Name: Ankita Kolapte
Intern ID:270

Proof of Concept (PoC) - Stenographic File Integrity Checker

1. Introduction

This Proof of Concept (PoC) demonstrates the implementation of a Stenographic File Integrity Checker. The system ensures file integrity by generating cryptographic hashes (SHA256) of target files and embedding these hashes into cover files (images) using Least Significant Bit (LSB) steganography. Later, the embedded hash can be extracted and compared against the current file hash to verify whether the file has been modified.

2. Objectives

The tool should:

- - Generate SHA256 hash of one or more target files.
- - Embed these hashes into a cover image file.
- - Extract hidden hashes from the cover image.
- - Compare extracted hash with the current file hash to detect modification.

3. Implementation

The implementation is done in Python using the hashlib library for cryptographic hashing and Pillow (PIL) for image manipulation. Hash values are converted into binary form and hidden inside the least significant bits (LSB) of the image pixels. During verification, the embedded binary data is extracted back from the image and compared with the current hash of the target file.

4. Core Implementation Code (Simplified)

```
from PIL import Image
import hashlib
```

```
# Generate SHA256 hash
def get_file_hash(filename):
    h = hashlib.sha256()
    with open(filename, "rb") as f:
        while chunk := f.read(4096):
            h.update(chunk)
    return h.hexdigest()

# Embed hash into cover image (LSB)
def embed_hash_in_image(cover_image, output_image, hash_str):
    img = Image.open(cover_image)
    binary_hash = ''.join(format(ord(c), '08b') for c in hash_str)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    pixels = img.load()
    width, height = img.size
    data_index = 0
    for y in range(height):
        for x in range(width):
            if data_index >= len(binary_hash): break
            r, g, b = pixels[x, y]
            if data_index < len(binary_hash): r = (r & ~1) |
int(binary_hash[data_index]); data_index += 1
            if data_index < len(binary_hash): g = (g & ~1) |
int(binary_hash[data_index]); data_index += 1
            if data_index < len(binary_hash): b = (b & ~1) |
int(binary_hash[data_index]); data_index += 1
            pixels[x, y] = (r, g, b)
        if data_index >= len(binary_hash): break
    img.save(output_image)

# Extract hash from stego image
def extract_hash_from_image(stego_image, hash_length=64):
    img = Image.open(stego_image)
    pixels = img.load()
    width, height = img.size
    binary_hash = ""
    for y in range(height):
        for x in range(width):
            r, g, b = pixels[x, y]
            binary_hash += str(r & 1) + str(g & 1) + str(b & 1)
            if len(binary_hash) >= hash_length * 8: break
        if len(binary_hash) >= hash_length * 8: break
```

```
chars = [binary_hash[i:i+8] for i in range(0, len(binary_hash), 8)]
return ''.join(chr(int(c, 2)) for c in chars)[:hash_length]
```

---

5. Demo & Testing

The Stenographic File Integrity Checker was tested using the following workflow:

- - Selected a target file (e.g., report.pdf).
- - Generated SHA256 hash of the file.
- - Chose a cover image (e.g., cover.png) and embedded the hash.
- - Extracted the hidden hash from the stego image.
- - Compared the extracted hash with the current file hash to verify modification.

6. Deliverables

- - Demo execution showing embedding and extraction of hashes.
- - Short report (this document) describing embedding, extraction, and verification logic.
- - Unit tests for hash generation, embedding, extraction, and verification functions.

7. Conclusion

This PoC successfully demonstrates a simple method for ensuring file integrity using steganography. By embedding cryptographic hashes into image files, any file modification can be detected. This technique can be extended to support audio files, multiple hash algorithms, and user-friendly GUIs for broader applicability in real-world use cases.

Submitted To: Digisuraksha Parhari Foundation.