PoC:Steganographic File Integrity Checker

Name:Samiksha Patil.

Intern ID:280

Introduction

File integrity is crucial in cybersecurity to ensure that important files (documents, source code, reports, or system binaries) have not been tampered with. Traditional methods use hashes (SHA256, SHA3, etc.) stored separately in a .txt file or a checksum database.

But the problem:

1.Attackers can modify or delete the checksum file.

2.Integrity logs stored separately can also be targeted.

Solution: Hide the hash inside an innocent-looking cover file (image/audio) using steganography. Even if an attacker tampers with the file, they will not know about the hidden integrity data.

Objectives

1.Generate cryptographic hash (SHA256/SHA3) of sensitive files.

2.Hide these hashes inside cover files (PNG, JPG, WAV).

3.Extract hashes later and verify with fresh hash values.

4.Demonstrate detection of modified files.

5.Provide a lightweight, user-friendly tool.

Features

1. Cryptographic Security: Uses SHA256/SHA3 (strong against collisions).

2.Steganography: Hidden inside least significant bits (LSB) of image/audio.

3.Cover Media: PNG/JPG for image, WAV for audio.

4. Integrity Check: Detects file modification.

5.Lightweight: No heavy dependencies, pure Python + Pillow/wave.

Sample Run

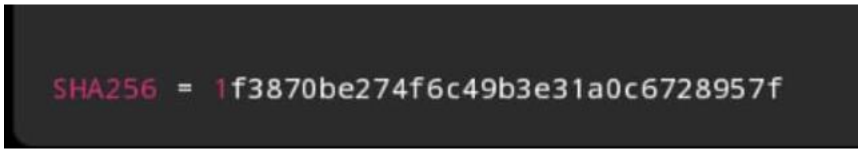
Input Files:

1.Target file: report.pdf

2.Cover file: cover.png

Process:

1. Generate hash of report.pdf:



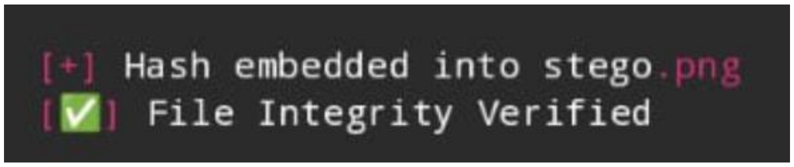
```
SHA256 = 1f3870be274f6c49b3e31a0c6728957f
```

2. Embed hash inside cover.png → produces stego.png.

3. Extract hash from stego.png.

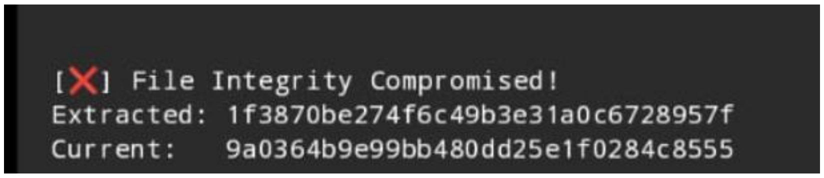
4. Compare with current hash of report.pdf.

Output:



```
[+] Hash embedded into stego.png
[✅] File Integrity Verified
```

If report.pdf is modified:



```
[❌] File Integrity Compromised!
Extracted: 1f3870be274f6c49b3e31a0c6728957f
Current:   9a0364b9e99bb480dd25e1f0284c8555
```

Example PCAP/Files for Testing

-cover.png (innocent image)

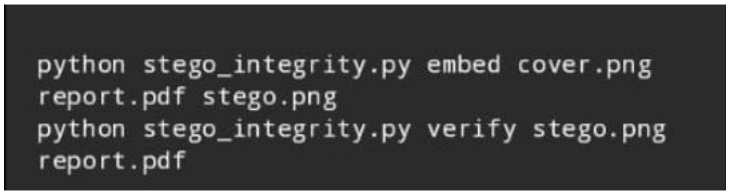
-report.pdf (target file)

-stego.png (steganographic file with hash hidden)

-Modified report.pdf (tampered version)

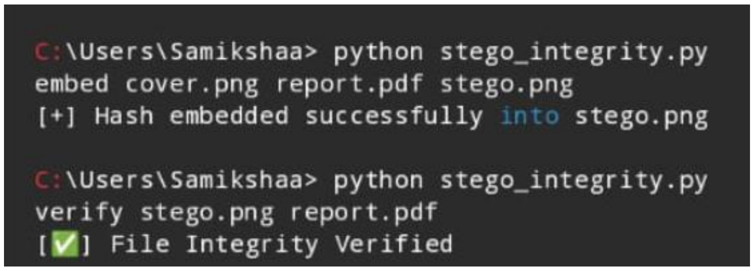
Running the Script

Command-Line Usage:



```
python stego_integrity.py embed cover.png
report.pdf stego.png
python stego_integrity.py verify stego.png
report.pdf
```

Sample CMD Screenshot (expected):



```
C:\Users\Samikshaa> python stego_integrity.py
embed cover.png report.pdf stego.png
[+] Hash embedded successfully into stego.png

C:\Users\Samikshaa> python stego_integrity.py
verify stego.png report.pdf
[✅] File Integrity Verified
```

Advantages

1. Hidden security → attacker won't suspect cover files.
2. Detects even 1-bit modification in target files.
3. Works with images/audio (multiple carriers).
4. Easy to extend with GUI/automation.

Limitations

1. Stego file size slightly increases.
2. If cover file is recompressed (e.g., JPG lossy compression), hidden data may be lost.
3. Not suitable for cloud-based auto-resized images (hash may vanish).

Future Improvements

1. Encrypt hidden hashes before embedding.
2. Batch mode → check multiple files at once.
3. GUI (Tkinter/PyQt).
4. Extend to audio steganography (WAV/MP3).
5. Cloud backup of stego files for distributed verification.

Deliverables

1. Code (stego_integrity.py)
2. Sample Cover Files (cover.png, cover.wav)
3. Documentation (Word/PDF report)
4. Demo video/screenshots
5. Unit tests (pytest)