1. Introduction To Legal Issuesdisk Imaging And Cloning: Use Vmware And Modify Device Configuration In A Vmware System - Image A Drive To A File - Extract Individual Partitions From An Image File - Mount The Image As A Loopback Device And Read Only For Analysis - Properly Sanitize A Disk For Cloning - Clone A Drive Versus Imaging The Drive - Verify Disk And File Integrity With Hashing.

AIM:

to create, manage, and verify disk images and clones for backup, analysis, and secure data handling.

ALGORITHM

 Image a Drive:

- Identify source drive (/dev/sdX).
- Use dd to create an image: dd if=/dev/sdX of=/path/to/image.img bs=4M.

 Extract Partition:

- Determine partition offset and size with fdisk.
- Extract partition: dd if=/path/to/image.img of=/path/to/partition.img bs=1M skip=<offset> count=<size>.

 Mount Image:

- Create mount point: mkdir /mnt.
- Mount image: mount -o loop,ro /path/to/image.img /mnt.

 Sanitize Disk:

- Identify target drive (/dev/sdY).
- Erase drive: shred -v -n 3 /dev/sdY.

 Clone Drive:

- Clone source drive to target: dd if=/dev/sdX of=/dev/sdY bs=4M.

 Verify Integrity:

- Generate hash: sha256sum /path/to/image.img > /path/to/image.img.sha256.

- Verify hash: sha256sum -c /path/to/image.img.sha256.

PROGRAM

```
#!/bin/bash

# Variables
SOURCE_DRIVE="/dev/sdX"
TARGET_DRIVE="/dev/sdY"
IMAGE_FILE="/path/to/image.img"
PARTITION_IMAGE="/path/to/partition.img"
MOUNT_POINT="/mnt"
HASH_FILE_IMG="${IMAGE_FILE}.sha256"

# 1. Image a Drive to a File
echo "Creating disk image of $SOURCE_DRIVE..."
sudo dd if=$SOURCE_DRIVE of=$IMAGE_FILE bs=4M status=progress
echo "Disk image created: $IMAGE_FILE"

# 2. Extract Individual Partitions from an Image File
# Example: Extract partition 1 with offset and size
PARTITION_OFFSET=2048  # Replace with actual offset
PARTITION_SIZE=102400  # Replace with actual size
echo "Extracting partition from image..."
sudo dd if=$IMAGE_FILE of=$PARTITION_IMAGE bs=1M
skip=$PARTITION_OFFSET count=$PARTITION_SIZE
```

```
echo "Partition extracted: $PARTITION_IMAGE"

# 3. Mount the Image as a Loopback Device
echo "Mounting image as loopback device..."
sudo mkdir -p $MOUNT_POINT
sudo mount -o loop,ro $IMAGE_FILE $MOUNT_POINT
echo "Image mounted at $MOUNT_POINT"

# 4. Properly Sanitize a Disk for Cloning
echo "Sanitizing target disk $TARGET_DRIVE..."
sudo shred -v -n 3 $TARGET_DRIVE
echo "Disk sanitized."

# 5. Clone a Drive
echo "Cloning $SOURCE_DRIVE to $TARGET_DRIVE..."
sudo dd if=$SOURCE_DRIVE of=$TARGET_DRIVE bs=4M status=progress
echo "Drive cloned."

# 6. Verify Disk and File Integrity with Hashing
echo "Generating hash for image..."
sha256sum $IMAGE_FILE > $HASH_FILE_IMG
echo "Hash generated: $HASH_FILE_IMG"

echo "Verification:"
```

```
sha256sum -c $HASH_FILE_IMG

# Clean up
echo "Unmounting image and cleaning up..."
sudo umount $MOUNT_POINT
sudo rmdir $MOUNT_POINT
echo "Done."
```

