**DIGITAL FORENSICS**

**CASE STUDY**

**Team Members**

**VU22CSEN0400142- Harini Mode**

**VU22CSEN0400066- G. Abhiram**

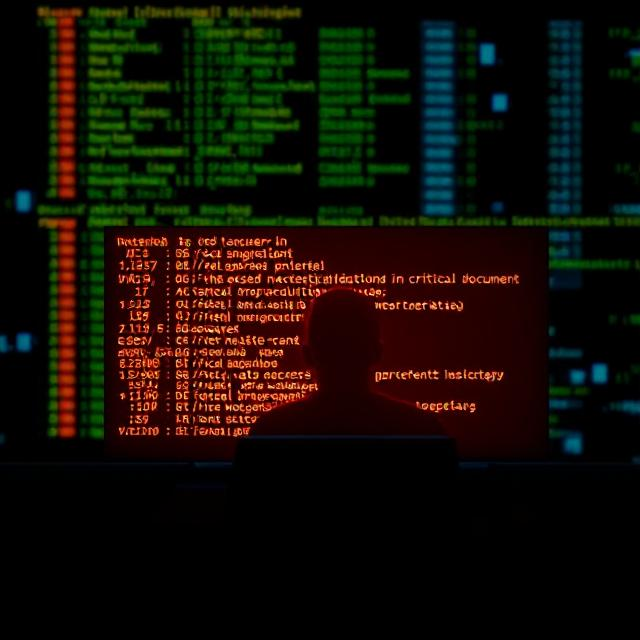
**VU22CSEN0400057-S. Siri**

**VU22CSEN0400087-Gorleen Saroya**

**Case Scenario: Digital Forensics Investigation Using the Digital Forensic Tool**

**Incident Overview:**

A financial organization detected anomalies in its security logs, raising concerns about unauthorized access and potential data tampering. Employees reported unusual modifications in critical financial documents, prompting an urgent forensic investigation.



**File Integrity Verification using hashlib in Python**

**Purpose of hashlib for File Integrity Verification:**

The hashlib library in Python provides cryptographic hash functions such as MD5, SHA-1, and SHA-256. These hashes are useful for ensuring the integrity of files by verifying that they have not been altered during storage or transmission. This technique is widely used in cybersecurity, software distribution, and digital forensics.

**Problem Statement:**

A financial organization detected anomalies in its security logs, raising concerns about unauthorized access and potential data tampering. Employees reported unusual modifications in critical financial documents, prompting an urgent forensic investigation.

**Python Code:**

import os

import hashlib

import subprocess

import argparse

from datetime import datetime

from pathlib import Path

def create\_disk\_image(source, destination):

    """Creates a forensic disk image using dd."""

    try:

        print(f"Creating disk image from {source} to {destination}...")

        subprocess.run(["sudo", "dd", f"if={source}", f"of={destination}", "bs=4M", "status=progress"], check=True)

        print("Disk image created successfully!")

    except subprocess.CalledProcessError as e:

        print(f"Error creating disk image: {e}")

#supports md5, sha1, sha256, sha512, and blake2b

def generate\_hash(file\_path, algorithm='sha256'):

    """Generates a hash of a file using the specified algorithm."""

    try:

        hash\_func = getattr(hashlib, algorithm)()

    except AttributeError:

        print(f"Unsupported hash algorithm: {algorithm}")

        return None

    with open(file\_path, "rb") as f:

        while chunk := f.read(4096):

            hash\_func.update(chunk)

    return hash\_func.hexdigest()

def compare\_files(file1, file2, algorithm='sha256'):

    """Compares the hash values of two files to check integrity."""

    hash1 = generate\_hash(file1, algorithm)

    hash2 = generate\_hash(file2, algorithm)

    if hash1 and hash2:

        print(f"{file1} ({algorithm}): {hash1}")

        print(f"{file2} ({algorithm}): {hash2}")

        if hash1 == hash2:

            print("✅ Files are identical.")

        else:

            print("⚠️ Files have been tampered with!")

def extract\_metadata(file\_path):

    """Extracts metadata from a given file."""

    file = Path(file\_path)

    if not file.exists():

        print("File does not exist!")

        return

    metadata = {

        "File Name": file.name,

        "Size (bytes)": file.stat().st\_size,

        "Created": datetime.fromtimestamp(file.stat().st\_ctime),

        "Modified": datetime.fromtimestamp(file.stat().st\_mtime),

    }

    for key, value in metadata.items():

        print(f"{key}: {value}")

def analyze\_log(file\_path):

