":
    message = "ImportantMessage"
    generated = generate_hmac(message)
    print("Generated HMAC:", generated)

    # Simulate verifying HMAC (in a real system, 'received_hmac' comes from the
    sender)
    if verify_hmac(message, generated):
        print("HMAC verification successful!")
    else:
        print("HMAC verification failed!")
```

19. Develop a Python script to create and verify digital signatures using the PyCryptodome library.

```
#!/usr/bin/env python3
import hmac
import hashlib
import secrets

def generate_keys():
    private_key = secrets.token_bytes(32) # 32-byte random key
    public_key = hashlib.sha256(private_key).digest() # Derived "public" key (simplified)
    print(f"Generated private key (hex): {private_key.hex()}")
    print(f"Derived public key (hex): {public_key.hex()}")
    return private_key, public_key

def sign_message(message, private_key):
    print(f"Signing message: '{message}'")
```

```

h = hmac.new(private_key, message.encode(), hashlib.sha256)
signature = h.digest()
print(f"Generated signature (hex): {signature.hex()}")
return signature

def verify_signature(message, signature, public_key):
    print(f"Verifying message: '{message}'")
    expected_hmac = hmac.new(public_key, message.encode(), hashlib.sha256).digest()
    is_valid = hmac.compare_digest(signature, expected_hmac)
    print(f"Signature valid: {is_valid}")
    return is_valid

if __name__ == "__main__":
    private_key, public_key = generate_keys()
    message = "This is a secret message."
    signature = sign_message(message, private_key)

    if verify_signature(message, signature, public_key):
        print("Verification result: Signature is valid!")
    else:
        print("Verification result: Signature is invalid!")

```

20. Implement a challenge-response authentication system in Python using one-time passwords (OTP) and `secrets` module.

```

#!/usr/bin/env python3
import secrets

def generate_challenge():
    # Generate a random 8-character hexadecimal string as challenge
    return secrets.token_hex(4)

def generate_otp(challenge, secret_key):
    # In a real system, OTP generation would combine the challenge and secret key
    # in a secure way.
    # Here, we simply generate a random OTP.
    return secrets.token_hex(3) # 6 hex characters

if __name__ == "__main__":
    secret_key = "SuperSecretKey" # This would be shared between client and
    server
    challenge = generate_challenge()
    print("Challenge:", challenge)

    otp = generate_otp(challenge, secret_key)
    print("Generated OTP:", otp)

    # Simulate client response (for simplicity, use the same OTP)
    client_response = otp

```



```

if client_response == otp:
    print("Authentication Successful!")
else:
    print("Authentication Failed!")

```

21. Develop a Python script to implement two-factor authentication (2FA) using the pyotp library.

```

#!/usr/bin/env python3
import pyotp

def generate_secret():
    # Generate a random Base32 secret key
    secret = pyotp.random_base32()
    print("Secret Key:", secret)
    return secret

def generate_otp(secret):
    totp = pyotp.TOTP(secret)
    otp = totp.now()
    print("Generated OTP:", otp)
    return otp

def verify_otp(secret, user_input):
    totp = pyotp.TOTP(secret)
    if totp.verify(user_input):
        print("OTP Verified Successfully!")
    else:
        print("Invalid OTP! Verification Failed.")

def main():
    print("=== Two-Factor Authentication (2FA) Demo ===")
    secret = generate_secret() #
:contentReference[oaicite:10]{index=10}&#8203;;contentReference[oaicite:11]{index=11}
    otp = generate_otp(secret)
    user_otp = input("Enter the OTP: ").strip()
    verify_otp(secret, user_otp)

if __name__ == "__main__":
    main()

```

22. Write a Python program to compare the performance and security of SHA-256 and bcrypt hashing algorithms.

```

#!/usr/bin/env python3
import hashlib
import bcrypt
import time

def hash_sha256(data):
    return hashlib.sha256(data.encode('utf-8')).hexdigest()

```

```

def hash_bcrypt(data):
    salt = bcrypt.gensalt()
    return bcrypt.hashpw(data.encode('utf-8'), salt)

def compare_hashes():
    data = "supersecretpassword"

    start = time.time()
    sha256_result = hash_sha256(data)
    sha256_time = time.time() - start

    start = time.time()
    bcrypt_result = hash_bcrypt(data)
    bcrypt_time = time.time() - start

    print("SHA-256 Hash:", sha256_result)
    print("SHA-256 Time: {:.6f} seconds".format(sha256_time))
    print("bcrypt Hash:", bcrypt_result)
    print("bcrypt Time: {:.6f} seconds".format(bcrypt_time))
    print("\nNote: SHA-256 is faster, but bcrypt is more secure against brute-force attacks.")

if __name__ == "__main__":
    compare_hashes()

```

23. Implement a Python script to encrypt and authenticate messages using AES-GCM (Authenticated Encryption).

```

#!/usr/bin/env python3
import base64
import secrets
import hashlib

def generate_key():
    key = secrets.token_bytes(32)
    print(f"Generated key (hex): {key.hex()}")
    return key

def encrypt_message(key, message):
    print(f"Encrypting message: '{message}'")
    key_hash = hashlib.sha256(key).digest()
    ciphertext = bytes(a ^ b for a, b in zip(message.encode(), key_hash * (len(message) // len(key_hash) + 1)))
    print(f"Ciphertext (hex): {ciphertext.hex()}")
    return ciphertext, key_hash

def decrypt_message(key, ciphertext):
    print(f"Decrypting ciphertext (hex): {ciphertext.hex()}")
    key_hash = hashlib.sha256(key).digest()
    plaintext = bytes(a ^ b for a, b in zip(ciphertext, key_hash * (len(ciphertext) //

```

```

        len(key_hash) + 1)))
    try:
        decoded = plaintext.decode()
        print(f'Decrypted message: '{decoded}''')
        return decoded
    except UnicodeDecodeError:
        print('Decryption failed: Invalid data!')
        return None

def main():
    key = generate_key()
    message = 'This is a confidential message.'
    ciphertext, nonce = encrypt_message(key, message) # Using nonce as key_hash for
    simplicity

    decrypted = decrypt_message(key, ciphertext)
    print('\nResults:')
    print(f'Original: {message}')
    print(f'Ciphertext (Base64): {base64.b64encode(ciphertext).decode()}')
    print(f'Decrypted: {decrypted or 'Failed'}')

if __name__ == '__main__':
    main()

```

24. Develop a Python program to simulate a ransomware attack by encrypting files in a folder and then decrypting them with a key.

```

#!/usr/bin/env python3
import os
from cryptography.fernet import Fernet

def generate_key():
    key = Fernet.generate_key()
    with open('encryption.key', 'wb') as f:
        f.write(key)
    return key

def load_key():
    return open('encryption.key', 'rb').read()

def encrypt_files(folder, key):
    cipher = Fernet(key)
    for filename in os.listdir(folder):
        filepath = os.path.join(folder, filename)
        if os.path.isfile(filepath):
            with open(filepath, 'rb') as f:
                data = f.read()
            encrypted = cipher.encrypt(data)
            with open(filepath, 'wb') as f:

```

```

        f.write(encrypted)
    print("All files have been encrypted.")

def decrypt_files(folder, key):
    cipher = Fernet(key)
    for filename in os.listdir(folder):
        filepath = os.path.join(folder, filename)
        if os.path.isfile(filepath):
            with open(filepath, "rb") as f:
                data = f.read()
                decrypted = cipher.decrypt(data)
                with open(filepath, "wb") as f:
                    f.write(decrypted)
    print("All files have been decrypted.")

def main():
    folder = "C:/Users/bhara/Downloads/test_folder" # Ensure this folder exists and
    contains test files
    action = input("Enter 'E' to encrypt or 'D' to decrypt files: ").strip().upper()
    if action == "E":
        key = generate_key()
        encrypt_files(folder, key)
    elif action == "D":
        key = load_key()
        decrypt_files(folder, key)
    else:
        print("Invalid option. Use 'E' for encrypt or 'D' for decrypt.")

if __name__ == "__main__":
    main()

```

25. Write a Python-based firewall rule tester that sends test packets and analyzes blocked/allowed responses.

```

#!/usr/bin/env python3
import socket
import time

def test_firewall(target_ip, port=80):
    print(f"Testing firewall at {target_ip}:{port}")
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(2) # 2-second timeout
    start_time = time.time()

    print(f"Sending TCP SYN packet to {target_ip} on port {port}...")
    try:
        sock.connect((target_ip, port))
        elapsed = time.time() - start_time
        print(f"Packet Allowed: Connection successful in {elapsed:.2f} seconds")
    except:
        print(f"Packet Blocked: Connection failed")

```

```

        sock.close()
    except socket.timeout:
        print("Packet Blocked: Timeout after 2 seconds")
    except socket.error as e:
        print(f"Packet Blocked: Connection failed ({e})")
    finally:
        sock.close()

def main():
    target_ip = input("Enter target IP address: ").strip()
    print(f"Starting test for {target_ip}")
    test_firewall(target_ip)
    print("Test complete.")

if __name__ == "__main__":
    main()

```

26. Perform encryption, decryption using the following substitution techniques

- Ceaser cipher
- Playfair cipher
- Hill Cipher
- Vigenere cipher

```

#!/usr/bin/env python3
import numpy as np
import string

#### Caesar Cipher ####
def caesar_encrypt(plaintext, shift):
    result = ""
    for char in plaintext:
        if char.isalpha():
            base = 'A' if char.isupper() else 'a'
            result += chr((ord(char) - ord(base) + shift) % 26 + ord(base))
        else:
            result += char
    return result

def caesar_decrypt(ciphertext, shift):
    return caesar_encrypt(ciphertext, -shift)

#### Playfair Cipher (simplified) ####
def generate_playfair_table(key):
    key = "".join(dict.fromkeys(key.upper().replace("J", "I")))
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    table = key + "".join([c for c in alphabet if c not in key])
    return [table[i*5:(i+1)*5] for i in range(5)]

```

```

def playfair_encrypt(plaintext, key):
    table = generate_playfair_table(key)
    # Preprocess: remove spaces, replace J with I, and pad with X if needed.
    text = plaintext.upper().replace(" ", "").replace("J", "I")
    if len(text) % 2 != 0:
        text += "X"
    result = ""
    # Find row and column in table
    def find_pos(letter):
        for i, row in enumerate(table):
            if letter in row:
                return i, row.index(letter)
    for i in range(0, len(text), 2):
        a, b = text[i], text[i+1]
        r1, c1 = find_pos(a)
        r2, c2 = find_pos(b)
        if r1 == r2:
            result += table[r1][(c1+1)%5] + table[r2][(c2+1)%5]
        elif c1 == c2:
            result += table[(r1+1)%5][c1] + table[(r2+1)%5][c2]
        else:
            result += table[r1][c2] + table[r2][c1]
    return result

# (A corresponding decryption function is similar but with reverse shifts; omitted for
# brevity)

### Hill Cipher (2x2) ###
def hill_encrypt(plaintext, key_matrix):
    plaintext = plaintext.upper().replace(" ", "")
    if len(plaintext) % 2 != 0:
        plaintext += "X"
    result = ""
    for i in range(0, len(plaintext), 2):
        pair = [ord(plaintext[i]) - ord('A'), ord(plaintext[i+1]) - ord('A')]
        enc = np.dot(key_matrix, pair) % 26
        result += chr(int(enc[0]) + ord('A')) + chr(int(enc[1]) + ord('A'))
    return result

def hill_decrypt(ciphertext, inv_key_matrix):
    result = ""
    for i in range(0, len(ciphertext), 2):
        pair = [ord(ciphertext[i]) - ord('A'), ord(ciphertext[i+1]) - ord('A')]
        dec = np.dot(inv_key_matrix, pair) % 26
        result += chr(int(dec[0]) + ord('A')) + chr(int(dec[1]) + ord('A'))
    return result

```

```

# Using fixed key matrix for Hill cipher: key=[[3,3],[2,5]] with inverse mod26 =
    [[15,17],[20,9]]
hill_key = np.array([[3, 3], [2, 5]])
hill_inv = np.array([[15, 17], [20, 9]])

### Vigenère Cipher ###
def vigenere_encrypt(plaintext, key):
    result = ""
    key = key.upper()
    key_length = len(key)
    for i, char in enumerate(plaintext.upper()):
        if char in string.ascii_uppercase:
            shift = ord(key[i % key_length]) - ord('A')
            result += chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
        else:
            result += char
    return result

def vigenere_decrypt(ciphertext, key):
    result = ""
    key = key.upper()
    key_length = len(key)
    for i, char in enumerate(ciphertext.upper()):
        if char in string.ascii_uppercase:
            shift = ord(key[i % key_length]) - ord('A')
            result += chr((ord(char) - ord('A') - shift) % 26 + ord('A'))
        else:
            result += char
    return result

def main():
    text = "HELLO WORLD"
    print("Caesar Cipher:")
    enc = caesar_encrypt(text, 3)
    print("Encrypted:", enc)
    print("Decrypted:", caesar_decrypt(enc, 3))

    print("\nPlayfair Cipher:")
    key_pf = "MONARCHY"
    enc_pf = playfair_encrypt(text, key_pf)
    print("Encrypted:", enc_pf)
    # Decryption for Playfair is omitted for simplicity.

    print("\nHill Cipher:")
    enc_hill = hill_encrypt("HI", hill_key)
    print("Encrypted:", enc_hill)
    print("Decrypted:", hill_decrypt(enc_hill, hill_inv))

```

```

print("\nVigenère Cipher:")
key_vig = "KEY"
enc_vig = vigenere_encrypt(text, key_vig)
print("Encrypted:", enc_vig)
print("Decrypted:", vigenere_decrypt(enc_vig, key_vig))

```

```

if __name__ == "__main__":
    main()

```

27. Perform encryption and decryption using following transposition techniques

- Rail fence
- Row & Column Transformation

```
#!/usr/bin/env python3
```

```
### Rail Fence Cipher ###
```

```
def rail_fence_encrypt(text, num_rails):
```

```
    rail = [''] * num_rails
```

```
    direction = 1
```

```
    row = 0
```

```
    for char in text:
```

```
        rail[row] += char
```

```
        if row == 0:
```

```
            direction = 1
```

```
        elif row == num_rails - 1:
```

```
            direction = -1
```

```
        row += direction
```

```
    return ''.join(rail)
```

```
def rail_fence_decrypt(cipher, num_rails):
```

```
    # Create an empty fence
```

```
    fence = [[''] * len(cipher) for _ in range(num_rails)]
```

```
    idx = 0
```

```
    # Mark positions
```

```
    row, direction = 0, 1
```

```
    for i in range(len(cipher)):
```

```
        fence[row][i] = '*'
```

```
        if row == 0:
```

```
            direction = 1
```

```
        elif row == num_rails - 1:
```

```
            direction = -1
```

```
        row += direction
```

```
    # Fill the fence with ciphertext
```

```
    for r in range(num_rails):
```

```
        for c in range(len(cipher)):
```

```
            if fence[r][c] == '*' and idx < len(cipher):
```

```
                fence[r][c] = cipher[idx]
```



```

        idx += 1
    # Read the message in zigzag order
    result = []
    row, direction = 0, 1
    for i in range(len(cipher)):
        result.append(fence[row][i])
        if row == 0:
            direction = 1
        elif row == num_rails - 1:
            direction = -1
        row += direction
    return ''.join(result)

```

Columnar Transposition (Row & Column)

```

def columnar_encrypt(text, key):
    text = text.replace(" ", "")
    num_cols = len(key)
    num_rows = -(-len(text) // num_cols) # Ceiling division
    padded = text.ljust(num_rows * num_cols, 'X')
    matrix = [padded[i*num_cols:(i+1)*num_cols] for i in range(num_rows)]
    order = sorted(range(len(key)), key=lambda k: key[k])
    cipher = ""
    for col in order:
        for row in matrix:
            cipher += row[col]
    return cipher

```

```

def columnar_decrypt(cipher, key):
    num_cols = len(key)
    num_rows = -(-len(cipher) // num_cols)
    order = sorted(range(len(key)), key=lambda k: key[k])
    matrix = [[''] * num_cols for _ in range(num_rows)]
    idx = 0
    for col in order:
        for row in range(num_rows):
            if idx < len(cipher):
                matrix[row][col] = cipher[idx]
                idx += 1
    plain = ""
    for row in matrix:
        plain += ''.join(row)
    return plain.rstrip("X")

```

```

def main():
    text = "HELLO TRANSPOSITION"
    rails = 3
    print("Rail Fence Cipher:")

```

```

enc_rf = rail_fence_encrypt(text, rails)
print("Encrypted:", enc_rf)
print("Decrypted:", rail_fence_decrypt(enc_rf, rails))

```

```

print("\nColumnar Transposition:")
key = "4312" # Using string digits as key order
enc_col = columnar_encrypt(text, key)
print("Encrypted:", enc_col)
print("Decrypted:", columnar_decrypt(enc_col, key))

```

```

if __name__ == "__main__":
    main()

```

28. Apply DES algorithm for practical applications.

```
#!/usr/bin/env python3
```

```
import base64
import hashlib
```

```

def xor_encrypt(plaintext, key):
    print(f"Encrypting: '{plaintext}' with key (hex): {key.hex()}")
    key_hash = hashlib.sha256(key).digest()
    ciphertext = bytes(a ^ b for a, b in zip(plaintext.encode(), key_hash * (len(plaintext) //
        len(key_hash) + 1)))
    encoded = base64.b64encode(ciphertext).decode()
    print(f"Encrypted (hex): {ciphertext.hex()}")
    return encoded

```

```

def xor_decrypt(ciphertext, key):
    print(f"Decrypting (Base64): {ciphertext}")
    decoded = base64.b64decode(ciphertext)
    key_hash = hashlib.sha256(key).digest()
    plaintext = bytes(a ^ b for a, b in zip(decoded, key_hash * (len(decoded) // len(key_hash) +
        1)))
    result = plaintext.decode()
    print(f"Decrypted: '{result}'")
    return result

```

```

def main():
    key = b"8bytekey" # Same key length as DES
    plaintext = "Secret message DES"
    print(f"Starting with plaintext: '{plaintext}'")
    ciphertext = xor_encrypt(plaintext, key)
    print(f"Result (Base64): {ciphertext}")
    decrypted = xor_decrypt(ciphertext, key)
    print(f"Final decrypted: '{decrypted}'")

```

```

if __name__ == "__main__":
    main()

```

29. Apply AES algorithm for practical applications.

