File Integrity Checker

Cybersecurity Internship – Phase 2, Project 1

Name: Parmar Rakesh
Date:04,SEP,2025
Internship Program: Cybersecurity Internship – UJAR TECH

Introduction

The **File Integrity Checker** is a Python-based tool designed to protect files by detecting unauthorized changes or tampering. It acts like a security guard who takes a "fingerprint" (hash) of your files, saves it securely, and later checks if the files have been altered. This project was created to ensure important files (like documents or images) remain unchanged, which is useful for security-conscious users or system administrators. The goal is to provide a simple, beginner-friendly tool that uses file hashing to monitor integrity, with added features like read-only protection and overwrite confirmation.

Abstract

This project develops a command-line tool that uses the SHA256 hashing algorithm to generate unique fingerprints for files. It has two main modes: **store** (saves hashes in a secure file) and **check** (verifies files against saved hashes). If a file is changed, it alerts the user with a message and logs the event. The tool secures the hash file (Hash.txt) by setting it to read-only and prevents accidental overwrites by prompting the user before replacing the hash file. Logs are maintained in Logs.txt for tracking actions. This tool matters because it helps users detect tampering (e.g., by malware or unauthorized edits), ensuring file security in a simple way.

Tools Used

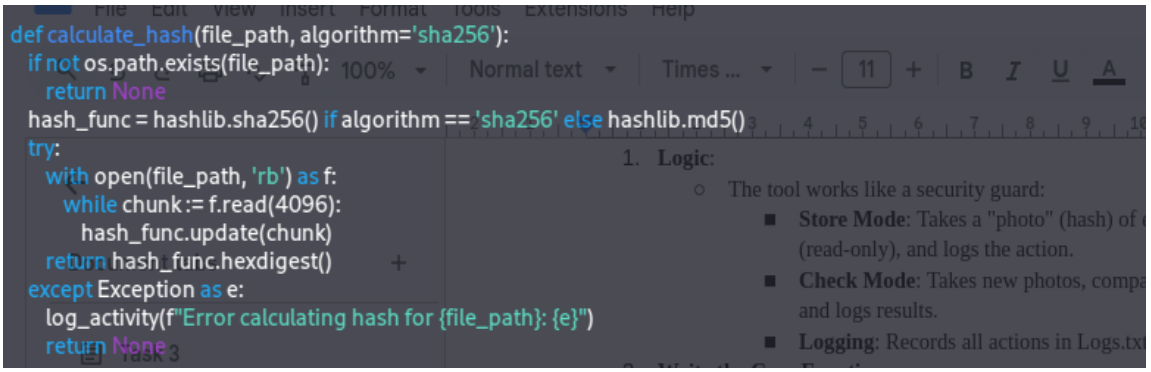
- **Programming Language**: Python 3.x
- **Operating Systems**: Tested on Linux.
- **Libraries**:
    - hashlib: Built-in Python module for generating SHA256 hashes (secure fingerprints).
    - os: Built-in module for file operations (checking existence, reading files).
    - datetime: Built-in module for adding timestamps to logs.
- **Additional Tools**:
    - Terminal (Linux) for running the script.
    - Optional: cron (Linux) for scheduling automatic checks.
- **No external packages**: All dependencies are part of Python's standard library.

Steps Involved in Building the Project

The project was built step-by-step to create a robust file integrity checker. Here's how it was developed:

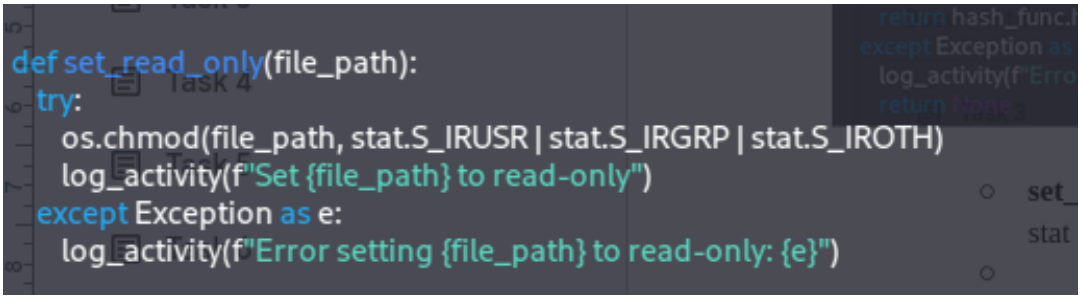
1. **Logic**:
    - The tool works like a security guard:
        - **Store Mode**: Takes a "photo" (hash) of each file, saves it in Hash.txt, locks it (read-only), and logs the action.
        - **Check Mode**: Takes new photos, compares with Hash.txt, alerts if different, and logs results.
        - **Logging**: Records all actions in Logs.txt with timestamps.
2. **Write the Core Functions**:

- **calculate_hash**: Generates a SHA256 hash for a file by reading it in 4KB chunks (to handle large files safely). Returns None if the file doesn't exist or errors occur.