OUTPUT

```
# 1. Image a Drive
$ sudo dd if=/dev/sdX of=/path/to/image.img bs=4M status=progress
123456789 bytes (123 MB, 117 MiB) copied, 12.3456 s, 10.0 MB/s
Disk image created: /path/to/image.img

# 2. Extract Partition
$ sudo dd if=/path/to/image.img of=/path/to/partition.img bs=1M skip=2048 count=102400
123456789 bytes (123 MB, 117 MiB) copied, 11.3456 s, 10.8 MB/s
Partition extracted: /path/to/partition.img

# 3. Mount Image
$ sudo mkdir /mnt
$ sudo mount -o loop,ro /path/to/image.img /mnt
$ df -h /mnt
Filesystem          Size  Used Avail Use% Mounted on
```

```
/path/to/image.img         120G   50G   70G  42% /mnt
Image mounted at /mnt

# 4. Sanitize Disk
$ sudo shred -v -n 3 /dev/sdY
shred: /dev/sdY: pass 1/3 (random)...done
shred: /dev/sdY: pass 2/3 (random)...done
shred: /dev/sdY: pass 3/3 (random)...done
Disk sanitized.

# 5. Clone Drive
$ sudo dd if=/dev/sdX of=/dev/sdY bs=4M status=progress
123456789 bytes (123 MB, 117 MiB) copied, 15.6789 s, 8.0 MB/s
Drive cloned.

# 6. Verify Integrity
$ sha256sum /path/to/image.img > /path/to/image.img.sha256
$ sha256sum -c /path/to/image.img.sha256
/path/to/image.img: OK
Hash generated: /path/to/image.img.sha256
```

RESULT

2. IMPLEMENT SHA-1 ALGORITHM

AIM

To implement SHA-1 algorithm

ALGORITHM

□ **Initialization**:

- Set five 32-bit hash values (H0 to H4) with predefined constants.

□ **Preprocessing**:

- **Pad** the message to be a multiple of 512 bits (include message length).
- **Divide** into 512-bit blocks.

□ **Process Each Block**:

- **Break** into 16 words.
- **Extend** to 80 words.
- **Initialize** temporary variables with hash values.
- **Run 80 Rounds**: Update variables using bitwise operations and constants.
- **Update** hash values after processing each block.

□ **Output**:

- **Combine** hash values into a final 160-bit hash.
- **Convert** the hash to a hexadecimal string.

PROGRAM

```
import hashlib


def sha1_hash(data):
    """

    Compute the SHA-1 hash of the given data.
    :param data: Input data to be hashed (bytes).
    :return: SHA-1 hash of the data (hexadecimal string).
    """
    sha1 = hashlib.sha1()
    sha1.update(data)
    return sha1.hexdigest()

# Example usage
if __name__ == "__main__":
    # Sample input data
    data = b"Hello, World!"

    # Compute SHA-1 hash
    hash_value = sha1_hash(data)

    print(f"SHA-1 hash of the data: {hash_value}")
```

OUTPUT

```
SHA-1 hash of the data: 65a8e27d8879283831b664bd8b7f0ad4c5e8d9f7
```

RESULT

3. Implement MD5 algorithm for practical appications.

AIM

To Implement MD5 algorithm for practical appications.

ALGORITHM

□ **Initialization**: Set up four 32-bit variables (A, B, C, D) with predefined constants.

□ **Preprocessing**:

- **Pad** the input message to make its length 64 bits short of a multiple of 512.
- **Divide** the padded message into 512-bit blocks.

□ **Process Each Block**:

- **Break** into 32-bit words.
- **Initialize** temporary variables with current hash values.
- **Perform 64 Rounds**: Use bitwise operations and functions to update hash values.

□ **Output**:

- **Combine** the final values of A, B, C, and D.
- **Convert** to a hexadecimal string.