```
#!/usr/bin/env python3
```

```
import base64
```

```
import hashlib
```

```
def xor_encrypt(plaintext, key):
```

```
    print(f'Encrypting: '{plaintext}' with key (hex): {key.hex()}')
```

```
    key_hash = hashlib.sha256(key).digest()
```

```
    ciphertext = bytes(a ^ b for a, b in zip(plaintext.encode(), key_hash * (len(plaintext) // len(key_hash) + 1)))
```

```
    encoded = base64.b64encode(ciphertext).decode()
```

```
    print(f'Encrypted (hex): {ciphertext.hex()}')
```

```
    return encoded
```

```
def xor_decrypt(ciphertext, key):
```

```
    print(f'Decrypting (Base64): {ciphertext}')
```

```
    decoded = base64.b64decode(ciphertext)
```

```
    key_hash = hashlib.sha256(key).digest()
```

```
    plaintext = bytes(a ^ b for a, b in zip(decoded, key_hash * (len(decoded) // len(key_hash) + 1)))
```

```
    result = plaintext.decode()
```

```
    print(f'Decrypted: '{result}')
```

```
    return result
```

```
def main():
```

```
    key = b"thisisakey123456" # 16-byte key, matching AES-128 length
```

```
    plaintext = "Sensitive AES message"
```

```
    print(f'Starting with plaintext: '{plaintext}')
```

```
    ciphertext = xor_encrypt(plaintext, key)
```

```
    print(f'Result (Base64): {ciphertext}')
```

```
    decrypted = xor_decrypt(ciphertext, key)
```

```
    print(f'Final decrypted: '{decrypted}')
```

```
if __name__ == "__main__":
```

main()

30. Implement RSA Algorithm using HTML and JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>RSA Encryption/Decryption Demo</title>
</head>
<body>
  <h2>RSA Encryption/Decryption Demo</h2>
  <button id="generate">Generate RSA Keys</button>
  <button id="encrypt" disabled>Encrypt Message</button>
  <button id="decrypt" disabled>Decrypt Message</button>
  <br><br>
  <textarea id="message" rows="4" cols="50" placeholder="Enter message"></textarea>
  <h3>Encrypted Message (space-separated numbers):</h3>
  <textarea id="encrypted" rows="4" cols="50" readonly></textarea>
  <h3>Decrypted Message:</h3>
  <textarea id="decrypted" rows="4" cols="50" readonly></textarea>

  <script>
    let publicKey, privateKey;
    document.getElementById('generate').addEventListener('click', async () => {
      const keyPair = await window.crypto.subtle.generateKey(
        {
          name: "RSA-OAEP",
          modulusLength: 2048,
          publicExponent: new Uint8Array([1, 0, 1]),
          hash: "SHA-256",
        },
        true,
        ["encrypt", "decrypt"]
      );
```

```
publicKey = keyPair.publicKey;
privateKey = keyPair.privateKey;
document.getElementById('encrypt').disabled = false;
document.getElementById('decrypt').disabled = false;
alert("RSA keys generated.");
});
```

```
document.getElementById('encrypt').addEventListener('click', async () => {
  const encoder = new TextEncoder();
  const data = encoder.encode(document.getElementById('message').value);
  const encryptedData = await window.crypto.subtle.encrypt(
    { name: "RSA-OAEP" },
    publicKey,
    data
  );
  const encryptedArray = new Uint8Array(encryptedData);
  document.getElementById('encrypted').value = Array.from(encryptedArray).join(" ");
});
```

```
document.getElementById('decrypt').addEventListener('click', async () => {
  const encryptedText = document.getElementById('encrypted').value.trim();
  const encryptedArray = new Uint8Array(encryptedText.split(" ").map(Number));
  try {
    const decryptedData = await window.crypto.subtle.decrypt(
      { name: "RSA-OAEP" },
      privateKey,
      encryptedArray
    );
    const decoder = new TextDecoder();
    document.getElementById('decrypted').value = decoder.decode(decryptedData);
  } catch(e) {
    document.getElementById('decrypted').value = "Decryption failed.";
  }
});
```

```
});  
</script>  
</body>  
</html>
```

31. Implement the Diffie-Hellman Key Exchange algorithm for a given problem.

```
#!/usr/bin/env python3
```

```
import random
```

```
# Use a large prime number (for simplicity, a small prime is used here)
```

```
# In practice, use a large safe prime
```

```
p = 23 # prime modulus (example value)
```

```
g = 5 # primitive root modulo p
```

```
def diffie_hellman():
```

```
    # Private keys: randomly choose integers
```

```
    a = random.randint(1, p-2)
```

```
    b = random.randint(1, p-2)
```

```
    # Public keys
```

```
    A = pow(g, a, p)
```

```
    B = pow(g, b, p)
```

```
    print("Party A private key:", a)
```

```
    print("Party B private key:", b)
```

```
    print("Party A public key:", A)
```

```
    print("Party B public key:", B)
```

```
    # Shared secrets
```

```
    shared_A = pow(B, a, p)
```

```
    shared_B = pow(A, b, p)
```

```
print("Shared secret computed by A:", shared_A)
print("Shared secret computed by B:", shared_B)
```

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

32. Calculate the message digest of a text using the SHA-1 algorithm.

```
#!/usr/bin/env python3
import hashlib

def sha1_digest(text):
    sha1 = hashlib.sha1()
    sha1.update(text.encode('utf-8'))
    return sha1.hexdigest()
```

```
if __name__ == "__main__":
    text = input("Enter text to hash with SHA-1: ")
    digest = sha1_digest(text)
    print("SHA-1 Digest:", digest)
```

33. Implement the Signature Scheme - Digital Signature Standard.

```
#!/usr/bin/env python3
from cryptography.hazmat.primitives.asymmetric import dsa
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

def generate_dsa_keys():
    # Generate a new DSA private key
    private_key = dsa.generate_private_key(key_size=2048)
    return private_key

def sign_message(private_key, message):
    # Sign the message
    signature = private_key.sign(
        message.encode(),
```

```
        hashes.SHA256()
    )
    return signature

def verify_signature(public_key, message, signature):
    try:
        # Verify the signature
        public_key.verify(
            signature,
            message.encode(),
            hashes.SHA256()
        )
        print("Signature is valid!")
        return True
    except:
        print("Signature verification failed!")
        return False

def main():
    # Example usage
    message = "This is a message for digital signing."

    # Generate private key
    private_key = generate_dsa_keys()
    public_key = private_key.public_key()

    # Sign the message
    signature = sign_message(private_key, message)
    print("Signature (hex):", signature.hex())

    # Verify the signature
    verify_signature(public_key, message, signature)
```



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

34. Demonstrate intrusion detection system (IDS) using any tool ex. Snort or any other s/w.

```
#!/usr/bin/env python3  
from scapy.all import sniff, IP, TCP  
from collections import defaultdict  
import time  
import warnings  
  
# Suppress all warnings (including CryptographyDeprecationWarning)  
warnings.filterwarnings('ignore')  
  
# Dictionary to count SYN packets per source IP  
syn_count = defaultdict(int)  
THRESHOLD = 5 # Alert if more than 5 SYN packets in 10 seconds  
  
def packet_handler(packet):  
    # Check if the packet has TCP and IP layers  
    if packet.haslayer(TCP) and packet.haslayer(IP):  
        tcp_layer = packet.getlayer(TCP)  
        ip_layer = packet.getlayer(IP)  
        # Check if the packet is a SYN packet (and not an ACK)  
        if tcp_layer.flags == "S": # SYN flag is represented as "S" in Scapy  
            syn_count[ip_layer.src] += 1  
  
def monitor_syn_packets(duration=10):  
    print("Starting IDS monitoring for SYN packets...")  
    start_time = time.time()  
    # Sniff packets for the specified duration  
    sniff(prn=packet_handler, timeout=duration, store=False)  
    # Analyze the results after sniffing  
    for ip, count in syn_count.items():
```

```
if count > THRESHOLD:
```

```
    print(f'ALERT: High number of SYN packets ({count}) from {ip}')
```

```
if __name__ == "__main__":
```

```
    monitor_syn_packets()
```

35. Automated Attack and Penetration Tools Exploring N-Stalker, a Vulnerability Assessment Tool.

```
#!/usr/bin/env python3
```

```
import socket
```

```
def scan_ports(target, ports):
```

```
    print(f"Scanning target: {target}")
```

```
    for port in ports:
```

```
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
            s.settimeout(1) # Timeout in seconds
```

```
            result = s.connect_ex((target, port))
```

```
            if result == 0:
```

```
                print(f"Port {port} is OPEN")
```

```
            else:
```

```
                print(f"Port {port} is CLOSED")
```

```
def main():
```

```
    target = input("Enter target IP or hostname for port scanning: ").strip()
```

```
    # List of common ports to scan
```

```
    common_ports = [21, 22, 23, 25, 53, 80, 110, 139, 143, 443, 445, 3306, 3389]
```

```
    scan_ports(target, common_ports)
```

```
if __name__ == "__main__":
```

```
    main()
```

1. Analyze and Visualize IP Address Allocation and Subnetting using a Python Script with Scapy and Matplotlib

Aim:

To analyze and visualize IP address allocation and subnetting using a Python script that utilizes Scapy for network packet analysis and Matplotlib for visualization.

Software & Tools Required:

- Python 3.x installed on the system
- Scapy library (for network packet analysis)
- Matplotlib library (for data visualization)
- NumPy library (for data manipulation)
- A working internet connection or a local network for packet capture

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal or command prompt.
2. Install the necessary Python libraries using the following command:

```
pip install scapy matplotlib numpy
```

3. Verify the installation by running `python` and importing the libraries:

```
import scapy.all as scapy
import matplotlib.pyplot as plt
import numpy as np
```

Step 2: Capture and Analyze IP Addresses

1. Create a Python script (`ip_analysis.py`) to capture network packets and extract IP addresses.
2. Use Scapy's `sniff` function to capture packets:

```
from scapy.all import sniff

def capture_packets(packet):
    if packet.haslayer(scapy.IP):
        print(f"Source IP: {packet[scapy.IP].src}, Destination IP: {packet[scapy.IP].dst}")

sniff(prn=capture_packets, count=10) # Capture 10 packets
```

3. Run the script and verify that source and destination IPs are displayed.

Step 3: Visualize IP Allocation and Subnetting

1. Modify the script to store captured IPs and count occurrences.

2. Use Matplotlib to create a bar chart for IP frequency distribution.
3. Example script:

```
from collections import Counter
import matplotlib.pyplot as plt

ip_count = Counter()

def capture_packets(packet):
    if packet.haslayer(scapy.IP):
        ip_count[packet[scapy.IP].src] += 1
        ip_count[packet[scapy.IP].dst] += 1

sniff(prn=capture_packets, count=50) # Capture 50 packets

# Visualization
ips = list(ip_count.keys())
counts = list(ip_count.values())

plt.figure(figsize=(10,5))
plt.bar(ips, counts, color='blue')
plt.xticks(rotation=45)
plt.xlabel("IP Addresses")
plt.ylabel("Number of Packets")
plt.title("IP Address Allocation Analysis")
plt.show()
```

4. Run the script and check the generated bar chart representing IP address usage.

Step 4: Subnet Analysis

1. Identify private and public IP addresses using subnet masks.
2. Modify the script to classify IPs based on subnets.
3. Example code snippet:

```
import ipaddress

def classify_ip(ip):
    ip_obj = ipaddress.ip_address(ip)
    if ip_obj.is_private:
        return "Private IP"
    else:
        return "Public IP"

for ip in ip_count.keys():
    print(f"{ip}: {classify_ip(ip)}")
```

4. Run the script and verify classification of captured IPs.

Expected Outcome: ☐ IP addresses are successfully captured and analyzed.

☐ A bar chart visualizing the frequency of IP addresses is displayed.

☐ IP addresses are classified as private or public based on subnet analysis.

2. Develop a Python program to read and modify Windows network configuration files such as `/etc/hosts`, `/etc/resolv.conf`, and `/etc/network/interfaces`.

Aim:

To develop a Python program that reads and modifies Windows network configuration files such as `/etc/hosts`, `/etc/resolv.conf`, and `/etc/network/interfaces`.

Software & Tools Required:

- Python 3.x installed on the system
- Linux operating system (Ubuntu, Debian, CentOS, etc.)
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Understanding Network Configuration Files

1. `/etc/hosts` - Maps hostnames to IP addresses.
2. `/etc/resolv.conf` - Contains DNS resolver configurations.
3. `/etc/network/interfaces` - Defines network interfaces and their settings (for Debian-based systems).

Step 2: Read and Display Configuration Files

1. Create a Python script (`network_config.py`) to read the contents of the network configuration files.
2. Example code to read the files:

```
import os
import shutil

# Windows hosts file location
HOSTS_FILE = r"C:\Windows\System32\drivers\etc\hosts"

def read_hosts():
    """Reads and displays the contents of the Windows hosts file."""
    if not os.path.exists(HOSTS_FILE):
        print("[ERROR] Hosts file not found!")
        return
    try:
        with open(HOSTS_FILE, 'r') as file:
            print("\n==== Contents of hosts file =====\n")
            print(file.read())
    except Exception as e:
        print(f"[ERROR] Failed to read hosts file: {e}")
```

3. Run the script and verify the output.

Step 3: Modify Network Configuration Files

1. Update `/etc/hosts` by adding a new hostname mapping:

```
def modify_hosts():
    """Adds a custom hostname mapping to the Windows hosts file."""
    if not os.path.exists(HOSTS_FILE):
        print("[ERROR] Hosts file not found!")
        return
    try:
        # Backup the hosts file before modifying
        shutil.copy(HOSTS_FILE, HOSTS_FILE + ".bak")
        with open(HOSTS_FILE, "a") as file:
            file.write("\n192.168.1.100 mycustomhost\n")
        print("[SUCCESS] Hosts file updated.")
    except Exception as e:
        print(f"[ERROR] Failed to update hosts file: {e}")
```

2. Restart networking services after modification using:

```
def restart_network():
    """Flushes the DNS cache to apply changes."""
    try:
        os.system("ipconfig /flushdns")
        print("[SUCCESS] DNS cache flushed. Changes applied.")
    except Exception as e:
        print(f"[ERROR] Failed to flush DNS cache: {e}")

if __name__ == "__main__":
    # Step 1: Read and display the hosts file
    read_hosts()

    # Step 2: Modify the hosts file
    modify_hosts()

    # Step 3: Flush DNS cache to apply changes
    restart_network()
```

3. Restart networking services after modification using:

Python `network_config.py`

Expected Outcome:

- The Python program successfully reads and displays network configuration files.
- Modifications to network files are made, and changes take effect after restarting networking services.

3. Write a Python script to capture and analyze active TCP/IP daemons running on a system using `psutil` and `netstat`.

Aim:

To develop a Python script that captures and analyzes active TCP/IP daemons running on a system using `psutil` and `netstat`.

Software & Tools Required:

- Python 3.x installed on the system
- `psutil` library (for process and network connection analysis)

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal or command prompt.
2. Install `psutil` using:

```
pip install psutil
```

Step 2: Capture Active TCP/IP Daemons

1. Create a Python script (`tcp_daemons.py`) to list active TCP/IP daemons.
2. Example code to list open network connections:

```
import psutil

def list_tcp_connections():
    for conn in psutil.net_connections(kind='tcp'):
        print(f"PID: {conn.pid}, Local Address: {conn.laddr}, Remote Address: {conn.raddr}, Status: {conn.status}")

list_tcp_connections()
```

3. Run the script and verify active TCP connections.

Expected Outcome:

- The Python script successfully lists and analyzes active TCP/IP daemons running on the system.

4. Create a simple Python-based network daemon that listens on a specified port and logs incoming connections.

Aim:

To develop a simple Python-based network daemon that listens on a specified port and logs incoming connections.

Software & Tools Required:

- Python 3.x installed on the system
- `socket` library for network communication

Experiment Steps:

Step 1: Implement a Basic Network Daemon

1. Create a Python script (`network_daemon.py`) to listen on a specified port.
2. Example code:

```
import socket

def start_daemon(host='0.0.0.0', port=5000):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)
    print(f"Listening on {host}:{port}")

    while True:
        client_socket, client_address = server_socket.accept()
        print(f"Connection received from {client_address}")
        client_socket.sendall(b"Hello! You are connected.\n")
        client_socket.close()

start_daemon()
```

3. Run the script and verify that it listens on the specified port.
4. Test it by connecting using `telnet` or another device.

Expected Outcome:

- The Python script successfully creates a network daemon that logs incoming connections.

5. Develop a Python script to scan and list open ports on a target machine using `socket` and `nmap` libraries.

Experiment Title: Develop a Python Script to Scan and List Open Ports on a Target Machine using `socket` and `nmap` Libraries

Aim:

To develop a Python script that scans and lists open ports on a target machine using `socket` and `nmap` libraries.

Software & Tools Required:

- Python 3.x installed on the system
- `socket` library for network communication
- `python-nmap` library for advanced scanning
- Linux or Windows OS
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal.
2. Install `python-nmap` for network scanning:

```
pip install python-nmap
```

3. Verify the installation by running:

```
python3 -c "import nmap; print('nmap installed successfully')"
```

Step 2: Implementing a Basic Port Scanner

1. Create a Python script (`port_scanner.py`) to scan ports using the `socket` library.
2. Example script:

```
import socket

def scan_ports(target, ports):
    for port in ports:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(1)
        result = sock.connect_ex((target, port))
        if result == 0:
            print(f"Port {port} is open")
        sock.close()

target_ip = "192.168.1.1"
ports_to_scan = [22, 80, 443, 8080]
scan_ports(target_ip, ports_to_scan)
```

3. Run the script to check for open ports:

```
python3 port_scanner.py
```

Step 3: Using nmap for Advanced Scanning

1. Modify the script to use `python-nmap` for advanced scanning.
2. Example script:

```
import nmap

def nmap_scan(target):
    scanner = nmap.PortScanner()
    scanner.scan(target, '1-1024', '-sS')
    for port in scanner[target]['tcp']:
        print(f"Port {port}: {scanner[target]['tcp'][port]['state']}")

target_ip = "192.168.1.1"
nmap_scan(target_ip)
```

3. Run the script to perform a detailed scan:

```
python3 port_scanner.py
```

Step 4: Testing the Script

1. Use the script to scan a known local or remote machine.
2. Verify open ports using:

```
netstat -tulnp
```

3. Compare results between `socket` and `nmap` methods.

Expected Outcome:

- The Python script successfully scans and lists open ports on the target machine.

