KALASALINGAM
ACADEMY OF RESEARCH AND EDUCATION
(DEEMED TO BE UNIVERSITY)
Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A++" Grade

Anand Nagar, Krishnankoil, Srivilliputtur (Via), Virudhunagar (Dt) - 626126, Tamil Nadu | info@kalasalingam.ac.in

SCHOOL OF COMPUTING

Department of Computer Science and Engineering

SECURED COMPUTING
Lab Record (*212CSE3305*)

Student Name   :……………………………………………………….

Register Number:…………………………………………………………...

Section   :……………………………………………………………

1

BONAFIDECERTIFICATE

Bonafide record of work done by _____

of _____ in _____

during odd semester in academic year 2024-2025.

StaffIn-charge

Submitted to thepractical Examination held at Kalasalingam Academy of

Research and Education, Anand Nagar, Krishnankoil on _____

REGISTERNUMBER

Internal Examiner   External Examiner

1

TABLE OF CONTENTS

S.NO	DATE	TITLE OF THE PROGRAM	Aim & Algorithm (10 Marks)	Program (20 Marks)	Output & Viva (20 marks)	Total (50 Marks)	SIGN
1		Implementation of Caesar Cipher					
2		Basic Mono alphabetic Cipher					
3		Message Authentication Code					
4		Data Encryption Standard					
5		Advanced Encryption Standard					
6		Asymmetric Key Encryption					
7		Secure Key exchange					
8		Digital Signature Generation					
9		Implementation of Mobile Security					
10		Intrusion Detection/Prevention System with Snort					
11		Defeating Malware - Building Trojans					
12		Defeating Malware - Rootkit Hunter					
13		Implement Database Security					
14		Implement Encryption and Integrity Control-DatabaseSecurity					

EX.NO:1	**Implementation of Caesar Cipher**

Aim:

To implements a Caesar cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:

1) Define Function:

Define a function encrypt_text(plaintext, n) which takes plaintext (the text to be encrypted) and n (the shift pattern) as input.

2) Initialize Answer:

Initialize an empty string ans to store the encrypted text.

3) Iterate Over Plaintext:

Loop through each character ch in plaintext using its index i.

4) Check Character Type:

If ch is a space, append a space to ans.
Else if ch is an uppercase letter:
Convert ch to its corresponding encrypted character using the formula:
encrypted_char $==((ord(ch)+n-65)\%26)+65$
Append the encrypted character to ans.
Else if ch is a lowercase letter:
Convert ch to its corresponding encrypted character using the formula:
encrypted_char $=((ord(ch)+n-97)\%26)+97$
Append the encrypted character to ans.

5) Return Encrypted Text:

Return the encrypted text stored in ans.

6) Print Results:

Define plaintext and n.
Print the original plaintext and shift pattern.
Call encrypt_text(plaintext, n) and print the resulting encrypted text.

Example

Given the plaintext = "HELLO EVERYONE" and n = 1:
Define plaintext and n.
Initialize ans as an empty string.
Loop through each character in "HELLO EVERYONE":
'H' is uppercase, so encrypt it to 'I'.
'E' is uppercase, so encrypt it to 'F'.
'L' is uppercase, so encrypt it to 'M'.
'L' is uppercase, so encrypt it to 'M'.

Program:

Output:

Result:

EX.NO:2	**Basic Monoalphabetic Cipher**

Aim:

To implements a basic monoalphabetic cipher, a type of substitution cipher, which replaces each letter in a message with another letter based on a fixed shift value.

Algorithm:

1) Define generate_cipher_key Function:
   - Input: shift (integer)
   - Initialize alphabet as a string containing 'abcdefghijklmnopqrstuvwxyz'.
   - Create shifted_alphabet by shifting alphabet by shift positions.
   - Create a dictionary key by mapping each character in alphabet to the
   - corresponding character in shifted_alphabet.
   - Return key.

2) Define encrypt Function:
   - Input: message (string), key (dictionary)
   - Initialize an empty string encrypted_message.
   - Loop through each character char in message:
   - If char is alphabetic:
   - If char is lowercase, append key[char] to encrypted_message.
   - If char is uppercase, append key[char.lower()].upper() to encrypted_message.
   - Else, append char to encrypted_message.
   - Return encrypted_message.

3) Define decrypt Function:
   - Input: ciphertext (string), key (dictionary)
   - Create reverse_key by reversing the key dictionary.
   - Initialize an empty string decrypted_message.
   - Loop through each character char in ciphertext:
   - If char is alphabetic:
   - If char is lowercase, append reverse_key[char] to decrypted_message.
   - If char is uppercase, append reverse_key[char.lower()].upper() to decrypted_message.
   - Else, append char to decrypted_message.
   - Return decrypted_message.

4) Define main Function:
   - Prompt user to input shift value.
   - Generate key using generate_cipher_key(shift).
   - Prompt user to choose between encryption and decryption (e or d).

- If the user chooses 'e':
- Prompt for the plaintext message.
- Encrypt the plaintext using encrypt(plaintext, key).
- Print the encrypted message.
- If the user chooses 'd':
- Prompt for the ciphertext message.
- Decrypt the ciphertext using decrypt(ciphertext, key).
- Print the decrypted message.
- If the user inputs an invalid choice, print an error message.

5) Execute main Function:

If this script is run as the main module, call the main function.

Program:

Output:

Result:

EX.NO:3	
	Message Authentication Code

Aim:

      To calculate the messages digest of a text using the SHA-1 algorithm and thereby verifying data integrity

Algorithm:

1) Import Hashlib Module:

      Import the hashlib module to use various SHA hash functions.