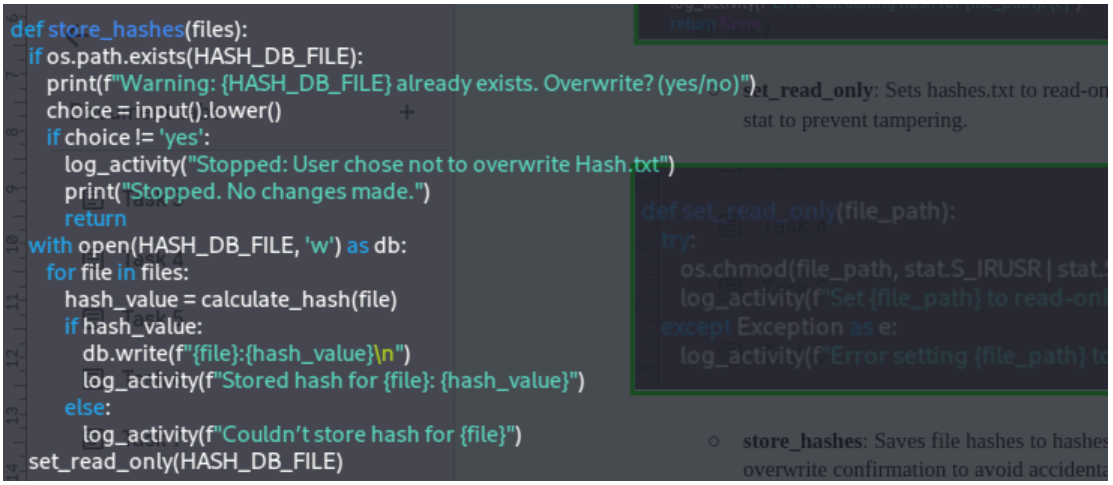
```
def calculate_hash(file_path, algorithm='sha256'):
    if not os.path.exists(file_path):
        return None
    hash_func = hashlib.sha256() if algorithm == 'sha256' else hashlib.md5()
    try:
        with open(file_path, 'rb') as f:
            while chunk := f.read(4096):
                hash_func.update(chunk)
        return hash_func.hexdigest()
    except Exception as e:
        log_activity(f"Error calculating hash for {file_path}: {e}")
        return None
```

- **set_read_only**: Sets Hash.txt to read-only (permissions 444) using os.chmod and stat to prevent tampering.



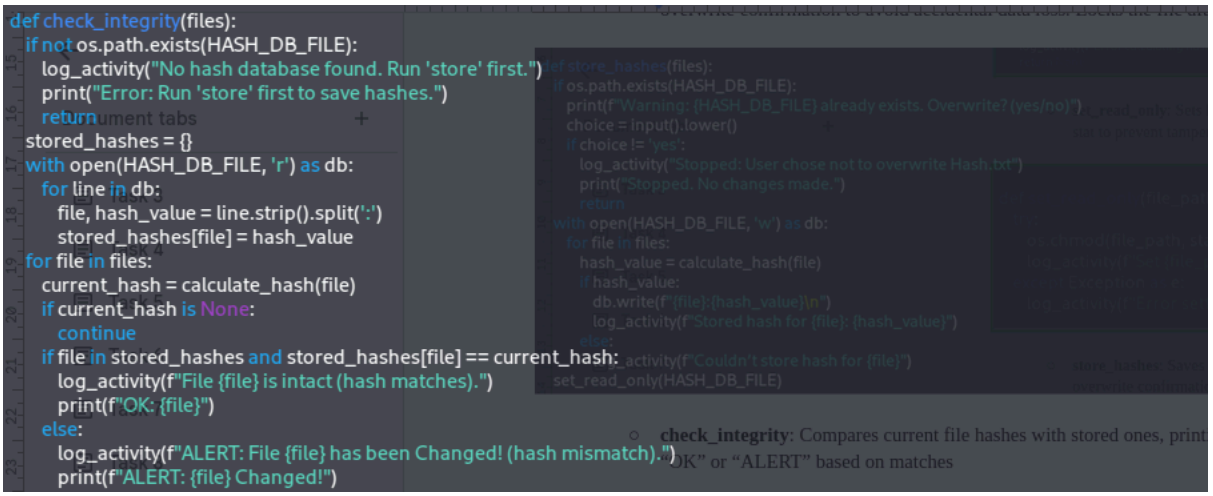
```
def set_read_only(file_path):
    try:
        os.chmod(file_path, stat.S_IRUSR | stat.S_IRGRP | stat.S_IROTH)
        log_activity(f"Set {file_path} to read-only")
    except Exception as e:
        log_activity(f"Error setting {file_path} to read-only: {e}")
```