6. Write a Java program to extract and display IP addresses, subnet masks, and default gateways from system network settings.

Aim:

To develop a Java program that extracts and displays IP addresses, subnet masks, and default gateways from system network settings.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Command Line or Terminal (Windows/Linux)

Experiment Steps:

Step 1: Set Up the Development Environment

1. Install Java Development Kit (JDK) if not already installed.

```
java -version
```

2. Install an IDE such as Eclipse or IntelliJ IDEA for writing and running Java programs.

Step 2: Understanding Network Interfaces in Java

1. Java provides the `NetworkInterface` class to obtain network-related details.
2. The `InetAddress` class is used to retrieve IP addresses.

Step 3: Implementing the Java Program

1. Create a new Java class (`NetworkDetails.java`).
2. Write the following Java program:

```
import java.net.*;
import java.util.*;

public class NetworkDetails {
    public static void main(String[] args) {
        try {
            Enumeration<NetworkInterface> interfaces =
NetworkInterface.getNetworkInterfaces();
            while (interfaces.hasMoreElements()) {
                NetworkInterface network = interfaces.nextElement();
                System.out.println("Interface: " +
network.getDisplayName());
                Enumeration<InetAddress> addresses =
network.getInetAddresses();
                while (addresses.hasMoreElements()) {
                    InetAddress address = addresses.nextElement();
                    System.out.println(" IP Address: " +
address.getHostAddress());
                }
            }
        }
    }
}
```

```

        } catch (SocketException e) {
            e.printStackTrace();
        }
    }
}

```

Step 4: Compiling and Running the Program

1. Open a terminal or command prompt.
2. Navigate to the directory where the Java file is saved.
3. Compile the Java program:

```
javac NetworkDetails.java
```

4. Run the program:

```
java NetworkDetails
```

Step 5: Extracting Subnet Mask and Default Gateway (Linux/Windows)

1. On Linux, use:

```
ifconfig -a
```

or

```
ip addr show
```

2. On Windows, use:

```
ipconfig
```

3. Modify the Java program to execute system commands and extract subnet masks and gateways using:

```

import java.io.*;

public class NetworkConfig {
    public static void main(String[] args) {
        try {
            Process process = Runtime.getRuntime().exec("ipconfig"); //
            Change to "ifconfig" for Linux
            BufferedReader reader = new BufferedReader(new
            InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains("Subnet Mask") ||
                line.contains("Gateway")) {
                    System.out.println(line.trim());
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```
}  
}
```

4. Compile and run the program as before.

Step 6: Testing the Program

1. Run the program on different operating systems (Windows, Linux, MacOS).
2. Verify the displayed network details with system commands (`ipconfig`, `ifconfig`, `ip addr show`).

Expected Outcome:

- ☐ The Java program successfully retrieves and displays the IP addresses, subnet masks, and default gateways from system network settings.

7. Implement a Python script to log all incoming and outgoing network connections using `psutil` and save the data for analysis.

Aim:

To develop a Python script that logs all incoming and outgoing network connections using the `psutil` library and saves the data for analysis.

Software & Tools Required:

- Python 3.x installed on the system
- `psutil` library for network connection monitoring
- Linux or Windows OS
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal or command prompt.
2. Install the `psutil` library if not already installed:

```
pip install psutil
```

3. Verify the installation by running:

```
python3 -c "import psutil; print('psutil installed successfully')"
```

Step 2: Implementing the Python Script

1. Create a Python script (`network_logger.py`) to monitor network connections.
2. Write the following Python code:

```
import psutil
import time
import csv

def log_connections(log_file="network_connections.csv"):
    with open(log_file, mode='w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(["Timestamp", "PID", "Local Address", "Remote Address", "Status", "Process Name"])

    while True:
        for conn in psutil.net_connections(kind='inet'):
            try:
                laddr = f"{conn.laddr.ip}:{conn.laddr.port}" if
conn.laddr else "N/A"
                raddr = f"{conn.raddr.ip}:{conn.raddr.port}" if
conn.raddr else "N/A"
                process = psutil.Process(conn.pid).name() if conn.pid
            else "N/A"
```

```
        writer.writerow([time.strftime("%Y-%m-%d %H:%M:%S"),
conn.pid, laddr, raddr, conn.status, process])
    except (psutil.NoSuchProcess, psutil.AccessDenied):
        continue
    time.sleep(5) # Log every 5 seconds

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

Step 3: Running the Script

1. Save the script as `network_logger.py`.
2. Run the script using:

```
python3 network_logger.py
```

3. The script will continuously log network connections and save them to `network_connections.csv`.

Step 4: Analyzing the Logged Data

1. Open the CSV file in Excel or a text editor.
2. Filter data based on process names, statuses, or remote addresses.
3. Identify unusual or suspicious network activity.

Step 5: Stopping the Script

1. Use `Ctrl + C` in the terminal to stop the script.
2. Verify the `network_connections.csv` file to ensure data has been logged.

Expected Outcome:

□ The Python script successfully logs all active incoming and outgoing network connections and saves them for further analysis.

8. Develop a Python-based monitoring tool to detect unauthorized changes to system network configuration files (/etc/network/interfaces, /etc/resolv.conf).

Aim:

To develop a Python-based monitoring tool that detects unauthorized changes to system network configuration files (/etc/network/interfaces, /etc/resolv.conf).

Software & Tools Required:

- Python 3.x installed on the system
- hashlib and time Python modules (pre-installed)
- Linux OS (Administrator/Root privileges required)
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Understanding the Concept

System network configuration files are critical for defining network interfaces and DNS settings. Any unauthorized modification to these files could lead to security vulnerabilities. This experiment develops a monitoring tool that computes hash values of target files and alerts the user if any modifications are detected.

Step 2: Implementing the Monitoring Script

1. Create a new Python script named `network_monitor.py`.
2. Write the following Python code:

```
import hashlib
import time
import os

# List of files to monitor
FILES_TO_MONITOR = ["/etc/network/interfaces", "/etc/resolv.conf"]

def get_file_hash(filepath):
    """Returns the SHA256 hash of the file contents."""
    try:
        with open(filepath, 'rb') as file:
            file_contents = file.read()
            return hashlib.sha256(file_contents).hexdigest()
    except FileNotFoundError:
        return None

def monitor_files(interval=10):
    """Continuously monitors the files for unauthorized changes."""
    file_hashes = {file: get_file_hash(file) for file in FILES_TO_MONITOR}

    print("Monitoring started. Press Ctrl+C to stop.")

    try:
        while True:
            time.sleep(interval)
```



```

        for file in FILES_TO_MONITOR:
            new_hash = get_file_hash(file)
            if new_hash is None:
                print(f"[ALERT] {file} has been deleted!")
            elif new_hash != file_hashes[file]:
                print(f"[WARNING] Unauthorized change detected in {file}!")
                file_hashes[file] = new_hash # Update stored hash
    except KeyboardInterrupt:
        print("Monitoring stopped.")

if __name__ == "__main__":
    monitor_files()

```

Step 3: Running the Script

1. Save the script as `network_monitor.py`.
2. Grant execution permissions (if necessary):

```
chmod +x network_monitor.py
```

3. Run the script using:

```
sudo python3 network_monitor.py
```

(Use `sudo` on Linux/macOS to ensure necessary read permissions.)

Step 4: Testing Unauthorized Changes

1. Open `/etc/resolv.conf` or `/etc/network/interfaces` in a text editor.

```
sudo nano /etc/resolv.conf
```

2. Modify and save the file.
3. Observe the monitoring tool's output in the terminal.

Step 5: Stopping the Monitor

1. Use `Ctrl + C` in the terminal to stop the script.
2. Analyze the output to identify detected changes.

Expected Outcome:

□ The Python script continuously monitors network configuration files and alerts the user if any unauthorized modifications are detected.

9. Develop a Python script to encrypt and decrypt files before transferring them over FTP using PyCryptodome.

Aim:

To develop a Python script using the PyCryptodome library to encrypt and decrypt files before transferring them over FTP securely.

Software & Tools Required:

- Python 3.x installed on the system
- PyCryptodome library for encryption and decryption
- `ftplib` for FTP file transfer
- FTP server (vsftpd, FileZilla, or other FTP services)
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal or command prompt.
2. Install PyCryptodome:

```
pip install pycryptodome
```

3. Verify the installation:

```
python3 -c "from Crypto.Cipher import AES; print('PyCryptodome installed successfully')"
```

Step 2: Implement File Encryption and Decryption

1. Create a Python script named `encrypt_decrypt.py`.
2. Write the following Python code:

```
from Crypto.Cipher import AES
import os

def pad(data):
    """Pads data to make it AES block size compliant."""
    return data + b" " * (16 - len(data) % 16)

def encrypt_file(input_file, output_file, key):
    """Encrypts a file using AES encryption."""
    cipher = AES.new(key, AES.MODE_ECB)
    with open(input_file, 'rb') as f:
        plaintext = f.read()
    ciphertext = cipher.encrypt(pad(plaintext))
    with open(output_file, 'wb') as f:
        f.write(ciphertext)
    print(f"File {input_file} encrypted successfully as {output_file}.")

def decrypt_file(input_file, output_file, key):
    """Decrypts a file using AES encryption."""
```

```

cipher = AES.new(key, AES.MODE_ECB)
with open(input_file, 'rb') as f:
    ciphertext = f.read()
plaintext = cipher.decrypt(ciphertext).rstrip(b" ")
with open(output_file, 'wb') as f:
    f.write(plaintext)
print(f"File {input_file} decrypted successfully as {output_file}.")

# Example Usage
key = b"thisisaverysecurekey!"[:16] # Ensure 16 bytes long
# Encrypt: encrypt_file("sample.txt", "sample_encrypted.bin", key)
# Decrypt: decrypt_file("sample_encrypted.bin", "sample_decrypted.txt", key)

```

Step 3: Implement Secure File Transfer Over FTP

1. Create another Python script named `ftp_transfer.py`.
2. Write the following Python code:

```

from ftplib import FTP
import os

def upload_file(ftp_server, username, password, file_path):
    ftp = FTP(ftp_server)
    ftp.login(user=username, passwd=password)
    with open(file_path, 'rb') as f:
        ftp.storbinary(f'STOR {os.path.basename(file_path)}', f)
    ftp.quit()
    print(f"Encrypted file {file_path} uploaded successfully.")

def download_file(ftp_server, username, password, remote_filename, local_path):
    ftp = FTP(ftp_server)
    ftp.login(user=username, passwd=password)
    with open(local_path, 'wb') as f:
        ftp.retrbinary(f'RETR {remote_filename}', f.write)
    ftp.quit()
    print(f"Encrypted file {remote_filename} downloaded successfully.")

# Example Usage
# upload_file("ftp.example.com", "user", "password", "sample_encrypted.bin")
# download_file("ftp.example.com", "user", "password", "sample_encrypted.bin",
               "downloaded_encrypted.bin")

```

Step 4: Running the Scripts

1. Encrypt a file before FTP transfer:

```
python3 encrypt_decrypt.py # Modify script to call encrypt_file function
```

2. Upload the encrypted file to an FTP server:

```
python3 ftp_transfer.py # Modify script to call upload_file function
```

3. Download the encrypted file from FTP:

```
python3 ftp_transfer.py # Modify script to call download_file function
```

4. Decrypt the downloaded file:

```
python3 encrypt_decrypt.py # Modify script to call decrypt_file function
```

Expected Outcome:

☐ The script successfully encrypts a file before uploading it via FTP and decrypts it after downloading, ensuring secure data transmission.

10. Implement a Python-based SSH brute-force attack detection system using paramiko to log failed login attempts.

Aim: To develop a Python script using the Paramiko library to detect and log failed SSH login attempts, helping in the prevention of brute-force attacks.

Software & Tools Required:

- Python 3.x installed on the system
- Paramiko library for SSH handling
- Logging module for maintaining attack logs
- SSH server (OpenSSH, Dropbear, or other SSH services)
- Text editor or IDE (VS Code, PyCharm, Nano, etc.)

Experiment Steps:

Step 1: Install Required Libraries

1. Open a terminal or command prompt.
2. Install Paramiko:

```
pip install paramiko
```

3. Verify the installation:

```
python3 -c "import paramiko; print('Paramiko installed successfully')"
```

Step 2: Implement SSH Brute-Force Detection

1. Create a Python script named `ssh_brute_force_detector.py`.
2. Write the following Python code:

```
import paramiko
import socket
import logging

# Configure logging
logging.basicConfig(filename="ssh_brute_force.log", level=logging.INFO,
                    format="%(asctime)s - %(message)s")

def detect_brute_force_attempts(server_ip, port, username, password_list):
    """Detects failed SSH login attempts using Paramiko."""
    for password in password_list:
        try:
            client = paramiko.SSHClient()
            client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
            client.connect(server_ip, port=port, username=username,
                           password=password, timeout=5)
            client.close()
            print("Login successful with password:", password)
            break
        except paramiko.AuthenticationException:
```

```
        logging.info(f"Failed login attempt for {username} with
password: {password}")
        print("Failed login attempt detected.")
    except socket.error:
        logging.error("Connection error occurred.")
        break

# Example usage
server_ip = "192.168.1.1"
port = 22
username = "admin"
password_list = ["password123", "admin123", "letmein", "123456"]

detect_brute_force_attempts(server_ip, port, username, password_list)
```

Step 3: Running the Script

1. Run the Python script to detect brute-force login attempts:

```
python3 ssh_brute_force_detector.py
```

2. Check the `ssh_brute_force.log` file for logged failed attempts:

```
cat ssh_brute_force.log
```

Expected Outcome: □ The script detects and logs failed SSH login attempts to a log file, providing insight into potential brute-force attacks.

11. Write a Java program to implement a secure login system using Java Security Manager policies.

Aim:

To develop a Java program that implements a secure login system by enforcing Java Security Manager policies to restrict unauthorized access.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Command Line or Terminal (Windows/Linux)

Experiment Steps:

Step 1: Understanding Java Security Manager

1. The Java Security Manager helps enforce security policies by restricting operations like file access and network operations.
2. It uses a security policy file (`.policy`) to define the allowed actions.

Step 2: Create a Security Policy File

1. Create a file named `secure.policy` with the following content:

```
grant {  
    permission java.io.FilePermission "login.txt", "read,write";  
    permission java.lang.RuntimePermission "exitVM";  
};
```

2. This policy allows reading and writing to `login.txt` and permits JVM exit.

Step 3: Implementing the Secure Login System in Java

1. Create a Java class (`SecureLogin.java`).
2. Write the following Java code:

```
import java.io.*;  
import java.security.AccessControlException;  
import java.util.Scanner;  
  
public class SecureLogin {  
    public static void main(String[] args) {  
        System.setSecurityManager(new SecurityManager());  
        String correctUsername = "admin";  
        String correctPassword = "password123";  
  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter username: ");  
        String username = scanner.nextLine();  
        System.out.print("Enter password: ");  
        String password = scanner.nextLine();
```

```

        if (username.equals(correctUsername) &&
password.equals(correctPassword)) {
            System.out.println("Login Successful!");
            try {
                FileWriter writer = new FileWriter("login.txt", true);
                writer.write("User " + username + " logged in
successfully.\n");
                writer.close();
            } catch (AccessControlException e) {
                System.out.println("Security Exception: File write access
denied!");
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Invalid Credentials!");
        }
        scanner.close();
    }
}

```

Step 4: Compiling and Running the Program

1. Compile the Java program:

```
javac SecureLogin.java
```

2. Run the program with the security manager enabled:

```
java -Djava.security.manager -Djava.security.policy=secure.policy
SecureLogin
```

Step 5: Testing Security Policy

1. Attempt login with correct credentials and check if login.txt is updated.
2. Modify secure.policy to remove FilePermission and rerun the program.
3. Observe that writing to login.txt is blocked due to security restrictions.

Expected Outcome:

□ The Java program successfully implements a secure login system while enforcing Java Security Manager policies to restrict unauthorized file access and ensure controlled security behavior.

12. Write a Java-based HTTPS server using Java's SSL libraries to encrypt client-server communication.

Aim:

To develop a secure HTTPS server in Java using SSL/TLS encryption for secure client-server communication.

Software & Tools Required:

- Java Development Kit (JDK)
- OpenSSL (for generating SSL certificates)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Web browser or cURL for testing

Experiment Steps:

Step 1: Generate an SSL Certificate

1. Open a terminal and generate a self-signed SSL certificate using Java's keytool:

```
keytool -genkeypair -alias myhttpsserver -keyalg RSA -keystore  
keystore.jks -validity 365 -storepass password
```

2. When prompted, enter details such as name, organization, and location.

Step 2: Create a Java HTTPS Server

1. Create a Java class (HttpsServerExample.java) to set up the server:

```
import com.sun.net.httpserver.HttpServer;  
import com.sun.net.httpserver.HttpsConfigurator;  
import com.sun.net.httpserver.HttpsServer;  
import javax.net.ssl.*;  
import java.io.IOException;  
import java.io.OutputStream;  
import java.net.InetSocketAddress;  
import java.security.KeyStore;  
import java.io.FileInputStream;  
  
public class HttpsServerExample {  
    public static void main(String[] args) throws Exception {  
        // Load the keystore  
        char[] password = "password".toCharArray();  
        KeyStore ks = KeyStore.getInstance("JKS");  
        FileInputStream fis = new FileInputStream("keystore.jks");  
        ks.load(fis, password);  
  
        // Set up the KeyManager  
        KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");  
        kmf.init(ks, password);  
  
        // Set up the SSL context  
        SSLContext sslContext = SSLContext.getInstance("TLS");
```

```

        sslContext.init(kmf.getKeyManagers(), null, null);

        // Create HTTPS server
        HttpsServer server = HttpsServer.create(new
        InetSocketAddress(8443), 0);
        server.setHttpsConfigurator(new HttpsConfigurator(sslContext));
        server.createContext("/test", exchange -> {
            String response = "Hello, Secure World!";
            exchange.sendResponseHeaders(200, response.length());
            OutputStream os = exchange.getResponseBody();
            os.write(response.getBytes());
            os.close();
        });

        server.setExecutor(null);
        server.start();
        System.out.println("HTTPS server started on port 8443");
    }
}

```

Step 3: Run the HTTPS Server

1. Compile and run the Java program:
2.