2) Compute SHA256 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode() to convert it to bytes.
   - Pass the encoded string to hashlib.sha256() to compute the SHA256 hash.
   - Get the hexadecimal representation of the hash using result.hexdigest().
   - Print the message "The hexadecimal equivalent of SHA256 is :" followed by the hexadecimal value of the SHA256 hash.

3) Compute SHA384 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode().
   - Pass the encoded string to hashlib.sha384() to compute the SHA384 hash.
   - Get the hexadecimal representation of the hash using result.hexdigest().
   - Print the message "The hexadecimal equivalent of SHA384 is :" followed by the hexadecimal value of the SHA384 hash.

4) Compute SHA224 Hash:
   - Initialize a string str with the value "GeeksforGeeks".
   - Encode the string using str.encode().
   - Pass the encoded string to hashlib.sha224() to compute the SHA224 hash.
   - Get the hexadecimal representation of the hash using `result.hexdigest()

5) Encode and hash using SHA512:
   - Reinitialize the string: str = "GeeksforGeeks"
   - Encode the string: encoded_str = str.encode()
   - Hash the encoded string using SHA512: result = hashlib.sha512(encoded_str)
   - Print the hexadecimal equivalent

6) Encode and hash using SHA1:
   - Reinitialize the string: str = "GeeksforGeeks"
   - Encode the string: encoded_str = str.encode()
   - Hash the encoded string using SHA1: result = hashlib.sha1(encoded_str)

9

- Print the hexadecimal equivalent.

Program:

Output:

Result:

EX.NO:4	
	Data Encryption Standard

Aim:

To implement a symmetric-key block cipher algorithm known as Data Encryption Standard (DES).

Algorithm:

1) **Hexadecimal to Binary Conversion (`hex2bin`):**

- Initialize a dictionary that maps each hexadecimal digit to its 4-bit binary equivalent.
- Initialize an empty string for the binary result.
- For each character in the input hexadecimal string:
    - o   Append the corresponding binary string from the dictionary to the result.
- Return the binary result.

2) **Binary to Hexadecimal Conversion (`bin2hex`):**

- Initialize a dictionary that maps each 4-bit binary string to its hexadecimal equivalent.
- Initialize an empty string for the hexadecimal result.
- For each group of 4 bits in the input binary string:
    - o   Append the corresponding hexadecimal character from the dictionary to the result.
- Return the hexadecimal result.

3) **Binary to Decimal Conversion (`bin2dec`):**

- Initialize the decimal result to 0.
- For each bit in the input binary string, from least significant to most significant:
    - o   Multiply the bit by $2i2^i2i$ (where iii is the bit's position) and add to the decimal result.
- Return the decimal result.

4) **Decimal to Binary Conversion (`dec2bin`):**

- Convert the decimal number to its binary representation using Python's `bin` function and remove the "0b" prefix.

- If the length of the binary result is not a multiple of 4, pad with leading zeros to make it a multiple of 4.

- Return the padded binary result.

12

5) **Permute Function (`permute`)**:

- Initialize an empty string for the permutation result.
- For each position in the permutation array:
  - Append the bit from the input string at the given position to the result.
- Return the permutation result.

6) **Left Shift Function (`shift_left`)**:

- For the specified number of shifts:
  - Perform a left circular shift on the input string.
- Return the shifted string.

7) **XOR Function (`xor`)**:

- Initialize an empty string for the XOR result.
- For each bit in the input strings:
  - Append the result of the XOR operation on the corresponding bits to the result.
- Return the XOR result.

8) **Encryption Function (`encrypt`)**:

- Convert the plaintext from hexadecimal to binary using `hex2bin`.
- Perform an initial permutation using the `initial_perm` table.
- Split the permuted text into left and right halves.
- For each of the 16 rounds:
  - Expand the right half from 32 to 48 bits using the `exp_d` table.
  - XOR the expanded right half with the round key.
  - Substitute the result using the S-boxes.
  - Perform a permutation using the `per` table.
  - XOR the result with the left half.
  - Swap the left and right halves, except in the final round.
- Combine the final left and right halves.
- Perform a final permutation using the `final_perm` table.
- Return the result as binary.

9) **Key Generation**:

- Convert the key from hexadecimal to binary using `hex2bin`.
- Perform a parity bit drop using the `keyp` table to get a 56-bit key.
- Split the key into left and right halves.
- For each of the 16 rounds:
  - Perform left shifts on both halves according to the `shift_table`.
  - Combine the left and right halves.
  - Compress the key from 56 to 48 bits using the `key_comp` table.

o   Append the round key in both binary and hexadecimal form to the round key lists.

10) **Main Process**:

- Define the plaintext and key in hexadecimal format.
- Generate the round keys.
- Perform encryption using the generated round keys.
- Reverse the round keys for decryption.
- Perform decryption using the reversed round keys.

Program:

Output:

Result:

EX.NO:5	
	Advanced Encryption Standard

Aim:

　　　　To understand the need of highly secured symmetric encryption algorithm known as Advanced Encryption Standard (AES)

Algorithm:

1. **Import Libraries:**
   - o `AES` for AES encryption/decryption.
   - o `get_random_bytes` to generate a random key.
   - o `pad` and `unpad` to ensure data is of a valid block size.
2. **Encrypt Function:**
   - o Creates a new AES cipher object in CBC mode.
   - o Pads the data to be a multiple of the block size.
   - o Encrypts the padded data.
   - o Returns the initialization vector (IV) and the ciphertext.
3. **Decrypt Function:**
   - o Creates a new AES cipher object with the same IV.
   - o Decrypts the ciphertext.
   - o Unpads and decodes the decrypted data.
4. **Example Usage:**
   - o Generates a random 16-byte key.
   - o Encrypts a sample message.
   - o Prints the ciphertext in hexadecimal format.
   - o Decrypts the ciphertext.
   - o Prints the decrypted message.

