CMPS426 - Security of Computer Systems and Networks

Cryptography Course Project

Submitted by:

Karim Magdy Mounir	1210070
Abdelrahman Ahmed Mohamed	1210248
Ahmed Essam	1210346
Mariam Essam Mohamed	1180027

Submitted to: Eng. Ayman
Eng. Mahinour

Date of Submission: 04/05/2025

Module (1)

1. Introduction

1.1 Background and Motivation

Ransomware represents a critical cybersecurity threat worldwide. Understanding its mechanisms is essential for network defenders, developers, and security educators. *FileSecurityDemo* simulates core ransomware behaviors—file encryption, key storage and retrieval, user interaction—within a safe, controlled environment. It serves as both a learning aid and a testbed for defensive strategies.

1.2 Scope and Objectives

- Implement robust file encryption using AES-256 in CBC mode.
- Obfuscate and securely store encryption keys and integrity hashes within large random-padding files.
- Provide a user interface (Tkinter) for ransom notification and decryption workflow.
- Package the tool into a standalone executable (PyInstaller) for easy distribution.
- Measure performance: encryption/decryption throughput, resource usage, and integrity verification accuracy.

2. Architecture and Design

2.1 System Overview

FileSecurityDemo follows a modular design:

1. **Encryption Module:** Processes files in 1MB chunks, applies PKCS#7 padding, and streams ciphertext to new .encblob files.

2. **Key Management:** Generates 32-byte AES keys and 16-byte IVs. Obfuscates them via rotating XOR and stores in a 10MB dummy file at fixed offset.

3. **Integrity Verification:** Computes a SHA-256 hash of all target files before encryption, stores the hash similarly, and verifies post-decryption.

4. **GUI Interaction:** Uses Tkinter for two windows: a ransom note during decryption attempts and a post-encryption warning GUI.

5. Packaging: Bundles Python script and assets into a single executable via PyInstaller.

2.2 Key Components

- **Encryption Module:** encrypt_file / decrypt_file methods with chunked streaming.
- **Key Obfuscator:** _xor_obfuscate rotates through byte masks.
- **Hash Computation:** compute_folder_hash and verify_folder_integrity for tamper detection.
- **GUI Classes:** show_ransom_gui and show_post_encrypt_gui deliver interactive dialogs.
- **Command Handler:** handle_encryption orchestrates encryption/decryption logic based on target path state.

3. Implementation Details

3.1 Encryption Algorithm (AES-CBC)

- Utilizes cryptography library's AES cipher in CBC mode.
- Secure key and IV length: 256-bit key, 128-bit IV.
- Chunk size: 1MB to balance memory usage and throughput.

3.2 Chunked File Processing

- Stream files to avoid loading entire content into memory.
- Pad each chunk via PKCS#7, update padder across chunks, finalize at EOF.

3.3 Key and Hash Obfuscation Technique

Key+IV combined into 48 bytes, XOR-obfuscated with index-based rotating mask.

Stored within a 10MB random-padding file at 1MB offset (.hidden_ransom_key.txt).

Folder hash stored likewise at 2MB offset (.hidden_ransom_hash.txt).

3.4 GUI Design (Tkinter)

- **Ransom Note GUI:** Prompt for manual key entry or "Check Payment" option.
- **Post-Encryption Warning GUI:** Scary message, countdown, and guidance to rerun tool.
- **Assets:** Icon displayed in window, color scheme (black/red) for psychological realism.

3.5 Packaging with PyInstaller

Command:

```
pyinstaller --onefile --name "FileSecurityDemo" --distpath "../dist" --workpath "../build" \
  --add-data "../assets/images.ico;assets" --hidden-import cryptography --noconsole --clean
main.py
```

4. Results and Discussion

4.1 Observations

- AES-CBC with chunked streaming delivers consistent performance for large files.
- GUI responsiveness unaffected by background encryption threads.

4.2 Limitations

- Lack of real network-based payment verification.
- No anti-tamper against key file deletion or modification.

4.3 Improvements

- Integrate asynchronous encryption to avoid UI blocking.
- Add secure channel (HTTPS) for simulated payment checks.
- Replace XOR obfuscation with authenticated encryption for key storage.