```
javac HttpsServerExample.java
java HttpsServerExample
```
3. The server will start on port 8443.

Step 4: Test the Server

1. Open a web browser and navigate to:

```
https://localhost:8443/test
```

2. Alternatively, use `cURL`:

```
curl -k https://localhost:8443/test
```

3. Verify that the response is `Hello, Secure World!`.

Expected Outcome:

□ The Java-based HTTPS server successfully encrypts client-server communication using SSL/TLS, ensuring secure data transmission.

13. Implement AES encryption in a Java application to securely store and retrieve sensitive data.

Aim:

To develop a Java application that uses AES encryption to securely store and retrieve sensitive data.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- AES Encryption Library (Java Cryptography API)

Experiment Steps:

Step 1: Understanding AES Encryption

1. AES (Advanced Encryption Standard) is a symmetric encryption algorithm used for secure data storage.
2. It operates on blocks of data and supports key sizes of 128, 192, or 256 bits.

Step 2: Implement AES Encryption in Java

1. Create a Java class (AEEncryption.java) to handle encryption and decryption:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AEEncryption {
    private static final String AES_ALGORITHM = "AES";

    // Generate AES Secret Key
    public static SecretKey generateKey() throws Exception {
        KeyGenerator keyGenerator =
        KeyGenerator.getInstance(AES_ALGORITHM);
        keyGenerator.init(128); // AES key size
        return keyGenerator.generateKey();
    }

    // Encrypt data using AES
    public static String encrypt(String data, SecretKey key) throws
    Exception {
        Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
        cipher.init(Cipher.ENCRYPT_MODE, key);
        byte[] encryptedBytes = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    // Decrypt data using AES
    public static String decrypt(String encryptedData, SecretKey key)
    throws Exception {
        Cipher cipher = Cipher.getInstance(AES_ALGORITHM);
```

```

        cipher.init(Cipher.DECRYPT_MODE, key);
        byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(encryptedData));
        return new String(decryptedBytes);
    }

    public static void main(String[] args) throws Exception {
        SecretKey key = generateKey();
        String originalData = "SensitiveData123";

        String encryptedData = encrypt(originalData, key);
        System.out.println("Encrypted Data: " + encryptedData);

        String decryptedData = decrypt(encryptedData, key);
        System.out.println("Decrypted Data: " + decryptedData);
    }
}

```

Step 3: Compile and Run the Java Program

1. Compile the program:

```
javac AESEncryption.java
```

2. Run the program:

```
java AESEncryption
```

3. Observe the encrypted and decrypted outputs in the console.

Step 4: Store and Retrieve Encrypted Data Securely

1. Modify the program to store the encrypted data in a file or database.
2. Retrieve and decrypt data securely when needed.

Expected Outcome:

□ The Java application successfully encrypts and decrypts sensitive data using AES, ensuring secure storage and retrieval.

14. Create a Java application that verifies the integrity of a downloaded file using a SHA-256 checksum.

Aim:

To develop a Java application that computes and verifies the SHA-256 checksum of a downloaded file to ensure its integrity.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Sample file for checksum verification

Experiment Steps:

Step 1: Understanding SHA-256 Checksum

1. SHA-256 is a cryptographic hash function that generates a fixed 256-bit hash value for data.
2. It is commonly used to verify file integrity after downloading.

Step 2: Implement SHA-256 Checksum Calculation in Java

1. Create a Java class (FileChecksum.java) to compute the SHA-256 hash:

```
import java.io.File;
import java.io.FileInputStream;
import java.security.MessageDigest;
import java.util.Formatter;

public class FileChecksum {
    public static String calculateSHA256(File file) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        FileInputStream fis = new FileInputStream(file);
        byte[] byteArray = new byte[1024];
        int bytesRead;

        while ((bytesRead = fis.read(byteArray)) != -1) {
            digest.update(byteArray, 0, bytesRead);
        }
        fis.close();

        byte[] hashBytes = digest.digest();
        return bytesToHex(hashBytes);
    }

    private static String bytesToHex(byte[] bytes) {
        Formatter formatter = new Formatter();
        for (byte b : bytes) {
            formatter.format("%02x", b);
        }
        String result = formatter.toString();
        formatter.close();
        return result;
    }
}
```

```
        public static void main(String[] args) throws Exception {  
            File file = new File("sample.txt"); // Replace with actual file  
path            String checksum = calculateSHA256(file);  
            System.out.println("SHA-256 Checksum: " + checksum);  
        }  
    }
```

Step 3: Compile and Run the Java Program

1. Compile the program:

```
javac FileChecksum.java
```

2. Run the program:

```
java FileChecksum
```

3. Observe the SHA-256 hash output for the file.

Step 4: Verify File Integrity

1. Compare the generated SHA-256 hash with the provided checksum of the downloaded file.
2. If they match, the file is intact; if not, the file is corrupted or tampered with.

Expected Outcome:

□ The Java application successfully calculates and verifies the SHA-256 checksum of a file, ensuring its integrity after downloading.

15. Develop a Java program that implements RSA encryption and decryption for secure data transfer.

Aim:

To develop a Java application that implements RSA encryption and decryption for secure data transfer.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Java Cryptography API

Experiment Steps:

Step 1: Understanding RSA Encryption

1. RSA (Rivest-Shamir-Adleman) is an asymmetric encryption algorithm used for secure data transmission.
2. It uses a pair of keys: a public key for encryption and a private key for decryption.

Step 2: Implement RSA Encryption and Decryption in Java

1. Create a Java class (RSAEncryption.java) to generate keys, encrypt, and decrypt data:

```
import java.security.*;
import javax.crypto.Cipher;
import java.util.Base64;

public class RSAEncryption {
    private static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator keyPairGenerator =
        KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(2048);
        return keyPairGenerator.generateKeyPair();
    }

    public static String encrypt(String data, PublicKey publicKey) throws
    Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] encryptedBytes = cipher.doFinal(data.getBytes());
        return Base64.getEncoder().encodeToString(encryptedBytes);
    }

    public static String decrypt(String encryptedData, PrivateKey
    privateKey) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        byte[] decryptedBytes =
        cipher.doFinal(Base64.getDecoder().decode(encryptedData));
        return new String(decryptedBytes);
    }
}
```

```
public static void main(String[] args) throws Exception {
    KeyPair keyPair = generateKeyPair();
    String originalData = "SecureMessage123";

    String encryptedData = encrypt(originalData, keyPair.getPublic());
    System.out.println("Encrypted Data: " + encryptedData);

    String decryptedData = decrypt(encryptedData,
    keyPair.getPrivate());
    System.out.println("Decrypted Data: " + decryptedData);
}
}
```

Step 3: Compile and Run the Java Program

1. Compile the program:

```
javac RSAEncryption.java
```

2. Run the program:

```
java RSAEncryption
```

3. Observe the encrypted and decrypted outputs in the console.

Step 4: Secure Key Storage and Transfer

1. Store private keys securely to prevent unauthorized access.
2. Use digital signatures to verify the authenticity of messages.

Expected Outcome:

□ The Java application successfully encrypts and decrypts data using RSA, ensuring secure data transfer.

16. Write a Java-based secure authentication system that uses hashed passwords with salting for user login.

Aim:

To develop a Java application that implements a secure authentication system using hashed passwords with salting to enhance security.

Software & Tools Required:

- Java Development Kit (JDK)
- Java IDE (Eclipse, IntelliJ IDEA, VS Code)
- Java Cryptography API
- SQLite or MySQL for user data storage

Experiment Steps:

Step 1: Understanding Hashed Passwords with Salting

1. Hashing converts passwords into a fixed-length string, making it irreversible.
2. Salting adds a unique random value to each password before hashing to prevent attacks like rainbow table attacks.
3. Secure hashing algorithms like PBKDF2, bcrypt, or SHA-256 are used for password security.

Step 2: Implement Secure Password Hashing in Java

1. Create a Java class (`PasswordHasher.java`) for hashing and salting passwords:

```
import java.security.SecureRandom;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;

public class PasswordHasher {
    private static final SecureRandom RANDOM = new SecureRandom();

    public static String generateSalt() {
        byte[] salt = new byte[16];
        RANDOM.nextBytes(salt);
        return Base64.getEncoder().encodeToString(salt);
    }

    public static String hashPassword(String password, String salt) throws
    NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(salt.getBytes());
        byte[] hashedPassword = md.digest(password.getBytes());
        return Base64.getEncoder().encodeToString(hashedPassword);
    }
}
```

Step 3: Implement User Authentication

1. Create a Java class (`UserAuthentication.java`) to register and authenticate users:

```

import java.util.HashMap;

public class UserAuthentication {
    private static HashMap<String, String[]> userDatabase = new
HashMap<>();

    public static void registerUser(String username, String password)
throws Exception {
        String salt = PasswordHasher.generateSalt();
        String hashedPassword = PasswordHasher.hashPassword(password,
salt);
        userDatabase.put(username, new String[]{hashedPassword, salt});
        System.out.println("User registered successfully!");
    }

    public static boolean authenticateUser(String username, String
password) throws Exception {
        if (userDatabase.containsKey(username)) {
            String[] storedData = userDatabase.get(username);
            String hashedPassword = PasswordHasher.hashPassword(password,
storedData[1]);
            return storedData[0].equals(hashedPassword);
        }
        return false;
    }

    public static void main(String[] args) throws Exception {
        registerUser("user1", "securePassword");
        boolean isAuthenticated = authenticateUser("user1",
"securePassword");
        System.out.println("Authentication success: " + isAuthenticated);
    }
}

```

Step 4: Compile and Run the Java Program

1. Compile the program:
`javac PasswordHasher.java UserAuthentication.java`
2. Run the program:
`java UserAuthentication`
3. Observe user registration and authentication in the console output.

Step 5: Enhance Security

1. Use PBKDF2 or bcrypt instead of SHA-256 for better security.
2. Store user credentials securely in a database instead of an in-memory HashMap.
3. Implement account lockout policies after multiple failed attempts.

Expected Outcome:

□ The Java application successfully implements a secure authentication system using hashed passwords with salting, ensuring robust user security.

17. Implement and compare hash functions (MD5, SHA-256, SHA-512) using a Python program to verify file integrity.

Aim: To implement hash functions (MD5, SHA-256, SHA-512) using Python and compare their outputs to verify file integrity.

Software & Tools Required:

- Python 3.x
- Text editor (VSCode, Sublime Text, etc.)
- Command line or terminal

Experiment Steps:

Step 1: Understanding Hash Functions

A hash function takes an input (or 'message') and returns a fixed-size string, which typically represents the content of the input. Hash functions like MD5, SHA-256, and SHA-512 are commonly used for tasks like file integrity checks, password storage, and digital signatures.

- **MD5:** Produces a 128-bit hash value (16 bytes).
- **SHA-256:** Part of the SHA-2 family, produces a 256-bit hash value (32 bytes).
- **SHA-512:** Also part of the SHA-2 family, produces a 512-bit hash value (64 bytes).

Each hash function is designed to return a unique hash for every distinct input, but small changes in the input will drastically change the output.

Step 2: Writing the Python Program

Create a Python program that calculates and compares the hash values for MD5, SHA-256, and SHA-512 for a given file.

Python Code (hash_functions.py):

```
import hashlib

# Function to compute hash values for a given file using MD5, SHA-256, and SHA-512
def compute_hash(file_path):
    # Create empty dictionaries to store the hash values
    hashes = {"MD5": "", "SHA-256": "", "SHA-512": ""}

    # Open the file in binary mode
    with open(file_path, 'rb') as file:
        # Read the file in chunks to avoid memory overflow for large files
        file_data = file.read()

        # MD5 hash calculation
        md5_hash = hashlib.md5()
        md5_hash.update(file_data)
        hashes["MD5"] = md5_hash.hexdigest()
```

```

    # SHA-256 hash calculation
    sha256_hash = hashlib.sha256()
    sha256_hash.update(file_data)
    hashes["SHA-256"] = sha256_hash.hexdigest()

    # SHA-512 hash calculation
    sha512_hash = hashlib.sha512()
    sha512_hash.update(file_data)
    hashes["SHA-512"] = sha512_hash.hexdigest()

    return hashes

# Function to compare hash values
def compare_hashes(hashes):
    print(f"MD5 Hash: {hashes['MD5']}")
    print(f"SHA-256 Hash: {hashes['SHA-256']}")
    print(f"SHA-512 Hash: {hashes['SHA-512']}")

# Main function to test the hash functions
if __name__ == "__main__":
    file_path = input("Enter the path of the file to hash: ")
    hashes = compute_hash(file_path)
    compare_hashes(hashes)

```

Step 3: Running the Program

1. Save the code in a file called `hash_functions.py`.
2. Open the terminal and navigate to the directory where the Python script is located.
3. Run the program using the command:

```

bash
CopyEdit
python hash_functions.py

```

4. Enter the path of the file you want to check for integrity when prompted.

Step 4: Verifying File Integrity Using Hashes

- Once the program runs, it will output the MD5, SHA-256, and SHA-512 hash values of the provided file.
- You can compare these hash values with a trusted source to verify that the file is intact.
- To check if a file has been altered, calculate the hash of the file at a later time and compare it to the original hash. If the hashes match, the file is intact; if they differ, the file has been modified.

Step 5: Comparing Hash Functions

1. **MD5:** Fast but considered weak for cryptographic purposes. It is vulnerable to collisions (two different inputs producing the same hash).
2. **SHA-256:** Stronger than MD5, producing a longer and more secure hash.
3. **SHA-512:** Provides an even longer hash than SHA-256 and is more resistant to collision attacks, making it the most secure of the three.

By comparing the hashes, you can observe the differences in hash length and security between MD5, SHA-256, and SHA-512.

Expected Outcome:

- The program successfully computes and displays the MD5, SHA-256, and SHA-512 hash values for the given file.
 - File integrity is verified by comparing the computed hash with the known hash of the original file.
 - The user can observe the difference in hash lengths and the security level of each hash function.
-

18. Write a Python program to generate an HMAC-based authentication system using the hmac module.

Aim: To develop a Python program that uses the `hmac` module to create a secure HMAC-based (Hash-based Message Authentication Code) authentication system.

Software & Tools Required:

- Python 3.x
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding HMAC

HMAC (Hash-based Message Authentication Code) is a mechanism used to verify the integrity and authenticity of a message. It combines a cryptographic hash function (like SHA-256) with a secret key to produce a unique value (HMAC).

- **How HMAC Works:**
 1. The message is combined with a secret key.
 2. A hash function is applied to the combined message and key.
 3. The resulting hash value is used to verify the authenticity of the message.

HMAC ensures that even if the message is intercepted during transmission, an attacker cannot modify the message without the correct key.

Step 2: Writing the Python Program

Create a Python program that uses the `hmac` module to authenticate messages and ensure secure communication.

Python Code (`hmac_authentication.py`):

```
import hmac
import hashlib
import time

# Secret key for HMAC (this should be kept secure)
SECRET_KEY = b'supersecretkey'

# Function to generate HMAC for a given message
def generate_hmac(message, key):
    return hmac.new(key, message.encode(), hashlib.sha256).hexdigest()

# Function to verify the HMAC for a given message
def verify_hmac(message, key, expected_hmac):
    generated_hmac = generate_hmac(message, key)
    return hmac.compare_digest(generated_hmac, expected_hmac)
```

```

# Simulating a basic authentication process
def authenticate_user(username, password):
    # In a real-world scenario, the password would be hashed and stored securely
    # Here, we simulate a stored password hash
    stored_password_hash = generate_hmac("userpassword", SECRET_KEY)

    # Verify the HMAC of the provided password against the stored hash
    provided_password_hash = generate_hmac(password, SECRET_KEY)

    if verify_hmac(provided_password_hash, SECRET_KEY, stored_password_hash):
        print("Authentication Successful!")
        return True
    else:
        print("Authentication Failed!")
        return False

# Simulating login attempt
if __name__ == "__main__":
    print("Welcome to the Secure Authentication System!")

    username = input("Enter username: ")
    password = input("Enter password: ")

    # Authenticate the user with HMAC
    authenticate_user(username, password)

```

Step 3: Running the Program

1. Save the Python script as `hmac_authentication.py`.
2. Open your terminal and navigate to the directory where the script is saved.
3. Run the script using the following command:

```

bash
CopyEdit
python hmac_authentication.py

```

4. When prompted, enter the username and password. The system will authenticate the user based on the password and secret key stored within the system.

Step 4: Testing the Authentication System

- **Correct Password:** If you enter the correct password (e.g., "userpassword"), the program should print "Authentication Successful!".
- **Incorrect Password:** If the password is incorrect, it will print "Authentication Failed!".

Step 5: Improving the Authentication System

- **Session Management:** For real-world applications, after successful authentication, implement session management (e.g., using tokens) to manage user sessions.
 - **Salting:** Salt the password before hashing it with HMAC to increase security.
 - **More Complex Security:** You can integrate additional features, such as rate-limiting login attempts or using multi-factor authentication (MFA).
-

Expected Outcome:

- The program successfully uses the HMAC mechanism to authenticate users securely by comparing the generated HMAC of the password against the stored HMAC.
 - The system correctly identifies and authenticates users if the correct password is entered and denies access if the password is incorrect.
-

19. Develop a Python script to create and verify digital signatures using the PyCryptodome library.

Aim: To develop a Python script that creates and verifies digital signatures using the PyCryptodome library in Python.

Software & Tools Required:

- Python 3.x
 - PyCryptodome library (Python package)
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding Digital Signatures

A **digital signature** is a cryptographic technique used to authenticate the sender of a message and to ensure that the message has not been tampered with. It uses a pair of keys:

- **Private key:** Used to create the digital signature.
- **Public key:** Used to verify the signature.

The signature is created by hashing the message and encrypting the hash with the sender's private key. The recipient can then decrypt the hash with the sender's public key and compare it to the hash of the received message to verify its authenticity.