Input

The input for the code consists of the plaintext message that you want to encrypt. In the example provided, the input message is hardcoded as "This is a secret message."

Output

The output of the code will consist of the ciphertext (in hexadecimal format) and the decrypted message.

Example

Let's break down the expected output when running the code:

1. **Encrypted Ciphertext:** The encrypted version of the plaintext message, displayed in hexadecimal format.
2. **Decrypted Data:** The original message after decrypting the ciphertext

Program:

Output:

Result:

EX.NO:6	
	Asymmetric Key Encryption

Aim:

      To implement the popular asymmetric key algorithm Rivest,Shamir ,Adleman (RSA)

Algorithm:

1) Input:

- Two prime numbers p and q.
- A plaintext message .
- Calculate n:

    $$n = p * q$$
    For p=53 and q=59, n = 53 * 59 = 3127

2) Calculate the totient t:

- t = (p - 1) * (q - 1)
- For p=53 and q=59, t = (53 - 1) * (59 - 1) = 52 * 58 = 3016
- Select the public key e:
- Iterate from 2 to t to find the smallest integer e such that gcd(e, t) == 1.

3) Select the public key **e:**

- Find the smallest integer e such that gcd(e, t) == 1
- In this case, e = 3 (assuming the smallest integer that satisfies the condition)

4) Select the private key d:

- Initialize j = 0.
- Increment j in a loop until (j * e) % t == 1.
- Set d = j.
- Find d such that (d * e) % t == 1
- Through iteration, if e = 3, then d = 2011 (assuming this is found through the while loop)

5) Encrypt the message:

    Calculate the ciphertext ct = (message ** e) % n.

- ct = (message ** e) % n
- For message=89, ct = (89 ** 3) % 3127 = 1394

6) Decrypt the message:

- Calculate the decrypted message mes = (ct ** d) % n.
- Print the encrypted message ct.

23

- Print the decrypted message mes
- mes = (ct ** d) % n
- For ct=1394, mes = (1394 ** 2011) % 3127 = 89

Program:

Result:

Secure Key exchange

Aim:

To securely exchange the crypto graphic keys over Internet to implement Diffie- Hellman key exchange mechanism

Algorithm:

1. **Input:**

   - p: A prime number.
   - g: A primitive root of p.
   - The user is prompted to enter a prime number p and a number g (which is a primitive root of p).

2. **Initialize Classes:**
   - **Class A:** Represents Alice and Bob.
     - __init__: Generate a random private number n for Alice/Bob.
     - publish: Calculate and return the public value g^n % p.
     - compute_secret: Compute the shared secret (gb^n) % p using another party's public value gb.
     - Represents Alice and Bob.
     - Generates a random private number n.
     - Computes and returns the public value using publish.
     - Computes the shared secret using compute_secret.

   - **Class B:** Represents Eve.
     - __init__: Generate two random private numbers a and b for Eve.
     - publish: Calculate and return the public value $g^{arr[i]}$ % p for Eve's private numbers.
     - compute_secret: Compute the shared secret $(ga^{arr[i]})$ % p using another party's public value ga.
     - Represents Eve.
     - Generates two random private numbers a and b.
     - Computes and returns the public value using publish.
     - Computes the shared secret using compute_secret

3. **Create Instances:**
   - Create an instance of A for Alice.
   - Create an instance of A for Bob.
   - Create an instance of B for Eve.
   - Instances of A are created for Alice and Bob.

- An instance of B is created for Eve.
- Private numbers selected by Alice, Bob, and Eve are printed.
- Public values are generated and printed.
- Shared secrets are computed and printed.

4. Print Private Numbers:

Print the private numbers selected by Alice, Bob, and Eve.

5. Generate Public Values:
- Calculate Alice's public value ga = g^alice.n % p.
- Calculate Bob's public value gb = g^bob.n % p.
- Calculate Eve's public values gea = g^eve.a % p and geb = g^eve.b % p.

6. Print Public Values:

Print the public values generated by Alice, Bob, and Eve.

7. Compute Shared Secrets:
- Calculate Alice's shared secret with Eve sa = gea^alice.n % p.
- Calculate Eve's shared secret with Alice sea = ga^eve.a % p.
- Calculate Bob's shared secret with Eve sb = geb^bob.n % p.
- Calculate Eve's shared secret with Bob seb = gb^eve.b % p.

8. Print Shared Secrets:

Print the shared secrets computed by Alice, Bob, and Eve.

Program:

Output:

Result:

EX.NO:8	
	Digital Signature Generation

Aim:

To authenticate a message sent over the Internet using digital signature mechanism

Algorithm:

1) Generate RSA Keys:
   - The RSA key pair (private and public keys) is generated with a keysize of 2048 bits.
   - The keys are saved to files private.pem and public.pem.

2) Sign Message Function:
   - Imports the private key.
   - Creates a SHA-256 hash of the message.
   - Signs the hash using pkcs1_15 with the private key.
   - Returns the signature.

3) Verify Signature Function:
   - Imports the public key.
   - Creates a SHA-256 hash of the message.
   - Verifies the signature using pkcs1_15 with the public key.
   - Returns True if the signature is valid, otherwise returns False.

Example Usage:

The message "This is a secret message." is signed with the private key.

The signature is printed in hexadecimal format.

The signature is verified with the public key, and the result is printed.

