

HashCheck - File Integrity & Change Detection Tool

Final Project - Written Report

IT 360, Section 001

December 3, 2025

Introduction

Digital forensic investigations will often begin with establishing a clear picture of what may have been changed on a system. Whether responding to malware, unauthorized access, insider threats, or accidental damage, investigators must quickly determine if or what files were modified, deleted, or newly created. Tracking these changes is essential for understanding the scope of an incident and potentially finding a pivot point.

The purpose of our project, HashCheck, is to provide a simple, lightweight tool that allows a user to generate a baseline of file hashes within a directory and later verify that directory for changes. This solves a core digital forensics problem: maintaining file integrity and identifying when files have been tampered with, modified without authorization, dropped by malware, or suddenly deleted.

While professional integrity-monitoring tools exist, many are complex, require installation, or are too heavy for personal or student use. HashCheck was built to give students and analysts a fast, Windows-friendly, easy-to-use tool for practicing real DFIR workflows.

Technical Implementation

Languages, Libraries & Rationale

HashCheck is built entirely with Python 3, chosen for its readability and strong standard library support.

Key components include:

- `hashlib`
 - Used to compute SHA-256, SHA-1, or MD5 hashes. Chosen because it's built-in, reliable, and widely used in forensics.

- `os, pathlib`
 - to traverse files and gather metadata.
- `csv`
 - To generate machine-readable reports for baselining and verification.
- `datetime`
 - To record modification times in UTC.
- `Tkinter`
 - Chosen because it ships with Python on Windows, requires no installation, and allows simple GUI configuration.

No external dependencies were used, making the tool portable and lightweight.

How HashCheck Works

The backend script (`hashcheck.py`) handles the core forensic logic, with two modes:

Scan Mode:

1. User selects a folder
2. Program walks through all files (optionally recursive)
3. Each file is hashed in chunks (4MB at a time) to support large files.
4. Metadata is collected:
 - File path
 - Size in bytes
 - Last modified timestamp
 - Hash digest
5. Results saved to a CSV “manifest”, your new baseline!

Verification Mode:

1. User selects the same folder and a previous baseline.
2. Files are re-hashed and compared.
3. Each file is labeled:
 - OK | Unchanged
 - MISMATCH | Modified (hash changed)
 - NEW | Was not in the baseline
 - MISSING | Baseline file is no longer present

- ERROR | File couldn't be read or hashed

This process mirrors real-world forensic workflows used to detect tampering and malware activity.

Graphical User Interface

The GUI (`hashcheck_gui.py`) provides a clean interface with:

- Two tabs: Scan and Verify
- File/Folder pickers
- Algorithm selection
- Recursive option
- Output CSV selection
- Color-coded results table (red, orange, gray, etc.)
- Export-to-CSV button
- Background worker threads so the GUI stays responsive

Tkinter styles were heavily customized to produce a clean, modern look suitable for a forensic tool.

Results

HashCheck produces two types of output:

1. Baseline Manifest:

path	size_bytes	mtime_utc	algo	hash	status
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file1.txt	122	2025-12-02T20:33:39Z	sha256	56475955af59050313703659da5a203b4190e7849a0c227bf261802b8b6695	BASELINE
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file2.txt	116	2025-12-02T20:33:39Z	sha256	0ea3413c1338569e35c44535f08acb71ff9e1052e70d92b44557b9eae457014e	BASELINE
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file3.txt	2390	2025-12-02T20:33:39Z	sha256	1682bb77ef95a0f79e18bbe4a196cfa3395623981cbfd90be56a5c712039c2b	BASELINE
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file4.txt	11	2025-12-02T22:23:07Z	sha256	3009dfd8ef5153c336be96c5a3adc4c381b901fe677d9b0215d936ab6b31026	BASELINE

This establishes the "expected" state of the directory.

2. Verification Report:

path	size_bytes	mtime_utc	algo	hash	status	error
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\malware.txt	0	2025-12-02T22:29:20Z	sha256	e2b0c44299fc1c149afbd4c8996fb03427ae1e640b924ca405991b7852b855	NEW	
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file1.txt	122	2025-12-02T20:33:39Z	sha256	56475955af59050313703659da5a203b4190e7849a0c227bf261802b8b6695	OK	
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file2.txt	91	2025-12-02T22:29:11Z	sha256	2d5550e1baad133002e578ab01ba756b89865ead4fd6cb70239cde20d5	MISMATCH	expected 0ea3413c1338...
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file3.txt	2390	2025-12-02T20:33:39Z	sha256	1682bb77ef95a0f79e18bbe4a196cfa3395623981cbfd90be56a5c712039c2b	OK	
C:\Users\Admin\OneDrive\Desktop\Cyber\Tools\HashCheck\data\test-file4.txt	11	2025-12-02T22:23:07Z	sha256	3009dfd8ef5153c336be96c5a3adc4c381b901fe677d9b0215d936ab6b31026	MISSING	

The GUI displays these results with the following colors:

- Red - MISMATCH
- Orange/Yellow - New File
- Gray - Missing File
- Purple - Error
- Default - Ok

Lessons Learned

What Worked Well

- Python's built-in libraries made hashing simple and reliable.
- Tkinter allowed for the creation of a functional GUI without the need for any external frameworks.
- Background tasks prevented the GUI from freezing during long scans.
- The CSV format made it easy to view results in Excel.
- The tool ended up being very lightweight, portable, and easy to run.

Challenges

- Cleaning up the GUI
- We tried to automatically check the hashes with VirusTotal, but in order to do that, we would need an API key, so we didn't follow through with that feature.

What We Would Improve or Add in the Future

- Add filtering and searching within the results table.
- Add hash comparison for entire directories via drag-and-drop.
- Add exclusions to improve scan speed (skip certain file types or folders).
- Support macOS/Linux officially.
- Implement digital signatures for the manifest.
- Add the ability to schedule recurring scans.
- Fix the "File Selection" entry with a button to select a file or a directory.

Conclusion:

HashCheck demonstrates a core concept in digital forensics: creating system baselines and verifying file integrity. Even though the tool is intentionally lightweight and simple, it

mirrors real techniques used by forensic analysts and incident responders to spot file tampering, malware activity, or unauthorized changes on a system. Building this project gave us hands-on experience scripting with Python, GUI development, threading, hashing algorithms, and turning cybersecurity concepts into an actual working tool.