Step 2: Installing the PyCryptodome Library

First, install the PyCryptodome library if you haven't already. Open your terminal or command prompt and run the following command:

```
pip install pycryptodome
```

Step 3: Writing the Python Script for Digital Signatures

Create a Python script that:

1. Generates a key pair (private and public keys).
2. Creates a digital signature using the private key.
3. Verifies the digital signature using the public key.

Python Code (digital_signature.py):

```
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.Random import get_random_bytes
import binascii
```

```

# Function to generate RSA key pair
def generate_keys():
    key = RSA.generate(2048) # 2048-bit RSA key
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key

# Function to create a digital signature
def create_signature(message, private_key):
    # Hash the message using SHA-256
    h = SHA256.new(message.encode())

    # Load the private key
    private_key_obj = RSA.import_key(private_key)

    # Create the signature using the private key and hashed message
    signature = pkcs1_15.new(private_key_obj).sign(h)
    return signature

# Function to verify the digital signature
def verify_signature(message, signature, public_key):
    # Hash the message
    h = SHA256.new(message.encode())

    # Load the public key
    public_key_obj = RSA.import_key(public_key)

    try:
        # Verify the signature using the public key
        pkcs1_15.new(public_key_obj).verify(h, signature)
        print("Signature is valid!")
    except (ValueError, TypeError):
        print("Signature verification failed!")

# Main function
if __name__ == "__main__":
    # Step 1: Generate RSA keys
    private_key, public_key = generate_keys()

    # Step 2: Define a message
    message = "This is a secret message!"

    # Step 3: Create a digital signature for the message
    print(f"Original Message: {message}")
    signature = create_signature(message, private_key)

    # Print the signature (in hexadecimal format)
    print(f"Digital Signature: {binascii.hexlify(signature).decode()}")

    # Step 4: Verify the digital signature
    print("\nVerifying the signature with the public key...")
    verify_signature(message, signature, public_key)

```

Step 4: Running the Program

1. Save the Python script as `digital_signature.py`.
2. Open your terminal and navigate to the directory where the script is saved.
3. Run the script using the following command:

```
python digital_signature.py
```

4. The program will generate a public-private key pair, create a digital signature for the message, and verify the signature.

Step 5: Verifying the Digital Signature

- The script generates a **private key** and **public key**.
- It then hashes a message and creates a **digital signature** using the private key.
- The program verifies the signature by comparing the hash of the message with the decrypted hash from the signature using the **public key**.

If the signature is valid, it will print "Signature is valid!"; otherwise, it will print "Signature verification failed!".

Step 6: Testing Signature Integrity

To test the integrity of the digital signature:

1. Modify the message and try to verify the signature again.
2. The program should fail to verify the signature because the message no longer matches the original one.

Expected Outcome:

- The program generates a key pair (private and public keys).
 - It creates a digital signature using the private key and verifies the signature using the public key.
 - The signature is verified successfully when the message is intact. If the message is altered, the verification should fail.
-

20. Implement a challenge-response authentication system in Python using one-time passwords (OTP) and `secrets` module.

Aim: To develop a challenge-response authentication system using one-time passwords (OTP) and Python's `secrets` module for secure OTP generation.

Software & Tools Required:

- Python 3.x
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding Challenge-Response Authentication with OTP

Challenge-response authentication is a method where:

- The server sends a challenge (a random number or question) to the client.
- The client must respond correctly by providing the correct response (an OTP) generated using a secret key and challenge.

In this experiment, the **OTP** (One-Time Password) will be generated using Python's `secrets` module, which provides a secure way to generate cryptographically strong random numbers suitable for cryptographic applications.

Step 2: Installing Required Libraries

The `secrets` module is built into Python's standard library (Python 3.6+), so no additional installation is required.

Step 3: Writing the Python Script for OTP-Based Challenge-Response Authentication

In this script:

1. The server generates a challenge (random string or number).
2. The server then generates an OTP using the challenge and a secret key.
3. The client receives the challenge and responds with the OTP.
4. The server verifies the OTP and provides access if correct.

Python Code (`otp_challenge_response.py`):

```
import secrets
import time

# Secret key for OTP generation (in real-world applications, this should be
# securely stored)
SECRET_KEY = "SuperSecretKey"
```

```

# Function to generate a one-time password (OTP) based on a secret key and
challenge
def generate_otp(challenge, secret_key):
    # Combine the challenge and secret key to generate a secure OTP
    otp_input = challenge + secret_key
    otp = secrets.token_hex(4) # Generate a 4-byte OTP (8 hex characters)
    return otp

# Server-side function to generate the challenge and OTP
def server_generate_challenge():
    challenge = secrets.token_hex(4) # Generate a random challenge (4-byte
random string)
    otp = generate_otp(challenge, SECRET_KEY)
    return challenge, otp

# Client-side function to respond with OTP
def client_respond(challenge, otp):
    # The client would typically generate the OTP based on the secret key and
challenge
    # Here, for simplicity, we're directly passing the correct OTP
    return otp

# Function to simulate the challenge-response authentication process
def authenticate_user():
    print("Welcome to the Secure OTP Authentication System!")

    # Server generates a challenge and OTP
    challenge, correct_otp = server_generate_challenge()

    # Server displays the challenge (this would be shown to the client)
    print(f"Challenge: {challenge}")

    # Simulating the client receiving the challenge and generating the response
(OTP)
    client_otp = client_respond(challenge, correct_otp)

    # Server verifies the client's response
    if client_otp == correct_otp:
        print("Authentication Successful!")
    else:
        print("Authentication Failed!")

# Main function
if __name__ == "__main__":
    authenticate_user()

```

Step 4: Running the Program

1. Save the Python script as `otp_challenge_response.py`.
2. Open your terminal or command prompt and navigate to the directory where the script is saved.
3. Run the script using the following command:

```
python otp_challenge_response.py
```

4. The program will generate a challenge, create an OTP, and then simulate the authentication process by verifying the OTP provided by the client.

Step 5: Testing the Authentication System

- The server generates a random challenge and an OTP.
- The client (simulated here) sends back the correct OTP.
- The server verifies the OTP and prints whether authentication was successful or failed.

Step 6: Enhancing the OTP System

- **Time-based OTPs (TOTP):** You can enhance this system by adding time-based OTPs using the current timestamp as part of the OTP generation.
 - **OTP Expiry:** Implement OTP expiration by checking the time when the OTP was generated.
 - **Secure Storage:** In a real-world system, store the secret key securely (using secure storage techniques or databases).
-

Expected Outcome:

- The program successfully generates a random challenge and OTP.
 - The server successfully verifies the OTP provided by the client.
 - The authentication process is simulated, and authentication results are printed (either successful or failed).
-

21. Develop a Python script to implement two-factor authentication (2FA) using the `pyotp` library.

Aim: To implement two-factor authentication (2FA) using the `pyotp` library in Python, which generates time-based one-time passwords (TOTP) for added security.

Software & Tools Required:

- Python 3.x
 - `pyotp` library (for generating OTPs)
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding Two-Factor Authentication (2FA)

Two-factor authentication (2FA) is a security process in which a user must provide two forms of identification to access an account:

1. **Something you know:** A password or PIN.
2. **Something you have:** A one-time password (OTP) generated by a system or an app.

In this experiment, we will implement 2FA using **TOTP (Time-based One-Time Password)**, which is a common method where the OTP expires after a short period of time (usually 30 seconds).

Step 2: Installing the `pyotp` Library

`pyotp` is a Python library used for generating OTPs. Install it using pip:

```
pip install pyotp
```

Step 3: Writing the Python Script for 2FA

We will create a simple system where:

1. The **server** generates a shared secret.
2. The **client** generates an OTP based on that secret.
3. The **server** verifies the OTP submitted by the client within a time window.

Python Code (`two_factor_auth.py`):

```
import pyotp
import time

# Function to generate a secret key for 2FA
def generate_secret():
    # Generate a random secret for the user (this is shared between client and server)
    totp = pyotp.TOTP(pyotp.random_base32())
```

```

    secret = totp.secret
    print(f"Your secret key is: {secret}")
    return secret

# Function to generate OTP (this is typically done by the client)
def generate_otp(secret):
    totp = pyotp.TOTP(secret)
    otp = totp.now() # Generate a valid OTP based on the shared secret
    print(f"Generated OTP: {otp}")
    return otp

# Function to verify OTP (this is typically done by the server)
def verify_otp(secret, otp):
    totp = pyotp.TOTP(secret)
    if totp.verify(otp):
        print("OTP Verified Successfully!")
    else:
        print("Invalid OTP! Verification Failed.")

# Main function
def main():
    print("Two-Factor Authentication (2FA) Demo")

    # Step 1: Generate a secret (this would be shared between the server and the client)
    secret = generate_secret()

    # Step 2: Generate OTP (client generates OTP based on the shared secret)
    otp = generate_otp(secret)

    # Step 3: User enters the OTP (simulating client input here)
    entered_otp = input("Enter the OTP: ")

    # Step 4: Verify OTP (server verifies the OTP entered by the client)
    verify_otp(secret, entered_otp)

if __name__ == "__main__":
    main()

```

Step 4: Running the Program

1. Save the Python script as `two_factor_auth.py`.
2. Open your terminal or command prompt and navigate to the directory where the script is saved.
3. Run the script using the following command:

```
python two_factor_auth.py
```

4. The program will:
 - Generate a secret key for the user.
 - Generate a one-time password (OTP) based on the secret.
 - Ask the user to enter the OTP.
 - Verify the OTP entered by the user.

Step 5: Testing the 2FA System

- When you run the script, it will display the secret key.
- The server will generate an OTP using this secret key.
- You will be asked to input the OTP. If it matches the generated OTP, the authentication will be successful.

Step 6: Expiry of OTP

- The OTP expires every 30 seconds (the default time window for TOTP).
- If you wait too long before entering the OTP, it will be invalid and the server will reject it.

Expected Outcome:

- The script will generate a secret key and an OTP based on it.
 - The OTP will expire after 30 seconds.
 - The user will be prompted to input the OTP, and if the OTP is valid, they will be authenticated successfully.
-

22. Write a Python program to compare the performance and security of SHA-256 and bcrypt hashing algorithms.

Aim: To compare the performance and security of the `SHA-256` and `bcrypt` hashing algorithms in terms of their computation time and resistance to brute-force attacks.

Software & Tools Required:

- Python 3.x
 - `hashlib` (for SHA-256 hashing)
 - `bcrypt` library (for `bcrypt` hashing)
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding Hashing Algorithms

Hashing algorithms are used to convert data into a fixed-size string of characters, which is typically a hash value or digest. Hash functions are one-way operations, meaning it's computationally infeasible to reverse the process and retrieve the original input.

- **SHA-256:** A part of the SHA-2 family, it produces a 256-bit (32-byte) hash value. It is fast but vulnerable to brute-force and rainbow table attacks, as it is not inherently designed to be computationally slow.
- **bcrypt:** A key derivation function that is deliberately designed to be slow, which helps defend against brute-force and rainbow table attacks. It incorporates a "salt" to ensure that identical passwords do not hash to the same value.

Step 2: Installing Required Libraries

Install the `bcrypt` library to perform `bcrypt` hashing:

```
pip install bcrypt
```

Step 3: Writing the Python Program

We will write a Python script that:

1. Hashes a given input string (e.g., a password) using both `SHA-256` and `bcrypt`.
2. Compares the computation time for both algorithms.
3. Discusses the security considerations and resistance to brute-force attacks.

Python Code (`hashing_comparison.py`):

```
import hashlib
import bcrypt
import time
```

```
# Function to hash using SHA-256
```

```

def hash_sha256(data):
    sha256 = hashlib.sha256()
    sha256.update(data.encode('utf-8'))
    return sha256.hexdigest()

# Function to hash using bcrypt
def hash_bcrypt(data):
    salt = bcrypt.gensalt()
    return bcrypt.hashpw(data.encode('utf-8'), salt)

# Function to compare performance and security of SHA-256 and bcrypt
def compare_hashing_algorithms():
    data = "supersecretpassword"

    # SHA-256 hashing
    start_time = time.time()
    sha256_hash = hash_sha256(data)
    sha256_time = time.time() - start_time
    print(f"SHA-256 Hash: {sha256_hash}")
    print(f"SHA-256 Time: {sha256_time:.6f} seconds")

    # bcrypt hashing
    start_time = time.time()
    bcrypt_hash = hash_bcrypt(data)
    bcrypt_time = time.time() - start_time
    print(f"bcrypt Hash: {bcrypt_hash}")
    print(f"bcrypt Time: {bcrypt_time:.6f} seconds")

    # Comparing performance
    print("\nPerformance Comparison:")
    print(f"SHA-256 is faster than bcrypt by {bcrypt_time/sha256_time:.2f} times.")

    # Security Considerations
    print("\nSecurity Considerations:")
    print("1. SHA-256 is fast, making it vulnerable to brute-force attacks and rainbow table attacks.")
    print("2. bcrypt is designed to be slow, increasing the time for brute-force attacks and making it more secure.")
    print("3. bcrypt also uses a salt to protect against rainbow table attacks.")
    print("4. bcrypt hashes cannot be reversed, and it includes a work factor (cost factor) to make hashing more expensive.")

# Main function to run the comparison
if __name__ == "__main__":
    compare_hashing_algorithms()

```

Step 4: Running the Program

1. Save the Python script as `hashing_comparison.py`.
2. Open your terminal or command prompt and navigate to the directory where the script is saved.
3. Run the script using the following command:

```
python hashing_comparison.py
```

4. The program will:

- Hash a password (`supersecretpassword`) using both SHA-256 and `bcrypt`.
- Display the resulting hashes.
- Show the computation times for both algorithms.
- Provide a comparison of the performance and security.

Step 5: Verifying the Results

- The SHA-256 hashing algorithm will likely be much faster than `bcrypt`.
- `bcrypt` will take longer to compute the hash due to its deliberate slowness.
- The security of `bcrypt` will be higher, especially against brute-force attacks, due to its work factor and use of a salt.

Expected Outcome:

- **Performance:** SHA-256 will be faster than `bcrypt` because it is designed to be a quick hash function.
 - **Security:** `bcrypt` will be more secure than SHA-256 because it is resistant to brute-force and rainbow table attacks due to its slowness and salt usage.
 - **Comparison:** The program will provide a side-by-side comparison of the two algorithms in terms of time and security.
-

23. Implement a Python script to encrypt and authenticate messages using AES-GCM (Authenticated Encryption).

Aim: To implement a Python script that encrypts and authenticates messages using the AES-GCM mode of operation for authenticated encryption, ensuring confidentiality, integrity, and authenticity.

Software & Tools Required:

- Python 3.x
 - `pycryptodome` library (for AES-GCM encryption)
 - Text editor (VSCode, Sublime Text, etc.)
 - Command line or terminal
-

Experiment Steps:

Step 1: Understanding AES-GCM (Authenticated Encryption)

AES-GCM (Galois/Counter Mode) is an authenticated encryption mode that provides both confidentiality (encryption) and integrity/authentication (message authentication code - MAC). It combines:

1. **Encryption:** Using AES to encrypt the plaintext message.
2. **Authentication:** Using a MAC (Galois MAC) to verify both the authenticity and integrity of the message.

AES-GCM requires:

- **Key:** A secret key for AES encryption.
- **Nonce/IV (Initialization Vector):** A unique value for each encryption operation to ensure that the same plaintext produces different ciphertext each time.
- **Additional Authenticated Data (AAD):** Optional data that is authenticated but not encrypted.
- **Ciphertext:** The encrypted message.
- **Tag:** The authentication tag to verify the integrity of the ciphertext.

Step 2: Installing the `pycryptodome` Library

Install the `pycryptodome` library, which supports AES encryption and other cryptographic operations:

```
pip install pycryptodome
```

Step 3: Writing the Python Script for AES-GCM Encryption and Authentication

We will write a Python script to:

1. Encrypt a message using AES-GCM.
2. Authenticate the message using an authentication tag.

3. Decrypt the message and verify its authenticity.

Python Code (aes_gcm_encryption.py):

```
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
from Crypto.Protocol.KDF import scrypt
import base64

# Function to generate a random key (AES-256 requires a 32-byte key)
def generate_key():
    key = get_random_bytes(32) # AES-256 key size
    return key

# Function to encrypt a message using AES-GCM
def encrypt_message(key, message, aad=None):
    # Generate a random 12-byte nonce (recommended size for GCM)
    nonce = get_random_bytes(12)

    # Create AES-GCM cipher object
    cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)

    # Encrypt the message and return ciphertext, nonce, and tag
    cipher.update(aad) # Optional AAD (Additional Authenticated Data)
    ciphertext, tag = cipher.encrypt_and_digest(message.encode())

    return ciphertext, nonce, tag

# Function to decrypt a message using AES-GCM and verify its authenticity
def decrypt_message(key, ciphertext, nonce, tag, aad=None):
    # Create AES-GCM cipher object
    cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)

    cipher.update(aad) # Optional AAD (Additional Authenticated Data)

    # Decrypt the message and verify the authenticity
    try:
        decrypted_message = cipher.decrypt_and_verify(ciphertext, tag)
        return decrypted_message.decode()
    except (ValueError, TypeError):
        print("Decryption failed or message is tampered!")
        return None

# Main function to demonstrate AES-GCM encryption and decryption
def main():
    # The message to be encrypted
    message = "This is a confidential and authenticated message."

    # Generate the AES key
    key = generate_key()

    # Encrypt the message
    print("Encrypting message...")
    aad = b"Additional Authenticated Data" # Optional AAD
    ciphertext, nonce, tag = encrypt_message(key, message, aad)
```

```

# Print the encrypted message details
print(f"Ciphertext (Base64): {base64.b64encode(ciphertext).decode()}")
print(f"Nonce (Base64): {base64.b64encode(nonce).decode()}")
print(f"Authentication Tag (Base64): {base64.b64encode(tag).decode()}")

# Decrypt the message
print("\nDecrypting message...")
decrypted_message = decrypt_message(key, ciphertext, nonce, tag, aad)

if decrypted_message:
    print(f"Decrypted Message: {decrypted_message}")
else:
    print("Failed to decrypt the message.")

if __name__ == "__main__":
    main()

```

Step 4: Running the Program

1. Save the Python script as `aes_gcm_encryption.py`.
2. Open your terminal or command prompt and navigate to the directory where the script is saved.
3. Run the script using the following command:

```
python aes_gcm_encryption.py
```

4. The program will:
 - o Encrypt the message using AES-GCM.
 - o Print the ciphertext, nonce, and authentication tag.
 - o Decrypt the message and verify its authenticity.