Program:

Output:

Result:

EX.NO:9	
	Implementation of Mobile Security

Aim:

     To implements basic mobile security functionalities such as scanning for known malicious apps, encrypting and decrypting sensitive data, monitoring network traffic, and authenticating users.

Algorithm:

1) Import Required Libraries

     Import hashlib, os, socket, ssl, base64, and Fernet from cryptography.fernet.
     Import getpass for password input.

2) Define Known Malicious App Hashes

     Initialize known_malicious_apps with hashes of known malicious apps.

3) Function Definitions:

     Initialize an empty list malicious_apps.
     Iterate through each app in app_list.
     Compute the MD5 hash of app.
     If the computed hash exists in known_malicious_apps, add app to malicious_apps.
     Return malicious_apps.

4) generate_key()

     Use Fernet.generate_key() to generate a symmetric encryption key.
     Return key.

5) encrypt_data(data, key)

     Input: data (plaintext data to encrypt), key (encryption key)
     Output: encrypted_data (encrypted data)
     Initialize a Fernet object with key.
     Encrypt data using Fernet.encrypt() method.
     Return encrypted_data.

6) decrypt_data(encrypted_data, key)

     Input: encrypted_data (data to decrypt), key (encryption key)
     Output: decrypted_data (decrypted plaintext)
     Initialize a Fernet object with key.
     Decrypt encrypted_data using Fernet.decrypt() method.
     Decode decrypted_data and return.

7) monitor_network_traffic()

     Print "Monitoring network traffic..." (simulated functionality).

8) secure_connection(host, port)

Input: host (hostname or IP address), port (port number)
Create a default SSL context with ssl.create_default_context().
Create a TCP connection to host and port using socket.create_connection().
Wrap the socket with SSL/TLS using context.wrap_socket() and
server_hostname=host.
Print the negotiated SSL/TLS version (ssock.version()).
9) authenticate_user(username, password, stored_hash)

Input: username (entered username), password (entered password),
stored_hash (hashed password from storage)
Output: True (authentication successful) or False (authentication failed)
Compute the SHA-256 hash of password.
Compare the computed hash with stored_hash.
Return True if they match, otherwise False.
Main Program Execution (if__name__== "__main__":)

Part 1 - Scan for Malicious Apps:

Demonstrates scanning installed apps (["app1", "malicious_app"] in this case) against a predefined list of known malicious app hashes (known_malicious_apps). It identifies 'malicious_app' as malicious.

Part 2 - Secure Data Storage:

Generates a key for encryption using generate_key().
Encrypts the sensitive data "This is a sensitive information" and prints both encrypted and decrypted versions.

Part 3 - Monitor Network Traffic:

Simulates monitoring network traffic.

Part 4 - Establish Secure Connection:

Establishes a secure TLS connection to www.example.com on port 443.

Part 5 - User Authentication:

Simulates user authentication where the user enters a username and password (user123 and secure_password respectively in this example). It verifies the entered password against a stored hash (stored_hash).

Program:

Output :

Result:

EX.NO:10	
	Intrusion Detection/Prevention System with Snort

Aim:

To Configure Snort to monitor network traffic, detect intrusion attempts, log them, and report when an intrusion attempt is detected.

Algorithm /Program:

1. Install Snort

Update your system:

bash

sudo apt-get update

Install necessary dependencies:

bash

sudo apt-get install -y build-essential libpcap-dev libpcre3-dev libdumbnet-dev bison flex

Download and install Snort:

bash

wget https://www.snort.org/downloads/snort/snort-2.9.17.tar.gz

tar -xzvf snort-2.9.17.tar.gz

cd snort-2.9.17

./configure

make

sudo make install

Verify Snort installation:

bash

snort -V

2. Configure Snort

Create necessary directories:

bash

sudo mkdir /etc/snort

sudo mkdir /etc/snort/rules

sudo mkdir /etc/snort/preproc_rules

sudo mkdir /var/log/snort

sudo mkdir /usr/local/lib/snort_dynamicrules

3. Copy configuration files:

bash

sudo cp etc/* /etc/snort

sudo cp src/dynamic-preprocessors/build/usr/local/ lib/snort_dynamicpreprocessor/* /usr/local/lib/snort_dynamicpreprocessor/

4.Download and update the rule set:

bash

wget https://www.snort.org/rules/snortrules-snapshot-29170.tar.gz -O /tmp/snortrules-

snapshot.tar.gz
tar -xzvf /tmp/snortrules-snapshot.tar.gz -C
/etc/snort/rules

5. Edit the Snort configuration file:
Open /etc/snort/snort.conf in a text editor:
bash
sudo nano /etc/snort/snort.conf

1. Set the following variables:

plaintext
ipvar HOME_NET any
ipvar EXTERNAL_NET any
var RULE_PATH /etc/snort/rules
var PREPROC_RULE_PATH /etc/snort/preproc_rules
var WHITE_LIST_PATH /etc/snort/rules
var BLACK_LIST_PATH /etc/snort/rules
output unified2: filename snort.u2, limit 128

2. Include rules:
plaintext
include $RULE_PATH/local.rules

3. Create a Local Rule File
Create local.rules file:
bash
sudo nano /etc/snort/rules/local.rules

4. Add a sample rule:

plaintext
alert icmp any any -> $HOME_NET any (msg:"ICMP Packet Detected";
sid:1000001; rev:001;)

5. Run Snort
Test Snort configuration:
bash
sudo snort -T -c /etc/snort/snort.conf

6. Run Snort in intrusion detection mode:
bash
sudo snort -A console -q -c /etc/snort/snort.conf -i eth0
Replace eth0 with the appropriate network interface.

