
# PROJECT REPORT:

Creating a GUI based File integrity checking application

**Submitted By:**
Name: Swastik Khan
Roll: 36
Year & Stream: 3rd Year CSE(AIML)
Academic Year: 2024-25
Enrollment No.:12022002016057

**Submitted To:**
Prof. Indrajit De & Prof. Pabak Indu

**Course:** Introduction to Cyber Security

# 1. Introduction

In many environments—be it corporate, educational, or personal—ensuring that critical files remain unaltered is essential. Malicious modifications, unintended changes, or accidental deletions can compromise the reliability and security of data. A **File Integrity Checker** helps detect these issues by creating a baseline (a snapshot of the files' integrity data) and then comparing current file states against this baseline. If any modifications or deletions are detected, the user is alerted.

This project demonstrates a simple yet effective way to perform file integrity checks using Python's `hashlib` for generating cryptographic hash values (SHA-256) and a Tkinter-based graphical user interface (GUI) to make the application user-friendly.

---

# 2. Objectives

1. **Compute Baseline**: Allow users to select one or more files and generate a baseline record of their SHA-256 hash values.
2. **Check Integrity**: Compare the current hash values of these files against the saved baseline to identify modifications or deletions.
3. **User-Friendly Interface**: Provide a simple GUI that can be operated without requiring in-depth knowledge of command-line tools or hashing algorithms.
4. **Error Handling**: Ensure the program can gracefully handle missing files, empty selections, and missing baseline data.

---

# 3. System Overview

## 3.1 Software Architecture

1. **Graphical User Interface (GUI)**:

    - Built using Python's `tkinter` module.
    - Contains:
        - A label to display selected files.
        - Two main buttons: **Compute Baseline** and **Check Integrity**.

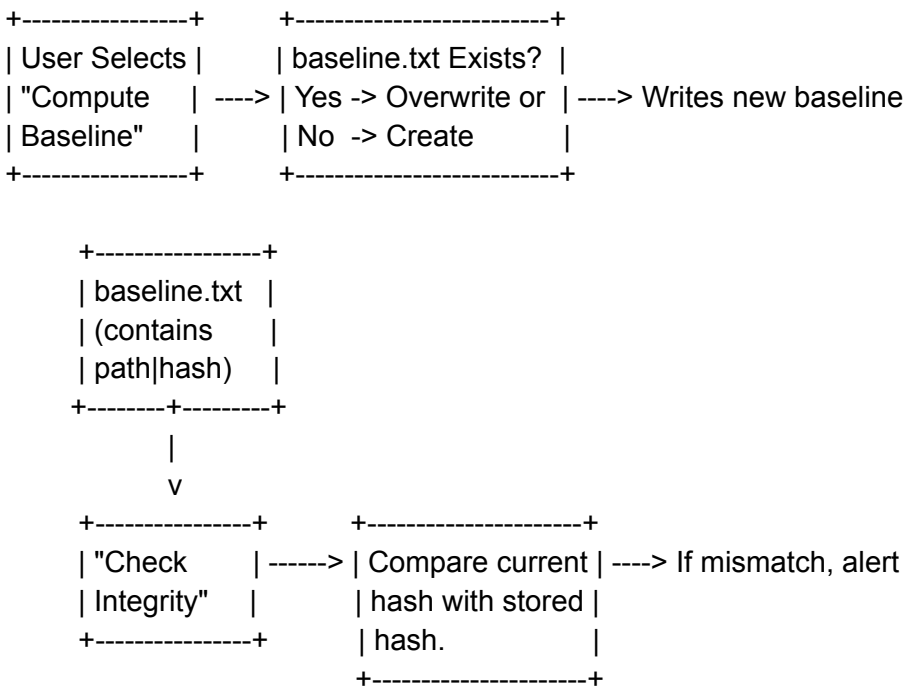
2. **Core Functionalities**:

- **File Selection**: Uses `filedialog.askopenfilenames` to allow the user to select multiple files.
- **Hash Computation**: Utilizes `hashlib.sha256()` to generate SHA-256 hashes of each file's contents.
- **Baseline Management**: Stores file paths and their corresponding hash values in a local file named `baseline.txt`.
- **Integrity Checking**: Reads the baseline data, re-hashes each file, and compares the new hash to the old one.

3. **Data Storage**:

- **baseline.txt**: A text file that holds the path and hash of each file, formatted as `path|hash`.