- **store_hashes**: Saves file hashes to Hash.txt. Checks if the file exists and asks for overwrite confirmation to avoid accidental data loss. Locks the file after saving.

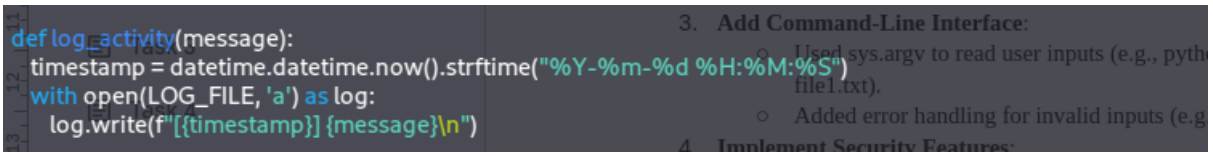


```
def store_hashes(files):
    if os.path.exists(HASH_DB_FILE):
        print(f"Warning: {HASH_DB_FILE} already exists. Overwrite? (yes/no)")
        choice = input().lower()
        if choice != 'yes':
            log_activity("Stopped: User chose not to overwrite Hash.txt")
            print("Stopped. No changes made.")
            return
    with open(HASH_DB_FILE, 'w') as db:
        for file in files:
            hash_value = calculate_hash(file)
            if hash_value:
                db.write(f"{file}:{hash_value}\n")
                log_activity(f"Stored hash for {file}: {hash_value}")
            else:
                log_activity(f"Couldn't store hash for {file}")
    set_read_only(HASH_DB_FILE)
```

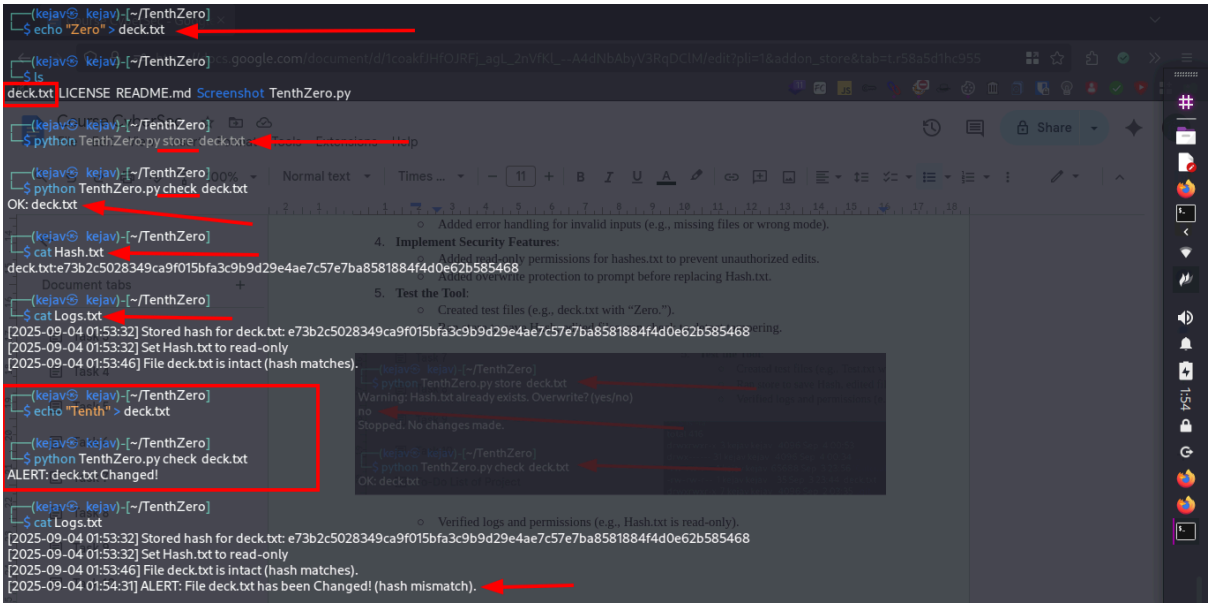
- **check_integrity**: Compares current file hashes with stored ones, printing and logging "OK" or "ALERT" based on matches.



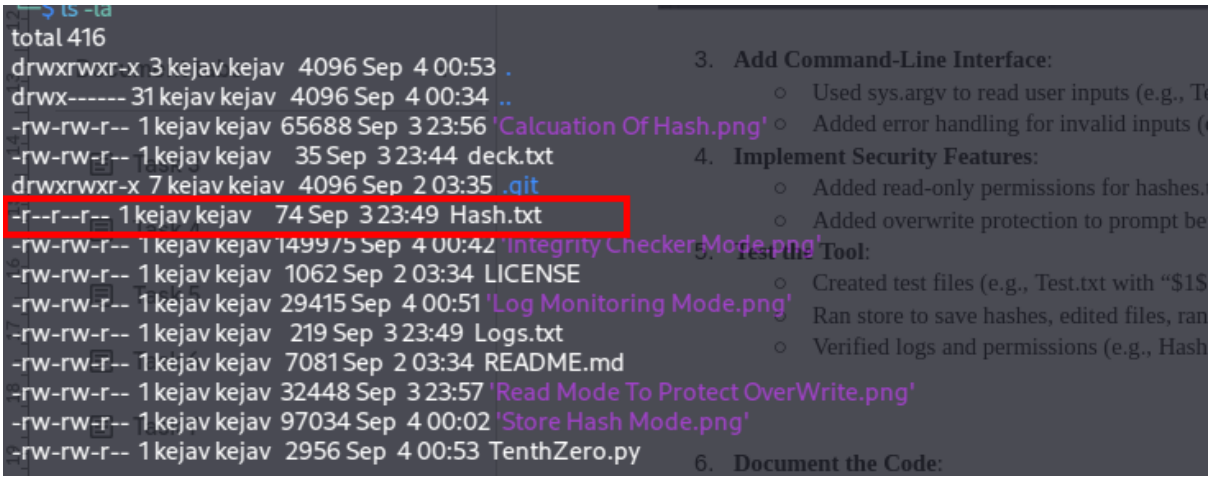
- **log_activity**: Appends timestamps and messages to Logs.txt for tracking.



3. **Add Command-Line Interface**:
    - Used sys.argv to read user inputs (e.g., TenthZero.py store file1 file2 file3 …).
    - Added error handling for invalid inputs (e.g., missing files or wrong mode).
4. **Implement Security Features**:
    - Added read-only permissions for Hash.txt to prevent unauthorized edits.
    - Added overwrite protection to prompt before replacing Hash.txt.
5. **Test the Tool**:
    - Created test files (e.g., deck.txt with "Zero.").
    - Ran store to save Hash, edited files, ran check to detect tampering.



- Verified logs and permissions (e.g., Hash.txt is read-only).



Conclusion

The File Integrity Checker successfully achieves its goal of detecting file tampering using SHA256 hashes. It's simple to use, secure (with read-only Hash.txt), and prevents accidental overwrites. The tool logs all actions, making it easy to track what happened. I learned how to:

- Use Python's hashlib for secure hashing.
- Manage file permissions with os and stat.
- Handle user inputs and errors in a command-line tool.
- Write clear logs for debugging and tracking.

This project is a great starting point for anyone wanting to monitor file integrity, and it can be extended with features like email alerts or folder monitoring.

Challenges Faced

- **File Permissions**: Setting Hash.txt to read-only was tricky on Windows, as os.chmod behaves differently. Using stat.S_IRUSR | stat.S_IRGRP | stat.S_IROTH solved this for cross-platform compatibility.
- **Overwrite Protection**: Initially, the tool overwrote Hash.txt without warning. Adding a user prompt (input()) required careful testing to ensure it didn't break the flow.

References:

1. Python Library
2. SHA - 2
3. How logic Work
4. File Permission