7. Generate and Test Intrusion Detection
Generate network traffic:
Use ping or other network utilities to generate traffic.
For example:
bash
ping -c 4 8.8.8.8
Check Snort alerts:
Snort should display alerts on the console for detected ICMP packets as specified in local.rules.

8. Log and Report Intrusion Attempts
Configure logging in snort.conf:

Ensure the following line is present for unified2 logging:
plaintext
Copy code
output unified2: filename snort.u2, limit 128
Analyze logs:
Install Barnyard2 to process Snort logs:
bash

```
sudo apt-get install -y barnyard2
```

Configure Barnyard2 to read Snort's unified2 logs and output to a database or other formats.
Start Barnyard2:
bash

```
sudo barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w
/var/log/snort/barnyard2.waldo
```

9. Automate Snort and Barnyard2 Startup
Create a systemd service for Snort:
bash

```
sudo nano /etc/systemd/system/snort.service
```

Add the following content:
plaintext

```
[Unit]
Description=Snort NIDS Daemon
After=network.target
[Service]
ExecStart=/usr/local/bin/snort -c /etc/snort/snort.conf -i eth0
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
[Install]
WantedBy=multi-user.target
```

Enable and start the service:
bash

```
sudo systemctl enable snort
sudo systemctl start snort
```

Create a systemd service for Barnyard2:
bash

```
sudo nano /etc/systemd/system/barnyard2.service
```

Add the following content:
plaintext

```
[Unit]
Description=Barnyard2 Daemon
After=network.target
[Service]
ExecStart=/usr/bin/barnyard2 -c /etc/snort/barnyard2.conf -d /var/log/snort -f snort.u2 -w
/var/log/snort/barnyard2.waldo
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
```

Restart=on-failure
[Install]
WantedBy=multi-user.target
Enable and start the service:
bash
sudo systemctl enable barnyard2
sudo systemctl start barnyard2

Result:

EX.NO:11	**DEFEATING MALWARE - BUILDING TROJANS**

Aim:

To build a Trojan and know the harmness of the Trojan malwares in a computer system.

Algorithm:

1. Create a simple Trojan by using Windows Batch File (*.bat*)
2. Type these below code in notepad and save it as **Trojan.bat**
3. Double click on *Trojan.bat* file.
4. When the Trojan code executes, it will open MS-Paint, Notepad, Command Prompt,Explorer, etc., infinitely.
5. Restart the computer to stop the execution of this Trojan.

TROJAN:

- In computing, a Trojan horse,or Trojan, is any malware which misleads users of itstrue intent.

- Trojans are generally spread by some form of social engineering, for example where a user is duped into executing an email attachment disguised to appear not suspicious, (e.g., a routine form to be filled in), or by clicking on some fake advertisement on social media or anywhere else.

- Although their payload can be anything, many modern forms act as a backdoor, contacting a controller which can then have unauthorized access to the affected computer.

- Trojans may allow an attacker to access users' personal information such as bankinginformation, passwords, or personal identity.

- *Example: Ransomware* attacks are often carried out using a *trojan*.

Setting Up a Safe Environment

1. Create a Virtual Machine

1. **Download and Install VirtualBox or VMware**: These are popular VM managers.
2. **Create a New Virtual Machine**: Install a Windows operating system on it.

2. Prepare the Virtual Environment

1. **Isolate the VM**: Ensure the VM network settings are set to "Host-only" or disconnected to prevent any potential spread.
2. **Take a Snapshot**: Before starting, take a snapshot of your VM. This allows you to revert to a clean state if needed.

Creating and Running the Batch Script

1. **Open Notepad in the VM**:
   o Press Win + R, type notepad, and press Enter.
2. **Create the Batch Script**:
   o Copy and paste the following code into Notepad:

   ```
   batch
   @echo off
   :x
   start mspaint
   start notepad
   start cmd
   start explorer
   start control
   start calc
   goto x
   ```

3. **Save the Script**:
   o Save the file with a .bat extension, for example, Trojan.bat.
4. **Execute the Script**:
   o Double-click the Trojan.bat file to execute it.

Observing the Behavior

- **Open Task Manager**:
  o Press Ctrl + Shift + Esc to open Task Manager and observe the running processes.
- **Monitor System Performance**:
  o Check CPU and memory usage to see the impact of the script.

Stopping the Script

1. **Open Task Manager**:
   - Press Ctrl + Shift + Esc to open Task Manager.
2. **End the Batch Script Process**:
   - Find the running cmd.exe processes related to the batch script and end them.
3. **Close the Applications**:
   - Manually close any opened applications (MS Paint, Notepad, CMD, Explorer, Control Panel, Calculator).

Clean Up and Restore

1. **Delete the Batch Script**:
   - Delete the Trojan.bat file to prevent accidental re-execution.
2. **Revert the VM**:
   - Revert to the snapshot taken before running the script to ensure the VM is clean.

Program:

Output:

Result:

EX.NO:12	**DEFEATING MALWARE - ROOTKIT HUNTER**

Aim:

To install a rootkit hunter and find the malwares in a computer.

Algorithm:

ROOTKIT HUNTER:
- rkhunter (Rootkit Hunter) is a Unix-based tool that scans for rootkits, backdoors and possible local exploits.
- It does this by comparing SHA-1 hashes of important files with known good ones in online databases, searching for default directories (of rootkits), wrong permissions, hidden files, suspicious strings in kernel modules, and special tests for Linux and FreeBSD.
- rkhunter is notable due to its inclusion in popular operating systems (Fedora, Debian,etc.)
- The tool has been written in Bourne shell, to allow for portability. It can run on almost all UNIX-derived systems.