PROGRAM

```
import hashlib


def md5_hash(data):
    """
    Compute the MD5 hash of the given data.
    :param data: Input data to be hashed (bytes).
    :return: MD5 hash of the data (hexadecimal string).
    """
    md5 = hashlib.md5()
    md5.update(data)
    return md5.hexdigest()

# Example usage
if __name__ == "__main__":
    # Sample input data
    data = b"Hello, World!"

    # Compute MD5 hash
    hash_value = md5_hash(data)

    print(f"MD5 hash of the data: {hash_value}")
```

OUTPUT

```
MD5 hash of the data: 65a8e27d8879283831b664bd8b7f0ad4
```

RESULT

4. Implementing Digital Signal Standard (DSS).

AIM

To implement Digital Signal Standard

ALGORITHM

- **Generate Keys**:
    - Generate a DSA private key and derive the public key from it.
- **Sign Message**:
    - Hash the message using SHA-256.
    - Sign the hash with the private key to create the digital signature.
- **Verify Signature**:
    - Hash the original message using SHA-256.
    - Verify the signature using the public key and the hashed message.

PROGRAM

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

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.serialization import Encoding,
PrivateFormat, PublicFormat, NoEncryption

from cryptography.hazmat.primitives.asymmetric import utils

import base64


def generate_keys():
    """
    Generate a DSA private key and corresponding public key.
    """
    private_key = dsa.generate_private_key(key_size=2048)
    public_key = private_key.public_key()

    return private_key, public_key

def sign_message(private_key, message):
    """
    Sign a message with the DSA private key.

    :param private_key: DSA private key.
    :param message: Message to be signed (bytes).
    :return: Signature (base64 encoded).
    """
    signature = private_key.sign(
        message,
        hashes.SHA256()
    )
    return base64.b64encode(signature).decode('utf-8')

def verify_signature(public_key, message, signature):
    """
    Verify a signature with the DSA public key.

    :param public_key: DSA public key.
    :param message: Original message (bytes).
    :param signature: Signature to be verified (base64 encoded).
    :return: True if verification is successful, otherwise False.
    """
    try:
        public_key.verify(
            base64.b64decode(signature),
            message,
            hashes.SHA256()
        )
        return True
    except Exception as e:
        print(f"Verification failed: {e}")
        return False

# Example usage
if __name__ == "__main__":
    # Generate keys
    private_key, public_key = generate_keys()

    # Sample message
    message = b"Hello, Digital Signature!"

    # Sign message
    signature = sign_message(private_key, message)
    print(f"Signature: {signature}")

    # Verify signature
    is_valid = verify_signature(public_key, message, signature)
    print(f"Signature valid: {is_valid}")
```

OUTPUT

```
Signature: <base64_encoded_signature>
Signature valid: True
```

RESULT

5. Crack passwords using John the Ripper.

AIM

To Crack passwords using John the Ripper.

ALGORITHM

 Install John the Ripper on your system.

 Prepare a file with hashed passwords.

 Run John the Ripper with the appropriate commands to crack passwords.

 View the results to see which passwords have been cracked.

PROGRAM

Install John the Ripper:

- **Linux**:

```
sudo apt-get install john
```

Windows: Download from John the Ripper's official site, extract, and navigate to the extracted directory.

 Prepare Your Password File: Create a file (passwords.txt) with hashed passwords. Example content:

```
admin:$1$abc123$T3yKl9RXx/ZLQr1xBcBMD/
user:$1$def456$KjxHn8xzY1tL5MghrWrfE.
```

Basic Cracking:

```
john passwords.txt
```

 This will use the default wordlist and settings.

 Using a Custom Wordlist:

```
john --wordlist=/path/to/wordlist.txt passwords.txt
```

Example using the RockYou wordlist:

```
john --wordlist=/usr/share/wordlists/rockyou.txt passwords.txt
```

Using Incremental Mode:

```
john --incremental passwords.txt
```

View Cracked Passwords:

```
john --show passwords.txt
```

OUTPUT

```
admin:password123

user:qwerty123
```

RESULT

6. Demonstrate penetration testing using any tool (Metasploit or wireshark, etc).

AIM

To Demonstrate penetration testing using any tool (Metasploit or wireshark, etc).

ALGORITHM

□ **Install Metasploit**:

- Set up Metasploit on your system.

□ **Start Metasploit Console**:

- Launch the msfconsole command.

□ **Scan the Target**:

- Use tools like nmap to identify open ports and services.

□ **Search for Exploits**:

- Use Metasploit's search command to find relevant exploits for the identified services.

□ **Select an Exploit**:

- Choose and configure the exploit module with the use command.

□ **Configure Exploit**:

- Set the necessary options, such as target (RHOST) and local host (LHOST).

□ **Run the Exploit**:

- Execute the exploit to attempt to gain access.

□ **Post-Exploitation**:

- Manage and interact with any sessions opened by the exploit.

□ **End the Session and Exit**:

- Terminate sessions and exit Metasploit.

PROGRAM

Install Metasploit:

- **Linux**:

```
curl https://raw.githubusercontent.com/rapid7/metasploit-framework/master/msfupdate | bash
```

Windows: Download and install from Metasploit's official site.

□ **Start Metasploit Console**:

```
Msfconsole
```

Scan the Target (using nmap or another scanner):

```
nmap -sV <target-ip>
```

Search for Exploits:

Find an appropriate exploit related to the discovered services:

```
search <service-name>
```

Select an Exploit:

Example for vsftpd 2.3.4 backdoor:

```
use exploit/unix/ftp/vsftpd_234_backdoor
```

Configure the Exploit:

Set required options such as remote host (RHOST) and local host (LHOST):

```
set RHOST <target-ip>
set LHOST <your-ip>
```

Run the Exploit:

```
Exploit
```

Post-Exploitation:

List active sessions:

```
sessions -l
```

Interact with a session:

```
sessions -i <session-id>
```

End the Session and Exit Metasploit:

To end an active session:

```
Exit
```

To exit Metasploit:

```
Exit
```

Example Workflow

1. **Scan Target IP**:

```
nmap -sV 192.168.1.10
```

Search for Exploit:

```
search vsftpd
```

Select and Configure Exploit:

```
use exploit/unix/ftp/vsftpd_234_backdoor

set RHOST 192.168.1.10

set LHOST 192.168.1.100
```

Run the Exploit:

```
exploit
```

Interact with the Session:

```
sessions -i 1
```

RESULT

7. Demonstrate intrusion detection system (IDS) using Snort.

AIM

To Demonstrate intrusion detection system (IDS) using Snort.

ALGORITHM

□ **Install Snort**:

- Install Snort on your system using package managers or by downloading from the official website.

□ **Configure Snort**:

- Edit the Snort configuration file (snort.conf) to set up network interfaces and rule paths.

□ **Update Rules**:

- Download and install Snort rules to detect various types of network intrusions.

□ **Start Snort**:

- Run Snort in IDS mode to monitor network traffic and generate alerts.

□ **Generate Test Traffic**:

- Use tools to generate network traffic that will trigger Snort alerts.

□ **View Alerts**:

- Check Snort’s output or log files for alerts and analyze detected intrusions.

PROGRAM

Start Snort

Command:

```
sudo snort -A console -c /etc/snort/snort.conf -i eth0
```

Sample Output:

```
Running in IDS mode

...

[1:1000001:1] ET POLICY Outbound SSLv2 Connection [**] [Classification: Policy Violation] [Priority: 2] {TCP} 192.168.1.100:443 -> 192.168.1.200:12345
```

Check Alerts

Command:

```
cat /var/log/snort/alert
```

Sample Output:

```
[**] [1:1000001:1] ET POLICY Outbound SSLv2 Connection [**] [Classification: Policy Violation] [Priority: 2] {TCP} 192.168.1.100:443 -> 192.168.1.200:12345

[**] [1:1000002:1] ET SCAN Potential SSH Scan [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.1.10:22 -> 192.168.1.200:33333
```

RESULT

8. Demonstrate OS fingerprinting using Nmap

AIM

To Demonstrate OS fingerprinting using Nmap

ALGORITHM

Step 1: Install Nmap (if not already installed)

- Check if Nmap is installed by running nmap in the terminal.
- If not installed, use the appropriate installation command:
  - On Linux:

```
sudo apt-get install nmap
```

  - On Windows: Download and install from the official Nmap website.

Step 2: Identify the target IP address

- Determine the IP address of the machine or network device you want to fingerprint.
- Example: 192.168.1.1 for a local router.

Step 3: Run Nmap with OS Detection Flag

- Open a terminal or command prompt and run the following Nmap command with the -O option (for OS detection):

```
sudo nmap -O <target-ip>
```

- Example:

```
sudo nmap -O 192.168.1.1
```

Step 4: Wait for the Scan to Complete

- Nmap will scan the target and gather information on open ports, services, and the operating system.
- This may take some time depending on network conditions and the target's configuration.

Step 5: Interpret the Results

- Once the scan finishes, analyze the output:
    - The guessed operating system, version, and uptime (if available) will be displayed.
    - It will show open ports and their associated services.

Step 6: Optional - Increase Verbosity

- For more detailed results, you can add the verbose flag -v to the Nmap command:

```
sudo nmap -v -O <target-ip>
```

Step 7: Verify the Results

- Cross-check the guessed operating system with known data about the target.
- If the detection seems off, rerun the scan or try additional options like version detection -sV.

Step 8: Finish

- Analyze the final results and close the terminal when done.

PROGRAM

```
sudo apt-get install nmap

sudo nmap -O <target-ip>  # Replace <target-ip> with the actual IP of your target

sudo nmap -O 192.168.1.1  # For a local router

sudo nmap -v -O 192.168.1.1
```

RESULT

9. Implementing system call filters using Seccomp BPF filter.

AIM

To Implement system call filters using Seccomp BPF filter.

ALGORITHM

- **Initialize Seccomp Filter**:
  - Start by setting up a Seccomp filter context.
  - Define the default action as SCMP_ACT_KILL to kill the process when a disallowed system call is invoked.
- **Add Allowed System Calls**:
  - Add rules to allow specific system calls such as read, write, exit, and exit_group.
  - Use the seccomp_rule_add function (for C) or filter.add_rule (for Python) to specify which system calls should be allowed.
- **Load the Filter**:
  - After adding all the necessary rules, load the filter into the kernel. This applies the restrictions for the process.
- **Run the Program**:
  - Execute the code where only allowed system calls will pass through.
  - Any disallowed system call will trigger the default action, which is to terminate the process.
- **Optional**: Test Disallowed System Calls
  - To test the filter, intentionally invoke a blocked system call (like open or fork). The process should be terminated by the Seccomp filter.

PROCEDURE

Install the python-seccomp package (if not already installed):

On Ubuntu/Debian:

```
sudo apt-get install python3-seccomp
```

Alternatively, you can install it via pip:

```
pip install python-seccomp
```

PROGRAM

```
import os
import seccomp


def apply_seccomp():
    # Initialize the seccomp filter and set the default action to kill the process
    filter = seccomp.SyscallFilter(defaction=seccomp.KILL)

    # Allow specific system calls: read, write, exit, exit_group
    filter.add_rule(seccomp.ALLOW, "read")
    filter.add_rule(seccomp.ALLOW, "write")
    filter.add_rule(seccomp.ALLOW, "exit")
    filter.add_rule(seccomp.ALLOW, "exit_group")

    # Load the filter into the kernel
    filter.load()

if __name__ == "__main__":
    print("Applying Seccomp filters...")
    apply_seccomp()
    # Allowed syscalls
    print("This is a test for allowed syscalls.")
    os.write(1, b"Write syscall is allowed.\n")

    # Example: Uncomment the following to trigger a disallowed syscall (e.g., open)
    # os.open("/tmp/testfile", os.O_RDONLY)

    print("Exiting program.")
```

OUTPUT

□ **When Allowed Syscalls are Used**:

```
Applying Seccomp filters...
This is a test for allowed syscalls.
Write syscall is allowed.
Exiting program.
```

- The os.write syscall works fine, and the message is printed.
- The program exits normally.

□ **When Disallowed Syscalls are Used**: If you uncomment os.open, you will see:

```
Applying Seccomp filters...
This is a test for allowed syscalls.
Write syscall is allowed.
Traceback (most recent call last):
  ...
  os.open("/tmp/testfile", os.O_RDONLY)
```

```
OSError: [Errno 1] Operation not permitted
```

- The process is killed due to the Seccomp filter blocking the open syscall.

RESULT

10.Implementing Security Access Control using Multi-factor authentication.

AIM

To Implement Security Access Control using Multi-factor authentication.

ALGORITHM

 Generate TOTP Secret (Done once during setup):

- Generate a TOTP secret key.
- Store the secret key securely for later verification.

 User Registration (Optional):

- **Password**: Allow the user to set or change their password.
- **TOTP Setup**: Generate and display a QR code or secret key for the user to set up in their TOTP app.

 Authentication Process:

1. **Prompt for Password**:
    - Ask the user to enter their password.
    - Compare the entered password with the stored password.
2. **Verify Password**:
    - If the password is correct, proceed to the next step.
    - If the password is incorrect, deny access and terminate the process.
3. **Prompt for TOTP Code**:
    - Ask the user to enter their TOTP code from their authentication app.
4. **Verify TOTP Code**:
    - Use the stored TOTP secret to validate the entered code.

    - Check if the TOTP code is valid and matches the current time-based OTP.

5. **Access Control**:
    - If both the password and TOTP code are verified successfully, grant access.
    - If either verification fails, deny access.

PROCEDURE

□ **Install Required Libraries**:

```
pip install pyotp
```

□ **Generate a TOTP Secret (Usually done once and saved)**:

```
import pyotp

# Generate a new TOTP secret key
totp = pyotp.TOTP(pyotp.random_base32())
secret = totp.secret
print(f"Your new TOTP secret key is: {secret}")
```

PROGRAM

```
import pyotp
import getpass

# Predefined password (in practice, securely hash and store passwords)
PASSWORD = "SecurePassword123"
```

```
# TOTP secret key (in practice, store securely)
TOTP_SECRET = "JBSWY3DPEHPK3PXP"  # Replace with the generated secret key

def verify_password():
    password = getpass.getpass(prompt="Enter your password: ")
    return password == PASSWORD

def verify_totp():
    totp = pyotp.TOTP(TOTP_SECRET)
    token = input("Enter your TOTP code: ")
    return totp.verify(token)

def main():
    print("Multi-Factor Authentication Example")

    # Verify password
    if not verify_password():
        print("Invalid password.")
        return

    # Verify TOTP
    if not verify_totp():
        print("Invalid TOTP code.")
        return
    print("Authentication successful!")


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

OUTPUT

1. **Successful Authentication**

When the user provides the correct password and a valid TOTP code, the output will be:

```
Multi-Factor Authentication Example

Enter your password: ********

Enter your TOTP code: 123456

Authentication successful!
```

2. Incorrect Password

If the user enters an incorrect password, the output will be:

```
Multi-Factor Authentication Example

Enter your password: ********

Invalid password.
```

The program will terminate after this message, and the user will not be prompted for the TOTP code.

3. Incorrect TOTP Code

If the password is correct but the TOTP code is incorrect, the output will be:

```
Multi-Factor Authentication Example

Enter your password: ********
Enter your TOTP code: 654321

Invalid TOTP code.
```

The program will terminate after this message, and the user will not receive a success message.

4. Error Handling

If an unexpected error occurs (e.g., issues with the getpass function or pyotp), the output will include an error message:

```
Multi-Factor Authentication Example

Enter your password: ********

An error occurred during password verification: [error details]
```

or

```
Multi-Factor Authentication Example

Enter your password: ********

Enter your TOTP code: 123456

An error occurred during TOTP verification: [error details]
```

RESULT