**FILE INTEGRITY CHECKER**

Final Report

Cyber Security Assignment- 1

Name: Marpalli Nikhitha Sree

Date: 01-09-2025

Github repository link: <https://github.com/Marpalli-Nikhitha-Sree/file_integrity_checker.git>

**File Integrity Checker – Final Report**

**Project Overview**

This project implements a **File Integrity Checker** in Python using the **hashlib** library and **Gradio** for user interface. The system allows users to register an original file and later verify whether the file has been altered. It uses the **SHA-256 hashing algorithm**, which generates a unique digital fingerprint of the file. If the hash of a file changes, it indicates that the file has been tampered with.

The project demonstrates the concept of **data integrity** and highlights the role of hashing in **cybersecurity**, especially in detecting unauthorized modifications to sensitive files. The solution is designed with simplicity, user-friendliness, and practical applicability in mind.

**Technologies & Tools Used**

* **Python 3.10+** – Core programming language.
* **hashlib** – For generating SHA-256 hashes of files.
* **Gradio** – To build an interactive web-based user interface.
* **Google Colab / Jupyter Notebook** – For development and execution.
* **GitHub** – For version control and sharing the project.

**Work Flow**

1. The user uploads a file in the Gradio interface.
2. The user selects a mode:
   * **Register Original** – The file’s SHA-256 hash is generated and stored.
   * **Check Integrity** – A new hash is generated and compared with the stored hash.
3. Based on the comparison:
   * If hashes match → File is **original**.
   * If hashes differ → File is **tampered**.
4. Results (hash value and verification message) are displayed in the interface.

**Security Features**

* **SHA-256 Hashing:** Uses a cryptographic hash function to ensure strong integrity verification.
* **Tamper Detection:** Detects even the smallest modification in file content.
* **One-way Function:** Hash values cannot be reversed to obtain original content, ensuring security.
* **Simple UI:** Easy-to-use interface for both technical and non-technical users.

file-integrity-checker/

│── README.md # Project documentation

│── file\_integrity\_checker.ipynb. # Main notebook

**Screenshots**

**Registering a file as original:**

A screenshot of a computer

AI-generated content may be incorrect.

Checking the integrity of original file:

A screenshot of a computer

AI-generated content may be incorrect.

Checking the integrity of the edited original file which is not registered:

A screenshot of a computer

AI-generated content may be incorrect.

**Testing &Results**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Test Case** | **File Name** | **Mode** | **SHA256 (first 15 chars)** | **Result** | | 1 | original.jpeg | Register Original | c4d7db82d6305c3… | ✅ Original file registered | | 2 | original.jpeg | Check Integrity | c4d7db82d6305c3… | ✅ File is ORIGINAL | | 3 | edited.jpeg | Check Integrity | eba50ed9a90aa42… | ❌ File has been TAMPERED | |

**Observations**

* When the **original file** was registered, its SHA256 hash was stored successfully.
* Running the integrity check on the **same file** produced the exact same hash, confirming the file was **untampered**.
* Running the integrity check on the **edited version** of the file generated a **different hash**, immediately flagging the file as **tampered**.
* Even small changes in file content (like resizing or editing metadata) completely altered the hash value, showing the reliability of the checker.

**Deliverables**

* Python script (app.py) with Gradio interface.
* GitHub repository with full code and documentation.
* Final security report (this document).

**Learning Outcomes**

* Gained hands-on experience with **hash functions** (SHA-256).
* Understood the concept of **file integrity verification** in cybersecurity.
* Learned how to build a simple **Gradio-based UI** for file operations.
* Improved ability to apply Python for **practical security applications**.

**Conclusion**

This project demonstrates the use of **hashing algorithms for file integrity checking**. By comparing the hash of an uploaded file with the original, the system accurately detects tampering. The project highlights the importance of **data integrity, trust, and authenticity** in cybersecurity.

The File Integrity Checker provides a simple yet powerful demonstration of how **cryptographic hashing** can protect against unauthorized file modifications.