GMER ROOTKIT TOOL:
- GMER is a software tool written by a Polish researcher Przemysław Gmerek, for detecting and removing rootkits.
- It runs on Microsoft Windows and has support for Windows NT, 2000, XP, Vista, 7, 8

and 10. With version 2.0.18327 full support for Windows x64 is added

Step 1

Visit GMER's website (see Resources) and download the GMER executable.

Click the "Download EXE" button to download the program with a random file name, assome rootkits will close "gmer.exe" before you can open it.

Step 2
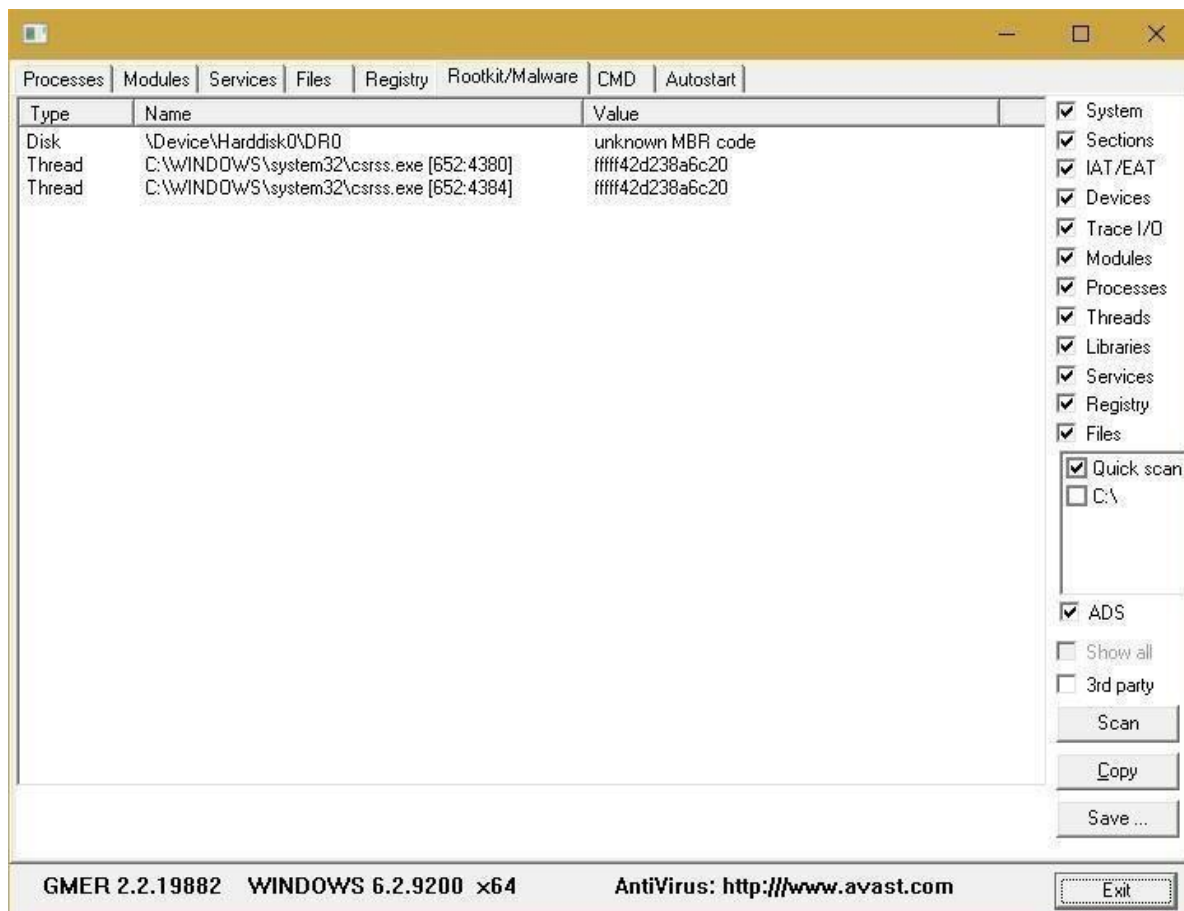
Double-click the icon for the program.
Click the "Scan" button in the lower-right corner ofthe dialog box. Allow the program toscan your entire hard drive.