## 3.2 Flow Diagram



# 4. Implementation Details

## 4.1 Key Functions

1. `calculate_file_hash(filepath)`
    - Uses hashlib.sha256() to compute the SHA-256 hash of the file's contents.
    - Reads the file in chunks (4096 bytes) to handle large files without excessive memory usage.
    - Returns the hexadecimal digest string.
2. `select_files()`
    - Opens a file dialog to let users select one or multiple files.
    - Updates a label in the GUI to display the chosen file paths.
    - Returns a list of the selected file paths.
3. `compute_baseline()`
    - Calls select_files() to get a list of files from the user.
    - For each file, computes the SHA-256 hash and writes the result (path|hash) into baseline.txt.
    - Shows a success message upon completion.
4. `check_integrity()`
    - Verifies the existence of baseline.txt. If it doesn't exist, prompts the user to compute a baseline first.
    - Reads each line from baseline.txt, storing path-hash pairs in a dictionary.
    - For each path in the dictionary:
        - Checks if the file still exists.
        - If it exists, re-computes its hash and compares it with the baseline hash.
        - If there's a mismatch, mark it as **Modified**.
        - If the file doesn't exist anymore, mark it as **Deleted**.
    - Displays a warning if any changes are detected; otherwise, informs the user that all files are unchanged.

## 4.2 Graphical User Interface

- **Window Title**: "File Integrity Checker"

- **Labels**:
    1. A label to display “No files selected” by default, updated with the paths of chosen files.
- **Buttons**:
    1. **Compute Baseline**:
        - Opens a file dialog for file selection.
        - Computes and saves the baseline.
    2. **Check Integrity**:
        - Reads and compares hashes with the baseline.

---

# 5. How to Use

1. **Run the Script**:

    - If you are on Replit, simply click the **Run** button.
    - Locally, run python main.py in a terminal or double-click the script (depending on your operating system setup).
2. **Compute Baseline**:

    - Click the **Compute Baseline** button.
    - In the file selection dialog, choose the files you want to monitor. Click **Open**.
    - A message box confirms the baseline is computed.
3. **Check Integrity**:

    - Make sure baseline.txt exists (i.e., you have already computed a baseline).
    - Click the **Check Integrity** button.
    - The program will re-hash each file and compare with the baseline. A message box will display if any files have been modified or deleted, or if everything is unchanged.

---

# 6. Testing and Validation

1. **No Files Selected**:

    - Attempting to compute a baseline without selecting any files should display an error message.
2. **Baseline Not Found**:
    - Clicking **Check Integrity** before creating a baseline should prompt an error message to create a baseline first.
3. **Modified File**:
    - After computing a baseline, edit a file's contents and click **Check Integrity** to confirm the program detects a "Modified" status.
4. **Deleted File**:
    - After computing a baseline, delete one of the monitored files and click **Check Integrity** to confirm the program detects a "Deleted" status.
5. **Unchanged File**:
    - After computing a baseline, leave the files unchanged. The program should confirm that all files are unchanged.

---

# 7. Limitations and Possible Improvements

1. **Hardcoded Baseline File**:
    - The baseline file name is currently baseline.txt. Allowing the user to specify a different baseline file or location might enhance flexibility.
2. **Lack of Recursive Directory Support**:
    - Currently, users must select individual files. Adding directory selection and automatically hashing all files within a directory would streamline usage.
3. **No Versioning**:
    - The system overwrites baseline.txt each time a new baseline is computed. Implementing versioning (keeping old baselines) could help track changes over time.
4. **Limited Error Logging**:
    - Currently, errors are printed in the console. A more robust logging system could capture these events in a log file for auditing purposes.
5. **Security Considerations**:
    - The program stores paths and hashes in plain text. If needed, additional encryption or secure storage measures could be implemented.

---

# 8. Conclusion

The **File Integrity Checker** project provides a straightforward way to track file integrity using SHA-256 hashes. By offering a user-friendly Tkinter GUI, it lowers the barrier for non-technical users to monitor critical files for unauthorized or accidental changes. The project can be extended or integrated into more comprehensive systems that require audit logs, directory monitoring, or versioning of integrity baselines.

Overall, this tool demonstrates core concepts in integrity verification—hash computation, baseline storage, and change detection—and serves as a foundation for more advanced file monitoring solutions.