    """Basic log analysis: Extracts error and warning messages."""

    with open(file\_path, "r") as log:

        for line in log:

            if "ERROR" in line or "WARNING" in line:

                print(line.strip())

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser(description="Digital Forensic Tool")

    parser.add\_argument("--image", nargs=2, metavar=("SOURCE", "DEST"), help="Create disk image")

    parser.add\_argument("--hash", nargs=2, metavar=("FILE", "ALGO"), help="Generate file hash (md5, sha1, sha256, sha512, blake2b)")

    parser.add\_argument("--compare", nargs=3, metavar=("FILE1", "FILE2", "ALGO"), help="Compare hash values of two files")

    parser.add\_argument("--meta", metavar="FILE", help="Extract file metadata")

    parser.add\_argument("--log", metavar="FILE", help="Analyze log file for errors/warnings")

    args = parser.parse\_args()

    if args.image:

        create\_disk\_image(args.image[0], args.image[1])

    elif args.hash:

        print(f"Hash ({args.hash[1]}):", generate\_hash(args.hash[0], args.hash[1]))

    elif args.compare:

        compare\_files(args.compare[0], args.compare[1], args.compare[2])

    elif args.meta:

        extract\_metadata(args.meta)

    elif args.log:

        analyze\_log(args.log)

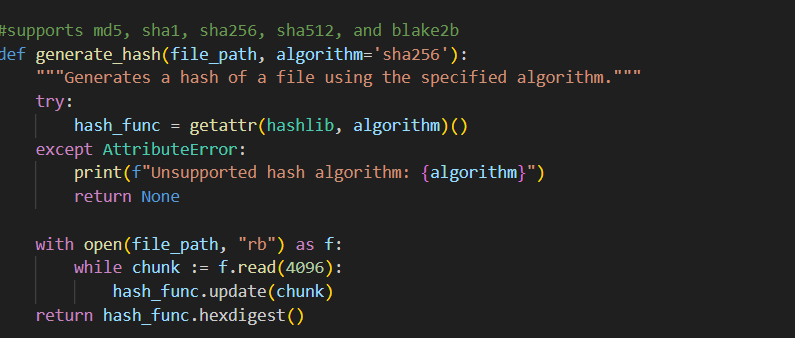
**Solution Approach**

**1.Generating a Hash**

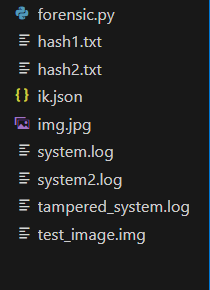
**Function: generate\_hash(file\_path, algorithm='sha256')**

**Use Case:**

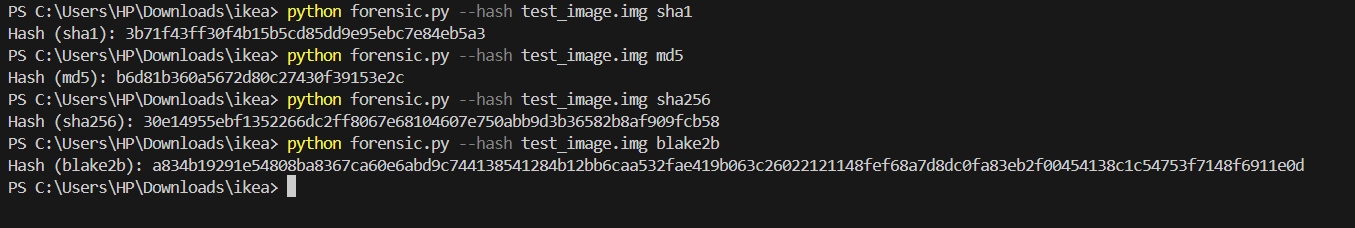
* Purpose: To generate a cryptographic hash of files for integrity verification.
* Application: After creating a disk image or extracting specific files (like financial documents), hashes can be generated for these files. This allows investigators to verify that the files have not been altered since the time of imaging. For example, if a financial document is suspected of being tampered with, generating a hash before and after any analysis can confirm whether the file's integrity has been compromised.



So here we can get the hash of the files which we check it at the time of investigation .In this it supports the md5, sha1, sha256, sha512, and blake2b.



Now in this we are checking the hashes of the file in the following way.

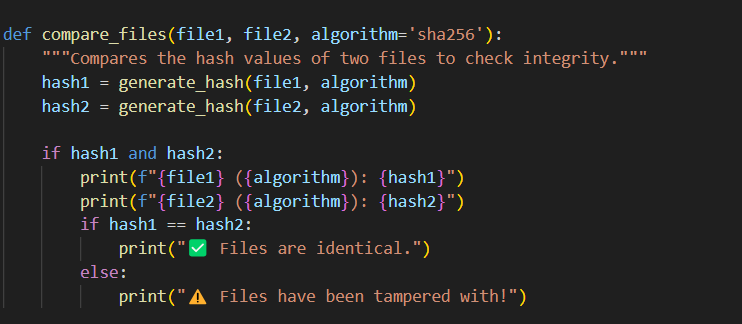


**2. Comparing Files**

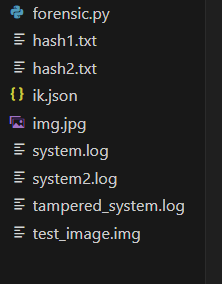
**Function**: **Compare\_files(file1, file2, algorithm='sha256')**

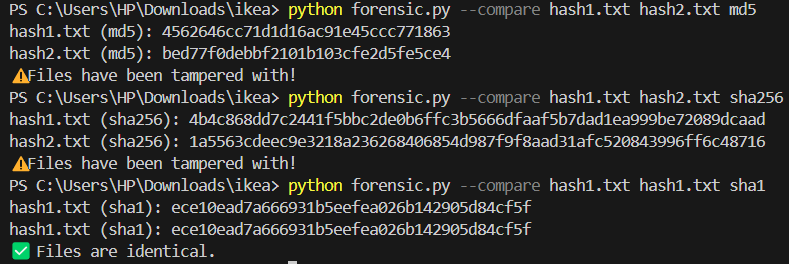
**Use Case**:

* **Purpose**: To compare the hash values of two files to check for integrity.
* **Application**: If the organization has backup copies of critical financial documents, investigators can compare the hashes of the current versions of these documents against the hashes of the backups. This comparison will help identify any unauthorized modifications or tampering. If the hashes differ, it indicates that the file has been altered.
* In this code we can compare the integrity of the code without any of the tools like MD5 Calculator.
* Here is the following piece of code:



* In the following way:





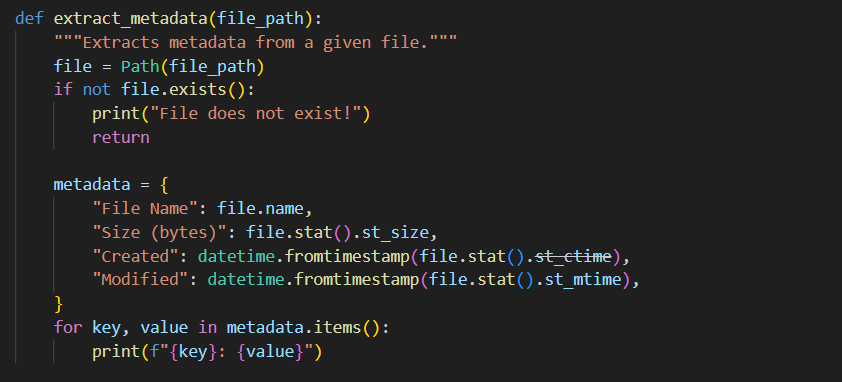
Now in this with the following command we can check the integrity of the 2 files where they are not tampered or not.

**4. Extracting Metadata**

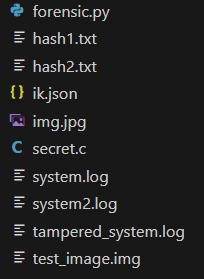
**Function**: **extract\_metadata(file\_path)**

**Use Case**:

* **Purpose**: To extract and analyze metadata from files.
* **Application**: Metadata can provide valuable information about the files in question, such as when they were created, last modified, and their size. In the context of the investigation, analyzing the metadata of critical financial documents can reveal suspicious activity, such as unexpected modification times or unauthorized user access. This information can help establish a timeline of events related to the anomalies.

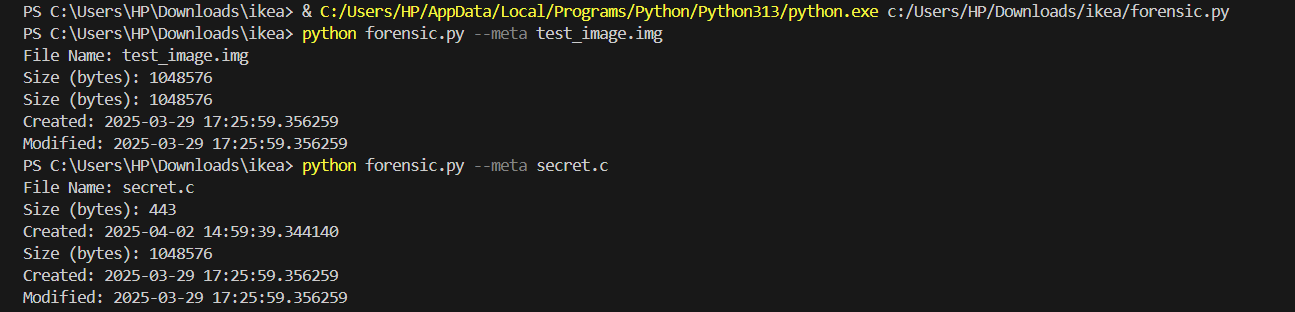


In this the files which we are having is:



With the following commands we can get the metadata analysis of the preferred files

Output:



**5. Analyzing Logs**

**Function**: **analyze\_log(file\_path)**

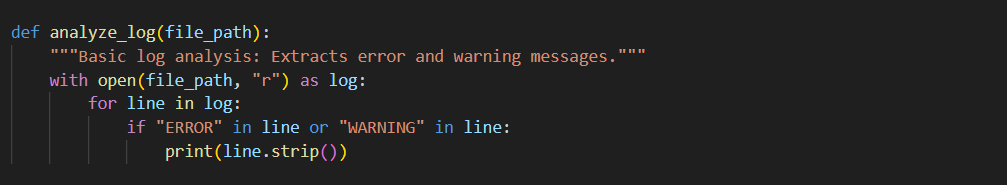
**Use Case**:

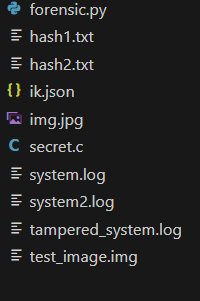
* **Purpose**: To extract error and warning messages from log files.
* **Application**: The organization can use this function to analyze security logs for any entries that indicate unauthorized access attempts, errors, or warnings that may suggest tampering. By filtering the logs for relevant messages, investigators can identify patterns of suspicious behavior, such as repeated failed login attempts or access to sensitive files outside of normal business hours.

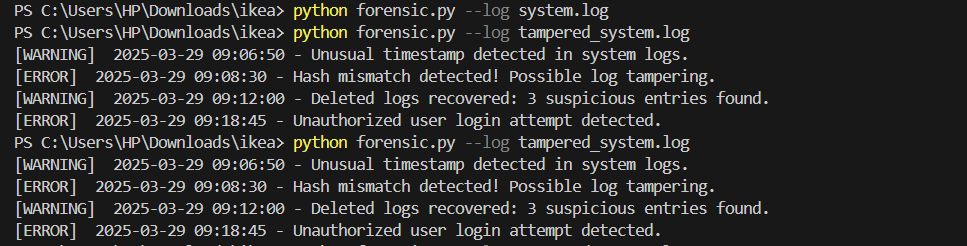
**Log Analysis in the Script**

The script's analyze\_log(file\_path) function scans log files for occurrences of "ERROR" or "WARNING", helping forensic analysts quickly detect critical issues. This can be extended further using:

* **Regular expressions (regex)** to extract specific log patterns.
* **Time-based filtering** to analyze logs within a certain timeframe.
* **Correlation with other logs** to understand multi-step attacks or failures.
* In our case we want to analyse the log files
* To find the system logs of the following file which is named in the system log



****

****

**Conclusion:**

In summary, the digital forensic tool implemented in the provided Python code can be effectively utilized in the investigation of anomalies in security logs and potential data tampering within a financial organization. Each function serves a specific purpose that contributes to the overall forensic analysis, helping to ensure that evidence is collected, preserved, and analyzed in a manner that supports the investigation's objectives. By leveraging these tools, the organization can gain insights into the nature of the anomalies and take appropriate actions to mitigate risks and enhance security.