Step 3
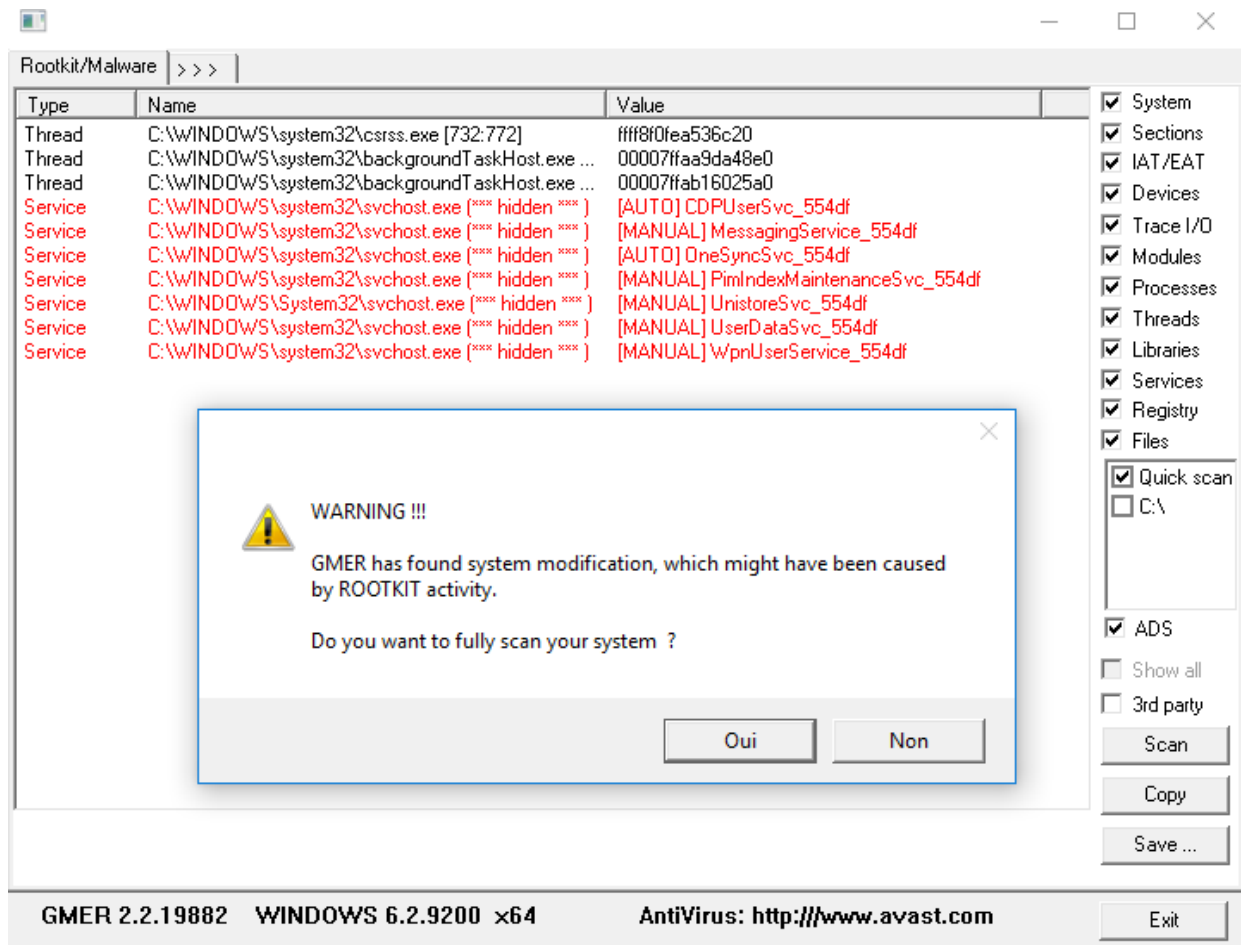
When the program completes its scan, select any program or file listed in red. Right-click it and select "Delete."

If the red item is a service, it may be protected. Right-click the service and select "Disable." Reboot your computer and run the scan again, this time selecting "Delete" when that service is detected.

When your computer is free of Rootkits, close the program and restart your PC.

RESULT:

EX.NO:13	

Implement Database Security

Aim:

        To implementing database security mechanisms, specifically access control and authentication, on a Windows OS

Algorithm / Program:

1. **Install Microsoft SQL Server**: Ensure that Microsoft SQL Server is installed on your Windows machine. You can download it from the Microsoft SQL Server website.
2. **Install SQL Server Management Studio (SSMS)**: This is a graphical tool for managing SQL Server instances. You can download it from the SSMS download page.

Step-by-Step Guide :

 This environment involves several steps. Here, we'll use Microsoft SQL Server as an example database management system to demonstrate how to set up and secure a database on Windows.

1. Understanding Database Security Threats

Common security threats to databases include:

- Unauthorized access
- SQL injection attacks
- Privilege escalation
- Data breaches due to weak authentication
- Insider threats

2. Implementing Access Control

Access control involves defining who can access the database and what actions they can perform. This is typically done using roles and permissions.

1. **Create Database and User Roles**:
   - Open SQL Server Management Studio (SSMS).
   - Connect to your SQL Server instance.
   - Create a new database:

     sql

     CREATE DATABASE SecureDB;
     GO

49

2. **Create Users and Roles**:
   - o Create a user with administrative privileges:

     sql
     USE SecureDB;
     CREATE LOGIN admin_user WITH PASSWORD = 'StrongPassword';
     CREATE USER admin_user FOR LOGIN admin_user;
     EXEC sp_addrolemember 'db_owner', 'admin_user';
     GO

   - o Create a read-only user:

     sql
     USE SecureDB;
     CREATE LOGIN read_only_user WITH PASSWORD = 'ReadOnlyPassword';
     CREATE USER read_only_user FOR LOGIN read_only_user;
     EXEC sp_addrolemember 'db_datareader', 'read_only_user';
     GO

3. **Verify Access Control**:
   - o Log in as read_only_user and try to perform write operations to confirm that they are restricted.

3. Implementing Authentication

Authentication ensures that only authorized users can access the database.

1. **Enforce Strong Password Policies**:
   - o Open SQL Server Management Studio (SSMS).
   - o Connect to your SQL Server instance.
   - o Set up strong password policies:

     sql
     Copy code
     ALTER LOGIN admin_user WITH CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON;
     ALTER LOGIN read_only_user WITH CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON;

2. **Enable Windows Authentication**:
   - o SQL Server supports both SQL Server authentication and Windows authentication. Windows authentication is generally more secure as it integrates with Windows Active Directory.
   - o In SSMS, navigate to **Security > Logins**.
   - o Right-click and select **New Login**.
   - o Choose **Windows authentication** and specify the Windows user or group.