Step 5: Verifying the Results

- The script will print the ciphertext (in Base64 format), nonce, and the authentication tag.
- The message will be decrypted using the same key and nonce. If the decryption is successful and the message has not been tampered with, the original plaintext message will be displayed.
- If the message has been altered in any way or if the authentication tag does not match, the decryption will fail, and an error message will be displayed.

Expected Outcome:

- The script will successfully encrypt and decrypt a message using AES-GCM.
 - The ciphertext, nonce, and authentication tag will be printed in Base64 format.
 - The decrypted message will match the original plaintext if no tampering occurred.
 - If the message or tag is altered, the decryption will fail, indicating tampering.
-

24. Develop a Python program to simulate a ransomware attack by encrypting files in a folder and then decrypting them with a key.

Aim: To develop a Python program that simulates a ransomware attack by encrypting files in a folder and then decrypting them using a key.

Software & Tools Required:

- Python 3.x
- cryptography library (for encryption and decryption)
- Text editor (VSCode, PyCharm, etc.)
- Command line or terminal

Experiment Steps:

Step 1: Understanding Ransomware Simulation

- Ransomware is a type of malware that encrypts a victim's files and demands payment for decryption.
- The simulation will include:
 1. Encrypting files in a specified folder.
 2. Storing the encryption key securely.
 3. Decrypting the files when provided with the correct key.

Step 2: Installing Required Libraries Install the `cryptography` library to handle encryption and decryption:

```
pip install cryptography
```

Step 3: Writing the Python Script for Ransomware Simulation

```
import os
from cryptography.fernet import Fernet

def generate_key():
    key = Fernet.generate_key()
    with open("encryption_key.key", "wb") as key_file:
        key_file.write(key)
    return key

def load_key():
    return open("encryption_key.key", "rb").read()

def encrypt_files(folder_path, key):
    cipher = Fernet(key)
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path):
            with open(file_path, "rb") as file:
                file_data = file.read()
                encrypted_data = cipher.encrypt(file_data)
```



```

        with open(file_path, "wb") as file:
            file.write(encrypted_data)
    print("All files have been encrypted.")

def decrypt_files(folder_path, key):
    cipher = Fernet(key)
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        if os.path.isfile(file_path):
            with open(file_path, "rb") as file:
                encrypted_data = file.read()
                decrypted_data = cipher.decrypt(encrypted_data)
            with open(file_path, "wb") as file:
                file.write(decrypted_data)
    print("All files have been decrypted.")

if __name__ == "__main__":
    folder = "test_folder" # Change to your target folder
    os.makedirs(folder, exist_ok=True)

    option = input("Enter 'E' to encrypt or 'D' to decrypt: ").upper()

    if option == 'E':
        key = generate_key()
        encrypt_files(folder, key)
    elif option == 'D':
        key = load_key()
        decrypt_files(folder, key)
    else:
        print("Invalid option. Use 'E' for encrypt or 'D' for decrypt.")

```

Step 4: Running the Program

1. Save the script as `ransomware_simulator.py`.
2. Create a test folder (`test_folder`) and place sample files in it.
3. Open the terminal, navigate to the script's directory, and run:

```
python ransomware_simulator.py
```

4. Choose encryption („E“) to encrypt all files.
5. Run the script again and choose decryption („D“) to restore the files.

Step 5: Verifying the Results

- The script should encrypt all files in the specified folder.
- Running the script with decryption should restore the files to their original state.
- The encryption key must be stored securely to enable decryption.

Expected Outcome:

- The Python script will encrypt and decrypt files successfully using the provided key.
- Users will understand the working of ransomware and the importance of key management in cybersecurity.

25. Write a Python-based firewall rule tester that sends test packets and analyzes blocked/allowed responses.

Aim: To write a Python-based firewall rule tester that sends test packets and analyzes blocked/allowed responses.

Required Tools:

- Python 3.x
- `scapy` library for sending test packets

Steps:

1. Install `scapy`:

```
pip install scapy
```

2. Write a Python script to send test packets:

```
from scapy.all import *

def test_firewall(destination_ip):
    print(f"Sending test packet to {destination_ip}")
    packet = IP(dst=destination_ip)/ICMP()
    response = sr1(packet, timeout=2, verbose=0)

    if response:
        print("Packet Allowed: Firewall is not blocking ICMP requests.")
    else:
        print("Packet Blocked: Firewall is blocking ICMP requests.")

if __name__ == "__main__":
    target_ip = input("Enter the target IP: ")
    test_firewall(target_ip)
```

3. Run the script and analyze responses.

Expected Outcome:

- Identifies whether ICMP packets are being blocked by the firewall.
- Helps in understanding firewall rules and configurations.

26. Perform encryption, decryption using the following substitution techniques

- Ceaser cipher
- Playfair cipher
- Hill Cipher
- Vigenere cipher

26a). Encryption and Decryption Using Ceaser Cipher

AIM:

To encrypt and decrypt the given message by using Ceaser Cipher encryption algorithm.

ALGORITHMS:

1. In Ceaser Cipher each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
2. For example, with a **left shift of 3**, **D** would be replaced by **A**, **E** would become **B**, and so on.
3. The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, **A = 0, B = 1, Z = 25**.
4. Encryption of a letter x by a shift n can be described mathematically as,

$$En(x) = (x + n) \bmod 26$$

5. Decryption is performed similarly,

$$Dn(x) = (x - n) \bmod 26$$

PROGRAM:

CaesarCipher.java

```
class caesarCipher {
    public static String encode(String enc, int offset) { offset =
        offset % 26 + 26;
        StringBuilder encoded = new StringBuilder(); for (char i
        : enc.toCharArray()) {
            if (Character.isLetter(i)) {
                if (Character.isUpperCase(i)) {
                    encoded.append((char) ('A' + (i - 'A' + offset) % 26));
                } else {
                    encoded.append((char) ('a' + (i - 'a' + offset) % 26));
                }
            } else {
                encoded.append(i);
            }
        }
    }
}
```

```

        }

    }

    return encoded.toString();
}

public static String decode(String enc, int offset) { return
    encode(enc, 26 - offset);
}

public static void main(String[] args) throws java.lang.Exception { String msg
    = "Anna University";
    System.out.println("Simulating Caesar Cipher\n----- ");
    System.out.println("Input : " + msg); System.out.printf("Encrypted Message
    : "); System.out.println(caesarCipher.encode(msg, 3));
    System.out.printf("Decrypted Message : ");
    System.out.println(caesarCipher.decode(caesarCipher.encode(msg, 3), 3));
}
}

```

OUTPUT:

Simulating Caesar Cipher

Input : Anna University
 Encrypted Message : Dqqd Xqlyhuvlwb
 Decrypted Message : Anna University

RESULT:

Thus the program for ceaser cipher encryption and decryption algorithm has been implemented and the output verified successfully.

26b). Playfair Cipher

AIM:

To implement a program to encrypt a plain text and decrypt a cipher text using play fair cipher substitution technique.

ALGORITHM:

1. To encrypt a message, one would break the message into digrams (groups of 2 letters)
2. For example, "HelloWorld" becomes "HELLO WORLD".
3. These digrams will be substituted using the key table.
4. Since encryption requires pairs of letters, messages with an odd number of characters usually append an uncommon letter, such as "X", to complete the final digram.
5. The two letters of the digram are considered opposite corners of a rectangle in the key table. To perform the substitution, apply the following 4 rules, in order, to each pair of letters in the plaintext:

PROGRAM:

playfairCipher.java

```
import java.awt.Point;
```

```
class playfairCipher {
    private static char[][] charTable; private
    static Point[] positions;

    private static String prepareText(String s, boolean chgJtoI) { s =
        s.toUpperCase().replaceAll("[^A-Z]", "");
        return chgJtoI ? s.replace("J", "I") : s.replace("Q", "");
    }

    private static void createTbl(String key, boolean chgJtoI) { charTable = new
        char[5][5];

        positions = new Point[26];
        String s = prepareText(key + "ABCDEFGHIJKLMNOPQRSTUVWXYZ", chgJtoI);
        int len = s.length();
        for (int i = 0, k = 0; i < len; i++) { char c
            = s.charAt(i);

            if (positions[c - 'A'] == null) {
                charTable[k / 5][k % 5] = c;
                positions[c - 'A'] = new Point(k % 5, k / 5); k++;
            }
        }
    }
}
```

```

private static String codec(StringBuilder txt, int dir) { int len =
    txt.length();
    for (int i = 0; i < len; i += 2) { char a =
        txt.charAt(i);
        char b = txt.charAt(i + 1);
        int row1 = positions[a - 'A'].y; int
        row2 = positions[b - 'A'].y; int col1
        = positions[a - 'A'].x; int col2 =
        positions[b - 'A'].x; if (row1 ==
        row2) {
            col1 = (col1 + dir) % 5; col2
            = (col2 + dir) % 5;
        } else if (col1 == col2) {

            row1 = (row1 + dir) % 5; row2 =
            (row2 + dir) % 5;
        } else {
            int tmp = col1;
            col1 = col2; col2
            = tmp;
        }

        txt.setCharAt(i, charTable[row1][col1]);
        txt.setCharAt(i + 1, charTable[row2][col2]);

    }

    return txt.toString();
}

```

```

private static String encode(String s) {
    StringBuilder sb = new StringBuilder(s); for
    (int i = 0; i < sb.length(); i += 2) {
        if (i == sb.length() - 1) { sb.append(sb.length() % 2
            == 1 ? 'X' : "");
        } else if (sb.charAt(i) == sb.charAt(i + 1)) {
            sb.insert(i + 1, 'X');
        }
    }

    return codec(sb, 1);
}

```

```

private static String decode(String s) { return
    codec(new StringBuilder(s), 4);
}

```

```

public static void main(String[] args) throws java.lang.Exception { String key =
    "CSE";
    String txt = "Security Lab"; /* make sure string length is even */ /* change J to I */
    boolean chgJtoI = true; createTbl(key,
    chgJtoI);
    String enc = encode(prepareText(txt, chgJtoI)); System.out.println("Simulating
    Playfair Cipher\n ----- ");
    System.out.println("Input Message : " + txt); System.out.println("Encrypted
    Message : " + enc); System.out.println("Decrypted Message : " +
    decode(enc));
}
}

```

OUTPUT:

Simulating Playfair Cipher

Input Message : Security Lab

Encrypted Message : EABPUGYANSEZ
Decrypted
Message : SECURITYLABX

RESULT:

Thus the program for playfair cipher encryption and decryption algorithm has been implemented and the output verified successfully.

26c). Hill Cipher

AIM:

To implement a program to encrypt and decrypt using the Hill cipher substitution technique

ALGORITHM:

1. In the Hill cipher Each letter is represented by a number modulo 26.
2. To encrypt a message, each block of n letters is multiplied by an invertible $n \times n$ matrix, again **modulus 26**.
3. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.
4. The matrix used for encryption is the cipher key, and it should be chosen randomly from the **set of invertible $n \times n$ matrices (modulo 26)**.
5. The cipher can, be adapted to an alphabet with any number of letters.
6. All arithmetic just needs to be done modulo the number of letters instead of modulo 26.

PROGRAM:

HillCipher.java

```
class hillCipher {
    /* 3x3 key matrix for 3 characters at once */
    public static int[][] keymat = new int[][] { { 1, 2, 1 }, { 2, 3, 2 },
        { 2, 2, 1 } }; /* key inverse matrix */
    public static int[][] invkeymat = new int[][] { { -1, 0, 1 }, { 2, -1, 0 }, { -2, 2, -1
    } };

    public static String key = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static String encode(char a, char b, char c) { String
        ret = "";
        int x, y, z;
        int posa = (int) a - 65; int
        posb = (int) b - 65; int
        posc = (int) c - 65;
        x = posa * keymat[0][0] + posb * keymat[1][0] + posc * keymat[2][0];
        y = posa * keymat[0][1] + posb * keymat[1][1] + posc * keymat[2][1];
        z = posa * keymat[0][2] + posb * keymat[1][2] + posc * keymat[2][2];
        a = key.charAt(x % 26);
        b = key.charAt(y % 26);
        c = key.charAt(z % 26); ret
        = "" + a + b + c; return ret;
    }
}
```



```

private static String decode(char a, char b, char c) { String
    ret = "";
    int x, y, z;
    int posa = (int) a - 65; int
    posb = (int) b - 65; int
    posc = (int) c - 65;
    x = posa * invkeymat[0][0] + posb * invkeymat[1][0] + posc *
invkeymat[2][0];
    y = posa * invkeymat[0][1] + posb * invkeymat[1][1] + posc *
invkeymat[2][1];
    z = posa * invkeymat[0][2] + posb * invkeymat[1][2] + posc *
invkeymat[2][2];
    a = key.charAt((x % 26 < 0) ? (26 + x % 26) : (x % 26));

    b = key.charAt((y % 26 < 0) ? (26 + y % 26) : (y % 26));
    c = key.charAt((z % 26 < 0) ? (26 + z % 26) : (z % 26)); ret = "" + a
    + b + c;
    return ret;
}

```

```

public static void main(String[] args) throws java.lang.Exception {

```

```

    String msg; String
    enc = ""; String dec
    = ""; int n;
    msg = ("SecurityLaboratory");
    System.out.println("simulation of Hill Cipher\n ----- ");
    System.out.println("Input message : " + msg); msg =
    msg.toUpperCase();
    msg = msg.replaceAll("\\s", "");
    /* remove spaces */ n = msg.length() % 3;
    /* append padding text X */ if (n != 0) { for (int
        i = 1; i <= (3 - n); i++) {
            msg += 'X';
        }
    }

    System.out.println("padded message : " + msg); char[]
    pdchars = msg.toCharArray();
    for (int i = 0; i < msg.length(); i += 3) {
        enc += encode(pdchars[i], pdchars[i + 1], pdchars[i + 2]);
    }
}

```

```
System.out.println("encoded message : " + enc); char[]  
dechars = enc.toCharArray();  
for (int i = 0; i < enc.length(); i += 3) {  
    dec += decode(dechars[i], dechars[i + 1], dechars[i + 2]);  
}  
  
System.out.println("decoded message : " + dec);  
}  
}
```

OUTPUT:

Simulating Hill Cipher

Input Message : SecurityLaboratory
Padded Message : SECURITYLABORATORY Encrypted
Message : EACSDKLCAEFQDUKSXU Decrypted Message
: SECURITYLABORATORY

RESULT:

Thus the program for hill cipher encryption and decryption algorithm has been implemented and the output verified successfully.

26d). Vigenere Cipher

AIM:

To implement a program for encryption and decryption using vigenere cipher substitution technique

ALGORITHM:

1. The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword.
2. It is a simple form of *polyalphabetic* substitution.
3. To encrypt, a table of alphabets can be used, termed a Vigenere square, or Vigenere table.
4. It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers.
5. At different points in the encryption process, the cipher uses a different alphabet from one of the rows used.
6. The alphabet at each point depends on a repeating keyword.

PROGRAM:

vigenereCipher.java

```
public class vigenereCipher {
    static String encode(String text, final String key) { String
        res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++) { char c
            = text.charAt(i);
            if (c < 'A' || c > 'Z') { continue;
            }

            res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A'); j = ++j
            % key.length();
        }

        return res;
    }

    static String decode(String text, final String key) { String
        res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++) { char c
            = text.charAt(i);
            if (c < 'A' || c > 'Z') { continue;
            }
        }
```

```

        res += (char) ((c - key.charAt(j) + 26) % 26 + 'A'); j = ++j
        % key.length();
    }

    return res;
}

public static void main(String[] args) throws java.lang.Exception { String key =
    "VIGENERECIPHER";
    String msg = "SecurityLaboratory";
    System.out.println("Simulating Vigenere Cipher\n----- ");
    System.out.println("Input Message : " + msg); String enc =
    encode(msg, key); System.out.println("Encrypted Message :
    " + enc);
    System.out.println("Decrypted Message : " + decode(enc, key));
}
}

```

OUTPUT:

Simulating Vigenere Cipher

Input Message : SecurityLaboratory
 Encrypted Message : NMIYEMKCNIQVVROWXC
 Decrypted Message : SECURITYLABORATORY

RESULT:

Thus the program for vigenere cipher encryption and decryption algorithm has been implemented and the output verified successfully.

27. Perform encryption and decryption using following transposition techniques

- Rail fence
- Row & Column Transformation

27a). Rail Fence Cipher Transposition Technique

AIM:

To implement a program for encryption and decryption using rail fence transposition technique.

ALGORITHM:

1. In the rail fence cipher, the plaintext is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail.
2. When we reach the top rail, the message is written downwards again until the whole plaintext is written out.
3. The message is then read off in rows.

PROGRAM:

railFenceCipher.java

```
class railfenceCipherHelper
{
```

```
    int depth;
```

```
    String encode(String msg, int depth) throws Exception { int r =
```

```
        depth;
```

```
        int l = msg.length(); int
```

```
        c = l / depth;
```

```
        int k = 0;
```

```
        char mat[][] = new char[r][c]; String enc
```

```
        = "";
```

```
        for (int i = 0; i < c; i++) { for
```

```
            (int j = 0; j < r; j++) {
```

```
                if (k != l) {
```

```
                    mat[j][i] = msg.charAt(k++);
```

```
                } else {
```

```
                    mat[j][i] = 'X';
```

```
                }
```

```
            }
```

```
        }
```

```
        for (int i = 0; i < r; i++) { for
```

```
            (int j = 0; j < c; j++) {
```

```

        enc += mat[i][j];
    }
}

return enc;
}

```

String decode(String encmsg, int depth) throws Exception { int r =

```

    depth;
    int l = encmsg.length(); int c
    = l / depth;
    int k = 0;
    char mat[][] = new char[r][c]; String dec
    = "";
    for (int i = 0; i < r; i++) { for
        (int j = 0; j < c; j++) {
            mat[i][j] = encmsg.charAt(k++);
        }
    }

    for (int i = 0; i < c; i++) { for
        (int j = 0; j < r; j++) {
            dec += mat[j][i];
        }
    }

    return dec;
}
}

```

class railFenceCipher

```

{
    public static void main(String[] args) throws java.lang.Exception { railfenceCipherHelper rf =
        new railfenceCipherHelper();
        String msg, enc, dec;
        msg = "Anna University, Chennai"; int
        depth = 2;
        enc = rf.encode(msg, depth); dec =
        rf.decode(enc, depth);
        System.out.println("Simulating Railfence Cipher\n ----- ");
    }
}

```

```
        System.out.println("Input Message : " + msg);
        System.out.println("Encrypted Message : " + enc);
        System.out.printf("Decrypted Message : " + dec);
    }
}
```

OUTPUT:

Simulating Railfence Cipher

Input Message : Anna University, Chennai Encrypted
Message : An nvriy hnanaUiest,Ceni Decrypted
Message : Anna University, Chennai

RESULT:

Thus the java program for Rail Fence Transposition Technique has been implemented and the output verified successfully.

27b). Row and Column Transformation Technique

AIM:

To implement a program for encryption and decryption by using row and column transformation technique.

ALGORITHM:

1. Consider the plain text hello world, and let us apply the simple columnar transposition technique as shown below

h	e	l	l
---	---	---	---

o	w	o	r
l	d		

2. The plain text characters are placed horizontally and the cipher text is created with vertical format as: **holewdlo lr**.
3. Now, the receiver has to use the same table to decrypt the cipher text to plain text.

PROGRAM:

TransCipher.java

```
import java.util.*; class
TransCipher {
    public static void main(String args[]) { Scanner sc =
        new Scanner(System.in);
        System.out.println("Enter the plain text"); String
        pl = sc.nextLine();
        sc.close(); String
        s = ""; int start =
        0;
        for (int i = 0; i < pl.length(); i++) { if
            (pl.charAt(i) == ' ') {
                s = s + pl.substring(start, i); start =
                i + 1;
            }
        }

        s = s + pl.substring(start);
        System.out.print(s); System.out.println();
        // end of space deletion
```



```

int k = s.length(); int
l = 0;
int col = 4;
int row = s.length() / col;
char ch[][] = new char[row][col]; for (int
i = 0; i < row; i++) {

    for (int j = 0; j < col; j++) { if (l
    < k) {
        ch[i][j] = s.charAt(l); l++;
    } else {
        ch[i][j] = '#';
    }

    }

}

// arranged in matrix

char trans[][] = new char[col][row]; for
(int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) { trans[j][i]
    = ch[i][j];
    }

}

for (int i = 0; i < col; i++) { for (int
j = 0; j < row; j++) {
    System.out.print(trans[i][j]);
    }

}

// display
System.out.println();
}

}

```

OUTPUT:

Enter the plain text
Security Lab
SecurityLab Sreictuy

RESULT:

Thus the java program for Row and Column Transposition Technique has been implemented and the output verified successfully.

28. Apply DES algorithm for practical applications.

AIM:

To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption.

ALGORITHM:

1. Create a DES Key.
2. Create a Cipher instance from Cipher class, specify the following information and separated by a slash (/).
 - a. Algorithm name
 - b. Mode (optional)
 - c. Padding scheme (optional)
3. Convert String into **Byte[]** array format.
4. Make Cipher in encrypt mode, and encrypt it with **Cipher.doFinal()** method.
5. Make Cipher in decrypt mode, and decrypt it with **Cipher.doFinal()** method.

PROGRAM:

DES.java

```
import java.security.InvalidKeyException; import
java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException; import
javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException; import
javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException; import
javax.crypto.SecretKey;

public class DES
{

    public static void main(String[] argv) {

        try{
            System.out.println("Message Encryption Using DES Algorithm\n -----");
            KeyGenerator keygenerator = KeyGenerator.getInstance("DES"); SecretKey
            myDesKey = keygenerator.generateKey();
            Cipher desCipher;
            desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            desCipher.init(Cipher.ENCRYPT_MODE, myDesKey); byte[] text =
            "Secret Information ".getBytes(); System.out.println("Message [Byte
            Format] : " + text);
```

```

System.out.println("Message : " + new String(text)); byte[] textEncrypted =
desCipher.doFinal(text);
        System.out.println("Encrypted Message: " + textEncrypted);
        desCipher.init(Cipher.DECRYPT_MODE, myDesKey); byte[]
        textDecrypted = desCipher.doFinal(textEncrypted);
        System.out.println("Decrypted Message: " + new
String(textDecrypted));

        }catch(NoSuchAlgorithmException e){
            e.printStackTrace();
        }catch(NoSuchPaddingException e){
            e.printStackTrace();
        }catch(InvalidKeyException e){
            e.printStackTrace();
        }catch(IllegalBlockSizeException e){
            e.printStackTrace();
        }catch(BadPaddingException e){
            e.printStackTrace();
        }
    }
}

```

OUTPUT:

Message Encryption Using DES Algorithm

```

Message [Byte Format] : [B@4dcbadb4
Message : Secret Information Encrypted
Message: [B@504bae78 Decrypted Message:
Secret Information

```

RESULT:

Thus the java program for DES Algorithm has been implemented and the output verified successfully.

29. Apply AES algorithm for practical applications.

AIM:

To use Advanced Encryption Standard (AES) Algorithm for a practical application like URL Encryption.

ALGORITHM:

1. AES is based on a design principle known as a substitution–permutation.
2. AES does not use a Feistel network like DES, it uses variant of Rijndael.
3. It has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits.
4. AES operates on a 4×4 column-major order array of bytes, termed the state

PROGRAM:

AES.java

```
import java.io.UnsupportedEncodingException; import
java.security.MessageDigest;
import java.security.NoSuchAlgorithmException; import
java.util.Arrays;
import java.util.Base64;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec; public

class AES {

    private static SecretKeySpec secretKey; private
    static byte[] key;

    public static void setKey(String myKey) { MessageDigest
        sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1"); key =
            sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        } catch (NoSuchAlgorithmException e) {
```

```

        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

public static String encrypt(String strToEncrypt, String secret) { try {
    setKey(secret);
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF
-8"))));
    } catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }

    return null;
}

public static String decrypt(String strToDecrypt, String secret) { try {
    setKey(secret);
    Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
    cipher.init(Cipher.DECRYPT_MODE, secretKey);
    return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
    } catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }

    return null;
}

public static void main(String[] args) {
    final String secretKey = "annaUniversity";

    String originalString = "www.annauniv.edu";
    String encryptedString = AES.encrypt(originalString, secretKey);

```

```
String decryptedString = AES.decrypt(encryptedString, secretKey);

System.out.println("URL Encryption Using AES Algorithm\n -----");
System.out.println("Original URL : " + originalString); System.out.println("Encrypted URL : "
+ encryptedString); System.out.println("Decrypted URL : " + decryptedString);
}
}
```

OUTPUT:

URL Encryption Using AES Algorithm

Original URL : www.annauniv.edu

Encrypted URL : vibpFJW6Cvs5Y+L7t4N6YWWe07+JzS1d3CU2h3mEvEg= Decrypted URL :
www.annauniv.edu

RESULT:

Thus the java program for AES Algorithm has been implemented for URL Encryption and the output verified successfully.

30. Implement RSA Algorithm using HTML and JavaScript

AIM:

To implement RSA (Rivest–Shamir–Adleman) algorithm by using HTML and Javascript.

ALGORITHM:

1. Choose two prime number p and q
2. Compute the value of n and p
3. Find the value of e (public key)
4. Compute the value of d (private key) using $\text{gcd}()$
5. Do the encryption and decryption
 - a. Encryption is given as,
$$c = t^e \bmod n$$
 - b. Decryption is given as,
$$t = c^d \bmod n$$

PROGRAM:

rsa.html

```
<html>

<head>
  <title>RSA Encryption</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
  <center>
    <h1>RSA Algorithm</h1>
    <h2>Implemented Using HTML & Javascript</h2>
    <hr>
    <table>
      <tr>
```


Enter First Prime Number:	<input id="p" type="number" value="53"/>
Enter Second Prime Number:	<input id="q" type="number" value="59"/>
Enter the Message(cipher text): [A=1, B=2,...]	<input id="msg" type="number" value="89"/>
Public Key:	<div id="publickey"></div>
Exponent:	<div id="exponent"></div>
Private Key:	<div id="privatekey"></div>
Cipher Text:	<div id="ciphertext"></div>

```

        <td><button onclick="RSA();">Apply RSA</button></td>
    </tr>
</table>
</center>
</body>
<script type="text/javascript">
    function RSA() {
        var gcd, p, q, no, n, t, e, i, x;
        gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); };
        p = document.getElementById('p').value;
        q = document.getElementById('q').value;
        no = document.getElementById('msg').value; n = p *
        q;
        t = (p - 1) * (q - 1);

        for (e = 2; e < t; e++) { if
            (gcd(e, t) == 1) {
                break;
            }
        }

        for (i = 0; i < 10; i++) { x =
            1 + i * t
            if (x % e == 0) { d =
                x / e; break;
            }
        }

        ctt = Math.pow(no, e).toFixed(0); ct =
        ctt % n;

        dtt = Math.pow(ct, d).toFixed(0); dt =
        dtt % n;

        document.getElementById('publickey').innerHTML = n;
        document.getElementById('exponent').innerHTML = e;
        document.getElementById('privatekey').innerHTML = d;
    }
</script>

```

```
        document.getElementById('ciphertext').innerHTML = ct;
    }
</script>
</html>
```

OUTPUT:

RSA Algorithm

Implemented Using HTML & Javascript

Enter First Prime Number:	<input type="text" value="53"/>
Enter Second Prime Number:	<input type="text" value="59"/>
Enter the Message(cipher text): [A=1, B=2,...]	<input type="text" value="89"/>
Public Key:	3127
Exponent:	3
Private Key:	2011
Cipher Text:	1394
<input type="button" value="Apply RSA"/>	

RESULT:

Thus the RSA algorithm has been implemented using HTML & CSS and the output has been verified successfully.

31. Implement the Diffie-Hellman Key Exchange algorithm for a given problem.

AIM:

To implement the Diffie-Hellman Key Exchange algorithm for a given problem .

ALGORITHM:

1. Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $a = 4$, then sends Bob $A = g^a \bmod p$
 - o $A = 5^4 \bmod 23 = 4$
3. Bob chooses a secret integer $b = 3$, then sends Alice $B = g^b \bmod p$
 - o $B = 5^3 \bmod 23 = 10$
4. Alice computes $s = B^a \bmod p$
 - o $s = 10^4 \bmod 23 = 18$
5. Bob computes $s = A^b \bmod p$
 - o $s = 4^3 \bmod 23 = 18$
6. Alice and Bob now share a secret (the number 18).

PROGRAM:

DiffieHellman.java

```
class DiffieHellman {
    public static void main(String args[]) {
        int p = 23; /* publicly known (prime number) */

        int g = 5; /* publicly known (primitive root) */ int x = 4;
        /* only Alice knows this secret */
        int y = 3; /* only Bob knows this secret */ double
        aliceSends = (Math.pow(g, x)) % p;
        double bobComputes = (Math.pow(aliceSends, y)) % p; double
        bobSends = (Math.pow(g, y)) % p;
        double aliceComputes = (Math.pow(bobSends, x)) % p; double
        sharedSecret = (Math.pow(g, (x * y))) % p;
        System.out.println("simulation of Diffie-Hellman key exchange algorithm\n--
        -----");

        System.out.println("Alice Sends : " + aliceSends); System.out.println("Bob
        Computes : " + bobComputes); System.out.println("Bob Sends : " +
        bobSends); System.out.println("Alice Computes : " + aliceComputes);
```

```
        System.out.println("Shared Secret : " + sharedSecret);
        /* shared secrets should match and equality is transitive */
        if ((aliceComputes == sharedSecret) && (aliceComputes == bobComputes))
            System.out.println("Success: Shared Secrets Matches! " + sharedSecret);
        else
            System.out.println("Error: Shared Secrets does not Match");
    }
}
```

OUTPUT:

simulation of Diffie-Hellman key exchange algorithm

```
Alice Sends : 4.0 Bob
Computes : 18.0 Bob
Sends : 10.0
Alice Computes : 18.0 Shared
Secret : 18.0
Success: Shared Secrets Matches! 18.0
```

RESULT:

Thus the *Diffie-Hellman key exchange algorithm* has been implemented using Java Program and the output has been verified successfully.

32. Calculate the message digest of a text using the SHA-1 algorithm.

AIM:

To Calculate the message digest of a text using the SHA-1 algorithm.

ALGORITHM:

1. Append Padding Bits
2. Append Length - 64 bits are appended to the end
3. Prepare Processing Functions
4. Prepare Processing Constants
5. Initialize Buffers
6. Processing Message in 512-bit blocks (L blocks in total message)

PROGRAM:

sha1.java

```
import java.security.*;
```

```
public class sha1 {
    public static void main(String[] a) { try {

        MessageDigest md = MessageDigest.getInstance("SHA1");
        System.out.println("Message digest object info:\n-----");
        System.out.println("Algorithm=" + md.getAlgorithm()); System.out.println("Provider="
+ md.getProvider()); System.out.println("ToString=" + md.toString());
        String input = "";
        md.update(input.getBytes()); byte[]
        output = md.digest();
        System.out.println();
        System.out.println("SHA1(\"" + input + "\")=" + bytesToHex(output)); input = "abc";
        md.update(input.getBytes()); output =
        md.digest(); System.out.println();
        System.out.println("SHA1(\"" + input + "\")=" + bytesToHex(output)); input =
        "abcdefghijklmnopqrstuvwxyz";
        md.update(input.getBytes()); output =
        md.digest(); System.out.println();
        System.out.println("SHA1(\"" + input + "\")=" + bytesToHex(output)); System.out.println();
    } catch (Exception e) { System.out.println("Exception:"
+ e);
    }
}
```

```

private static String bytesToHex(byte[] b) {
    char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
    StringBuffer buf = new StringBuffer();

    for (byte aB : b) {
        buf.append(hexDigit[(aB >> 4) & 0x0f]);
        buf.append(hexDigit[aB & 0x0f]);
    }

    return buf.toString();
}
}

```

OUTPUT:

Message digest object info:

Algorithm=SHA1 Provider=SUN

version 12

ToString=SHA1 Message Digest from SUN, <initialized>

SHA1("")=DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc")=A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz")=32D10C7B8CF96570CA04CE37F2A19 D84240D3A89

RESULT:

Thus the *Secure Hash Algorithm (SHA-1)* has been implemented and the output has been verified successfully.

33. Implement the Signature Scheme - Digital Signature Standard.

AIM:

To implement the Signature Scheme - Digital Signature Standard.

ALGORITHM:

1. Create a KeyPairGenerator object.
2. Initialize the KeyPairGenerator object.
3. Generate the KeyPairGenerator. ...
4. Get the private key from the pair.
5. Create a signature object.
6. Initialize the Signature object.
7. Add data to the Signature object
8. Calculate the Signature

PROGRAM:

```
import java.security.KeyPair;

import java.security.KeyPairGenerator;
import java.security.PrivateKey; import
java.security.Signature;
import java.util.Scanner;

public class CreatingDigitalSignature {
    public static void main(String args[]) throws Exception {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter some text"); String
        msg = sc.nextLine();

        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");

        keyPairGen.initialize(2048);

        KeyPair pair = keyPairGen.generateKeyPair();

        PrivateKey privKey = pair.getPrivate();

        Signature sign = Signature.getInstance("SHA256withDSA"); sign.initSign(privKey);
        byte[] bytes = "msg".getBytes(); sign.update(bytes);

        byte[] signature = sign.sign();
```



```
        System.out.println("Digital signature for given text: "+new String(signature, "UTF8"));
    }
}
```

OUTPUT:

Enter some text Hi

how are you

Digital signature for given text: 0=@gRD???-?.??? /yGL?i??a!?

RESULT:

Thus the Digital Signature Standard Signature Scheme has been implemented and the output has been verified successfully.

34. Demonstrate intrusion detection system (IDS) using any tool ex. Snort or any other s/w.

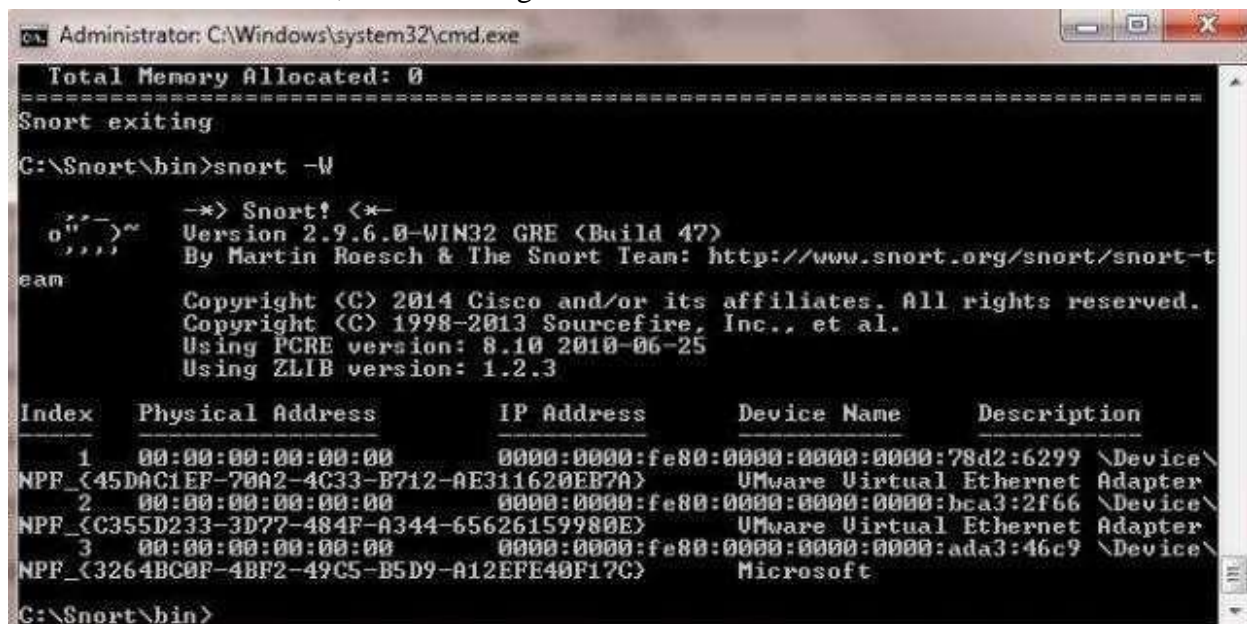
AIM:

To demonstrate Intrusion Detection System (IDS) using Snort software tool.

STEPS ON CONFIGURING AND INTRUSION DETECTION:

1. Download Snort from the Snort.org website. (<http://www.snort.org/snort-downloads>)
2. Download Rules(<https://www.snort.org/snort-rules>). You must register to get the rules. (You should download these often)
3. Double click on the .exe to install snort. This will install snort in the “C:\Snort” folder. It is important to have WinPcap (<https://www.winpcap.org/install/>) installed
4. Extract the Rules file. You will need WinRAR for the .gz file.
5. Copy all files from the “rules” folder of the extracted folder. Now paste the rules into “C:\Snort\rules” folder.
6. Copy “snort.conf” file from the “etc” folder of the extracted folder. You must paste it into “C:\Snort\etc” folder. Overwrite any existing file. Remember if you modify your snort.conf file and download a new file, you must modify it for Snort to work.
7. Open a command prompt (cmd.exe) and navigate to folder “C:\Snort\bin” folder. (at the Prompt, type cd\snort\bin)
8. To start (execute) snort in sniffer mode use following command: snort -dev -i 3
-i indicates the interface number. You must pick the correct interface number. In my case, it is 3.
-dev is used to run snort to capture packets on your network.

To check the interface list, use following command: snort -w



```
Administrator: C:\Windows\system32\cmd.exe
Total Memory Allocated: 0
=====
Snort exiting
C:\Snort\bin>snort -w

-*> Snort! <*-
o" >~ Version 2.9.6.0-WIN32 GRE (Build 47)
  >>> By Martin Roesch & The Snort Team: http://www.snort.org/snort/snort-team

Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using PCRE version: 8.10 2010-06-25
Using ZLIB version: 1.2.3

Index  Physical Address      IP Address      Device Name      Description
-----
1      00:00:00:00:00:00      0000:0000:fe80:0000:0000:0000:78d2:6299 \Device\
NPF_{45DAC1EF-70A2-4C33-B712-AE311620EB7A} VMware Virtual Ethernet Adapter
2      00:00:00:00:00:00      0000:0000:fe80:0000:0000:0000:bca3:2f66 \Device\
NPF_{C355D233-3D77-484F-A344-65626159980E} VMware Virtual Ethernet Adapter
3      00:00:00:00:00:00      0000:0000:fe80:0000:0000:0000:ada3:46c9 \Device\
NPF_{3264BC0F-4BF2-49C5-B5D9-A12EFE40F17C} Microsoft
C:\Snort\bin>
```

Finding an interface

You can tell which interface to use by looking at the Index number and finding Microsoft. As you can see in the above example, the other interfaces are for VMWare. My interface is 3.

9. To run snort in IDS mode, you will need to configure the file “snort.conf” according to your network environment.

10. To specify the network address that you want to protect in snort.conf file, look for the following line.

var HOME_NET 192.168.1.0/24 (You will normally see any here)

11. You may also want to set the addresses of DNS_SERVERS, if you have some on your network.

Example:

example snort

12. Change the RULE_PATH variable to the path of rules folder. var
RULE_PATH c:\snort\rules

path to rules

13. Change the path of all library files with the name and path on your system. and you must change the path of snort_dynamicpreprocessorvariable.

C:\Snort\lib\snort_dynamicccpreprocessor

You need to do this to all library files in the “C:\Snort\lib” folder. The old path might be:

“/usr/local/lib/...”. you will need to replace that path with your system path.

Using C:\Snort\lib

14. Change the path of the “dynamicengine” variable value in the “snort.conf” file..

Example:

dynamicengine C:\Snort\lib\snort_dynamicengine\sfe_engine.dll

15 Add the paths for “include classification.config” and “include reference.config” files.

include c:\snort\etc\classification.config include
c:\snort\etc\reference.config

16. Remove the comment (#) on the line to allow ICMP rules, if it is commented with a #.

include \$RULE_PATH/icmp.rules

17. You can also remove the comment of ICMP-info rules comment, if it is commented.

include \$RULE_PATH/icmp-info.rules

18. To add log files to store alerts generated by snort, search for the “output log” test in snort.conf and add the following line:

output alert_fast: snort-alerts.ids

19. Comment (add a #) the whitelist \$WHITE_LIST_PATH/white_list.rules and the blacklist

Change the nested_ip inner , \ to nested_ip inner #, \

20. Comment out (#) following lines:

```
#preprocessor normalize_ip4
```

```
#preprocessor normalize_tcp: ips ecn stream
```

```
#preprocessor normalize_icmp4 #preprocessor  
normalize_ip6
```

```
#preprocessor normalize_icmp6
```

21. Save the “snort.conf” file.

22. To start snort in IDS mode, run the following command:

```
snort -c c:\snort\etc\snort.conf -l c:\snort\log -i 3 (Note: 3 is  
used for my interface card)
```

If a log is created, select the appropriate program to open it. You can use WordPard or NotePad++ to read the file.

To generate Log files in ASCII mode, you can use following command while running snort in IDS mode:

```
snort -A console -i3 -c c:\Snort\etc\snort.conf -l c:\Snort\log -K ascii
```

23. Scan the computer that is running snort from another computer by using PING or NMap (ZenMap).

After scanning or during the scan you can check the snort-alerts.ids file in the log folder to insure it is logging properly. You will see IP address folders appear.

Snort monitoring traffic –

```
Administrator: C:\Windows\system32\cmd.exe - snort -A console -i3 -c c:\Snort\etc\snort.conf -l c:\Snort\var\log
Rules Engine: SF_SMORT_DETECTION_ENGINE Version 2.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_FIPIELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Commencing packet processing (pid=2164)
03/29-23:53:16.033913 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56506
03/29-23:53:16.035372 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56507
03/29-23:53:16.036479 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56508
03/29-23:53:16.037093 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56509
03/29-23:53:16.142921 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:302
03/29-23:53:16.194409 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56510
03/29-23:53:16.677078 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56512
03/29-23:53:16.808301 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56513
03/29-23:53:16.944237 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56514
03/29-23:53:16.948012 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56515
03/29-23:53:16.953992 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56516
03/29-23:53:16.967744 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56517
03/29-23:53:16.982649 [**] [120:3:1] <http_inspect> NO CONTENT-LENGTH OR TRANSF
ER-ENCODING IN HTTP RESPONSE [**] [Classification: Unknown Traffic] [Priority: 3
] <TCP> 192.168.1.1:80 -> 192.168.1.20:56518
```

RESULT:

Thus the Intrusion Detection System(IDS) has been demonstrated by using the Open Source Snort Intrusion Detection Tool.

35. Automated Attack and Penetration Tools Exploring N-Stalker, a Vulnerability Assessment Tool.

AIM:

To download the N-Stalker Vulnerability Assessment Tool and exploring the features.

EXPLORING N-STALKER:

- N-Stalker Web Application Security Scanner is a Web security assessment tool.
 - It incorporates with a well-known N-Stealth HTTP Security Scanner and 35,000 Web attack signature database.
 - This tool also comes in both free and paid version.
 - Before scanning the target, go to “License Manager” tab, perform the update.
 - Once update, you will note the status as up to date.
 - You need to download and install N-Stalker from www.nstalker.com.
-
1. Start N-Stalker from a Windows computer. The program is installed under Start ⇨ Programs ⇨ N-Stalker ⇨ N-Stalker Free Edition.
 2. Enter a host address or a range of addresses to scan.
 3. Click Start Scan.
 4. After the scan completes, the N-Stalker Report Manager will prompt
 5. you to select a format for the resulting report as choose Generate HTML.
 6. Review the HTML report for vulnerabilities.



Now goto “Scan Session”, enter the target URL.

In scan policy, you can select from the four options,

- Manual test which will crawl the website and will be waiting for manual attacks.
- full xss assessment
- owasp policy
- Web server infrastructure analysis.

Once, the option has been selected, next step is “Optimize settings” which will crawl the whole website for further analysis.

In review option, you can get all the information like host information, technologies used, policy name, etc.

The image shows a screenshot of the 'N-Stalker Scan Wizard' application window. The window has a title bar that says 'N-Stalker Scan Wizard'. The main content area is titled 'Start Web Application Security Scan Session' and includes a subtitle: 'You must enter an URL and choose policy. Scan Settings may be configured.' On the left side, there is a sidebar with a red and white graphic and a list of steps: 'Choose URL & Policy' (which is highlighted), 'Optimize Settings', 'Review Summary', and 'Start Scan Session'. The main area contains three sections: 'Enter Web Application URL' with a text input field containing 'www.target.com' and a hint '(E.g: http://www.example.tl/, https://www.test.tl/VirtualDirectory/, etc)'; 'Choose Scan Policy' with a dropdown menu showing '(choose one)'; 'Load Scan Session' with a dropdown menu showing '(choose one)' and a hint '(You may load scan settings from previously saved scan sessions)'; and 'Load Spider Data' with a dropdown menu showing 'Not available in N-Stalker Free Edition' and a hint '(You may load spider data from previously saved scan sessions)', followed by a checkbox labeled 'Use local cache from previously saved session (Avoid new web crawling)'. At the bottom, there are three buttons: 'Scan Settings', 'Cancel', and 'Next >>'.

N-Stalker Scan Wizard

Start Web Application Security Scan Session

You must enter an URL and choose policy. Scan Settings may be configured.

Choose URL & Policy

Optimize Settings
Review Summary
Start Scan Session

Enter Web Application URL

www.target.com
(E.g: http://www.example.tl/, https://www.test.tl/VirtualDirectory/, etc)

Choose Scan Policy

(choose one) ▼

Load Scan Session

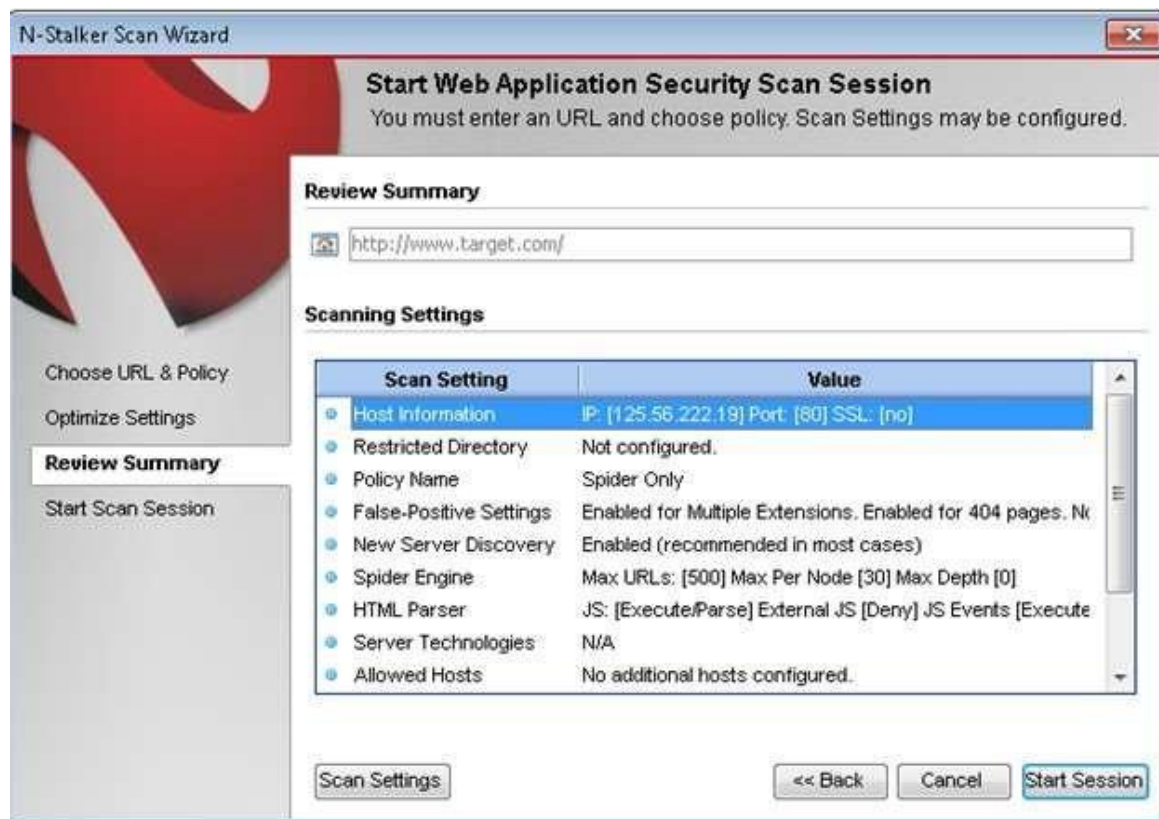
(choose one) ▼
(You may load scan settings from previously saved scan sessions)

Load Spider Data

Not available in N-Stalker Free Edition ▼
(You may load spider data from previously saved scan sessions)

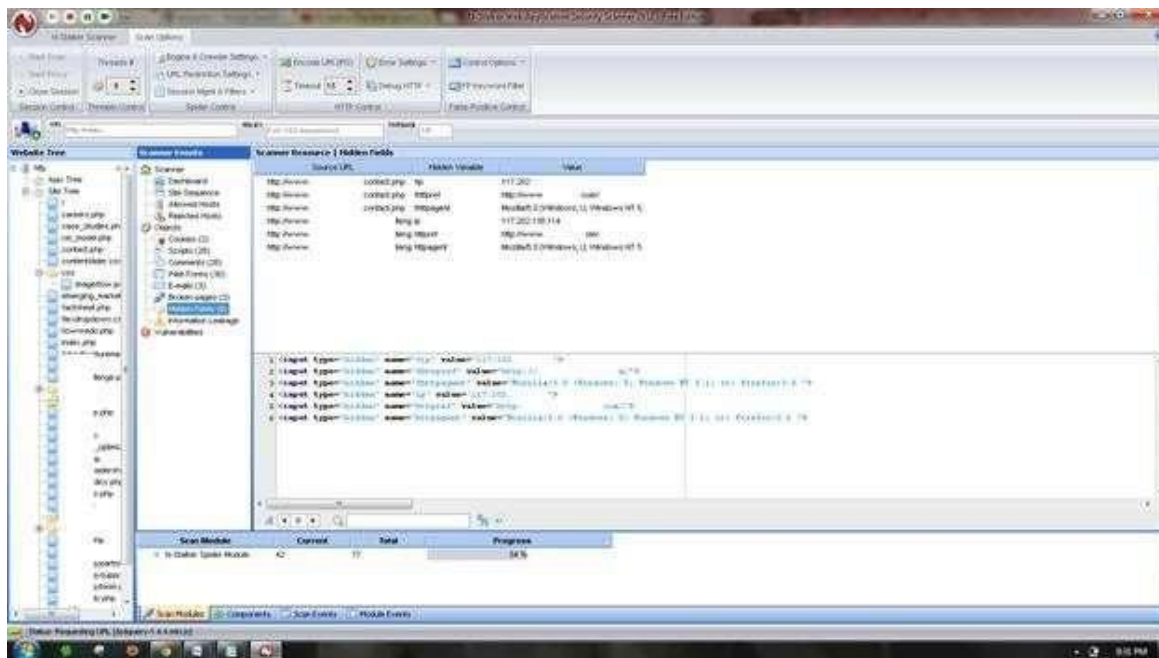
☐ Use local cache from previously saved session (Avoid new web crawling)

Scan Settings Cancel Next >>

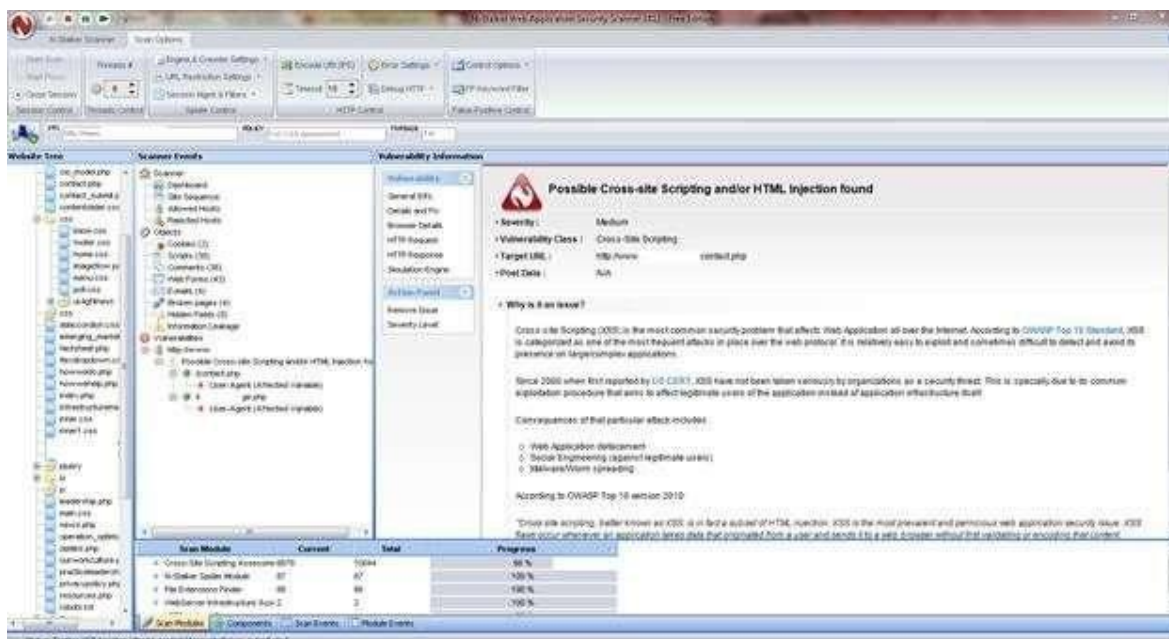


Once done, start the session and start the scan.

The scanner will crawl the whole website and will show the scripts, broken pages, hidden fields, information leakage, web forms related information which helps to analyze further.



Once the scan is completed, the NStalker scanner will show details like severity level, vulnerability class, why is it an issue, the fix for the issue and the URL which is vulnerable to the particular vulnerability?



RESULT:

Thus the N-Stalker Vulnerability Assessment tool has been downloaded, installed and the features has been explored by using a vulnerable website.