Module (2)

1. Overview

This module implements a phishing email campaign simulation designed to demonstrate common social engineering attack vectors. The script automates sending malicious emails with attachments while impersonating a legitimate organization (Vodafone).

2. Key components

2.1 Email Infrastructure

- **MTP Configuration:** Uses Gmail's SMTP server (smtp.gmail.com:587) with TLS encryption
- **Sender Account:** Authenticates with credentials (username/password)
- **Bulk Sending:** Processes a list of recipient emails from a text file

2.2 Message Construction

- **Spoofed Identity:** Masquerades as "Vodafone Recruitment Team"
- **Persuasive Content:**
  - Text and HTML versions of the message
  - Professional styling matching corporate communications
  - Urgency cues ("Deadline to submit")
- **Social Engineering Elements:**
  - Fake internship opportunity
  - Gamified assessment lure
  - Instructions to download/install attached "software"

3. Technical Implementation Details

3.1 Core Functions

- **send_phishing_emails():** Main function orchestrating the phishing campaign
  - Reads victim emails from text file
  - Establishes SMTP connection
  - Constructs and sends individualized messages

3.2 Message Features

- Mixed MIME type (text + HTML + attachment)
- Professional HTML styling
- Fake unsubscribe header
- Consistent branding (Vodafone colors, logos)

Module (3)
Ransomware Artifact Detection

1. Motivation

While our main tool simulates ransomware behavior, it's equally critical to explore detection techniques that identify ransomware-like patterns. One effective heuristic is **entropy analysis**, which estimates randomness in file contents—a characteristic of encrypted or obfuscated files.

1.2 Implementation Overview

We developed a `RansomwareScanner` module that recursively scans a target directory for ransomware-related artifacts based on a combination of static analysis heuristics and signature-based detection. The scanner identifies suspicious files and executables using the following key indicators:

- **Encrypted extension match**: Files ending in `.encblob` are flagged as potential ransomware-encrypted files.
- **Hidden key file signature**: Files named `.hidden_ransom_key.txt` are indicative of hidden key storage used by ransomware.
- **Suspicious extensions**: Files with extensions such as `.fun`, `.dog`, `.wcry`, `.locked`, and `.payforunlock` are flagged.
- **High entropy values**: Files with Shannon entropy above 7.5 are suspected of being encrypted or compressed.
- **Executable analysis**: For `.exe` files, the scanner performs detailed analysis using:
    - **Imported DLLs**: Checks for the presence of DLLs commonly used by ransomware (e.g., `advapi32.dll`, `bcrypt.dll`, `crypt32.dll`).
    - **API calls**: Flags usage of suspicious functions like `CryptEncrypt`, `DeleteFile`, and `HttpSendRequest`.
    - **Section name anomalies**: Detects non-standard section names in PE files.
    - **Static string detection**: Extracts readable strings and flags those containing suspicious keywords (e.g., “ransom”, “pay”, “vssadmin delete shadows”).

- **YARA-based matching**: Custom YARA rules are used to detect known ransomware behavior signatures within files.

The scanner generates a structured report listing flagged files along with the reason for suspicion, enhancing the ability to proactively detect potential ransomware threats.

here is an example of an output for TestCases_safe:


```
Enter the directory path to scan: C:\Users\ahmad\Desktop\Spring25\Security\Project\TestCases_Safe
Scanning folder for ransomware patterns...
Scanned 30 files. Flagged 27 as suspicious.
Possible ransomware artifacts detected:
[
  {
    "file": "C:\\Users\\ahmad\\Desktop\\Spring25\\Security\\Project\\TestCases_Safe\\AifEditor.exe",
    "indicator": [
      "Suspicious DLL imported: advapi32.dll",
      "Suspicious DLL imported: kernel32.dll",
      "Suspicious function used: GetProcAddress",
      "Unusual section name: .gfids",
      "Error extracting strings: [WinError 2] The system cannot find the file specified"
    ]
  },
  {
    "file": "C:\\Users\\ahmad\\Desktop\\Spring25\\Security\\Project\\TestCases_Safe\\AtlTraceTool8.exe",
    "indicator": [
      "Suspicious DLL imported: advapi32.dll",
      "Suspicious DLL imported: kernel32.dll",
      "Suspicious function used: GetProcAddress",
      "Error extracting strings: [WinError 2] The system cannot find the file specified"
    ]
  },
```