3. **Configure SQL Server for Mixed Mode Authentication (if needed)**:
   - o Open SQL Server Configuration Manager.
   - o Navigate to **SQL Server Services**.
   - o Right-click on the SQL Server instance and select **Properties**.
   - o In the **Security** tab, choose **SQL Server and Windows Authentication mode**.
   - o Restart the SQL Server service for the changes to take effect.

4. Testing and Verification

1. **Test Access Control**:
   - o Ensure users have the correct permissions and cannot access or modify data beyond their privileges.
2. **Test Authentication Mechanisms**:
   - o Attempt to log in with weak passwords and verify that access is denied.
   - o Ensure that both SQL Server and Windows Authentication modes work as expected.
3. **Audit and Logging**:
   - o Enable SQL Server auditing to monitor access and detect any unauthorized attempts.

     ```sql
     CREATE SERVER AUDIT AuditTest
     TO FILE (FILEPATH = 'C:\Audit\');
     ALTER SERVER AUDIT AuditTest WITH (STATE = ON);
     GO
     CREATE DATABASE AUDIT SPECIFICATION AuditSpec
     FOR SERVER AUDIT AuditTest
     ADD (SELECT ON DATABASE::SecureDB BY read_only_user),
     ADD (SELECT, INSERT, UPDATE, DELETE ON DATABASE::SecureDB BY admin_user);
     ALTER DATABASE AUDIT SPECIFICATION AuditSpec WITH (STATE = ON);
     GO
     ```

Output :

Result:

EX.NO:14	
	Implement Encryption and Integrity Control-Database Security

Aim:

To implementing encryption and integrity controls for databases is crucial to protect sensitive data and ensure that it remains unaltered.

Algorithm /Program:

1. **Microsoft SQL Server**: Ensure that SQL Server is installed on your Windows system.
2. **SQL Server Management Studio (SSMS)**: Ensure SSMS is installed for managing the SQL Server instance.

Steps to Implement Encryption and Integrity Controls

1. Transparent Data Encryption (TDE)

Transparent Data Encryption (TDE) helps protect data at rest by encrypting the database files. This ensures that the database files are not readable if accessed directly from the disk.

1. **Create a Master Key**

   The master key is required to encrypt the database encryption key.

   ```sql
   USE master;
   GO
   CREATE MASTER KEY ENCRYPTION BY PASSWORD =
   'StrongPasswordForMasterKey';
   GO
   ```

2. **Create a Certificate**

   The certificate is used to protect the database encryption key.

   ```sql
   USE master;
   GO
   CREATE CERTIFICATE TDE_Cert WITH SUBJECT = 'TDE Certificate';
   GO
   ```

3. **Create a Database Encryption Key**

The database encryption key is used to encrypt the database.

```sql
USE SecureDB;
GO
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDE_Cert;
GO
```

4. Enable TDE on the Database

```sql
ALTER DATABASE SecureDB
SET ENCRYPTION ON;
GO
```

5. Verify Encryption

```sql
USE SecureDB;
GO
SELECT name, is_encrypted
FROM sys.databases
WHERE name = 'SecureDB';
GO
```

Expected Output:

- o is_encrypted should be 1 for the SecureDB database.

2. Column-Level Encryption

Column-level encryption provides fine-grained control over the encryption of specific data within a table.

1. Create a Symmetric Key

```sql
USE SecureDB;
GO
CREATE SYMMETRIC KEY SymmetricKey
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE TDE_Cert;
GO
```

2. Encrypt Data in a Table

- o Create a table and insert some data:

```sql
CREATE TABLE SensitiveData (
    ID INT PRIMARY KEY,
    SensitiveInfo VARBINARY(MAX)
);
GO

OPEN SYMMETRIC KEY SymmetricKey
DECRYPTION BY CERTIFICATE TDE_Cert;

INSERT INTO SensitiveData (ID, SensitiveInfo)
VALUES (1, ENCRYPTBYKEY(KEY_GUID('SymmetricKey'), 'Sensitive
Information'));
GO
CLOSE SYMMETRIC KEY SymmetricKey;
```

3. Decrypt Data for Viewing

```sql
OPEN SYMMETRIC KEY SymmetricKey
DECRYPTION BY CERTIFICATE TDE_Cert;

SELECT ID, CONVERT(VARCHAR(MAX), DECRYPTBYKEY(SensitiveInfo)) AS
SensitiveInfo
FROM SensitiveData;
GO

CLOSE SYMMETRIC KEY SymmetricKey;
```

Expected Output:

- o The SensitiveInfo column should display the decrypted data.

3. Data Integrity Controls

Implementing data integrity controls ensures that the data is not tampered with and maintains its accuracy and consistency.

1. Using Hashes for Data Integrity

- o Create a table to store hashed data:

```sql
CREATE TABLE DataIntegrity (
```

```sql
    ID INT PRIMARY KEY,
    OriginalData NVARCHAR(255),
    DataHash VARBINARY(64)
);
GO
```

o   Insert data with a hash:

```sql
sql
INSERT INTO DataIntegrity (ID, OriginalData, DataHash)
VALUES (1, 'Important Data', HASHBYTES('SHA2_256', 'Important Data'));
GO
```

o   Verify data integrity:

```sql
sql
DECLARE @OriginalData NVARCHAR(255);
DECLARE @Hash VARBINARY(64);

SELECT @OriginalData = OriginalData, @Hash = DataHash
FROM DataIntegrity
WHERE ID = 1;

IF @Hash = HASHBYTES('SHA2_256', @OriginalData)
    PRINT 'Data integrity verified.';
ELSE
    PRINT 'Data has been tampered with.';
GO
```

2. Expected Output:
o   Data integrity verified. Should be printed if the data has not been altered.

Output:

Result: