International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Ransomware Simulation Using Python

[1]Mr. Hard Pandya, [2]Ms.Vedrucha Pandya

[1]Undergraduate Student in Cybersecurity at Gujarat University- CPC Department, Ahmedabad, Gujarat, India
[2]Assistant Professor at Aditya Silver Oak University, Ahmedabad, Gujarat, India

Abstract
This project introduces a safe and controlled ransomware simulation built in Python for cybersecurity education and training. It replicates key functionalities of real-world ransomware—such as Fernet-based file encryption, a GUI ransom note with a countdown timer, and simulated server communication—within a secure virtual machine environment. Designed with strict ethical safeguards, the simulation enables hands-on exploration of ransomware behavior without causing actual harm. It serves as a practical tool for students, researchers, and security professionals to better understand, analyze, and develop defenses against modern ransomware threats.

Keywords: Ransomware Simulation, Cybersecurity Education, File Encryption, Fernet, Python, Flask, Virtual Machine, Countdown Timer, Ransom Note, Ethical Hacking, Malware Behavior Analysis, Threat Simulation, Defensive Security Training.

1. Introduction

1.1 Background

Ransomware is a growing cybersecurity threat, causing data loss, financial damage, and service disruption across sectors by encrypting files and demanding payment for decryption.

Studying real ransomware is risky and unethical, as it can lead to uncontrolled infections and legal issues, making safe, hands-on education difficult.

There is a lack of practical tools for safe exploration, creating a gap in cybersecurity education where learners can't interact with realistic ransomware behavior without danger.

This project addresses that gap by simulating ransomware in a virtual environment, using Python and Flask to mimic encryption, ransom notes, countdown timers, and server interaction—without harming real systems.

1.2 Problem Statement

- Real ransomware attacks cause significant damage by encrypting files and demanding ransom, but studying their behavior directly poses security risks.
- There is a lack of safe, practical tools that allow students and researchers to analyze ransomware techniques without exposing systems to actual threats.
- Most educational approaches rely on theoretical explanations, which fail to show the real-world impact and urgency created by ransomware threats.

1.3 Objectives

- To develop a safe and controlled simulation of ransomware behavior, including file encryption, ransom note display, and countdown mechanisms.
- To provide a practical tool for cybersecurity education, enabling students and researchers to study ransomware functionality without using real malware.
- To demonstrate the impact of ransomware attacks by simulating how files are locked and how attackers pressure victims through time-limited ransom demands.
- To enhance awareness and defensive understanding by allowing experimentation with potential detection, response, or recovery strategies in a secure environment.

1.4 Contributions

- A Python-based ransomware simulation framework using Fernet and Flask.
- Comprehensive testing in a virtual machine with logged outcomes.
- Insights for cybersecurity education and defense strategies.

For this study, secondary data has been collected from reliable online sources. Monthly data on ransomware-related cybersecurity incidents and trends was obtained from reputable cybersecurity databases and threat intelligence platforms covering the period from January 2020 to December 2024. Technical data and tools used for simulating ransomware behavior, such as encryption methods and malware analysis techniques, were gathered from open-source cybersecurity repositories and documentation. In addition, supporting information on encryption standards and security protocols was referenced from official documentation provided by libraries like Python's cryptography package. This time series data was crucial in designing and validating the ransomware simulation model over a consistent five-year period.

2. Literature Review

2.1 Ransomware Overview

Ransomware is a type of malicious software that restricts access to a victim’s data, typically by encrypting files, and demands a ransom payment for decryption. The first known ransomware attack, the AIDS Trojan (1989), used rudimentary symmetric encryption and spread via floppy disks (Young & Yung, 1996). Since then, ransomware has become increasingly sophisticated, with modern variants such as WannaCry (2017) and REvil employing strong cryptographic algorithms like AES and RSA, along with network propagation mechanisms (Kharraz et al., 2015; Scaife et al., 2016).

Modern ransomware is classified mainly into two types: locker ransomware, which locks access to the system interface, and crypto-ransomware, which encrypts user files. The latter has become more prevalent due to the irreversible damage it can cause without a decryption key. These attacks often use social engineering (e.g., phishing emails) or exploit system vulnerabilities for infection and propagation (Rathore et al., 2017).

Understanding how ransomware operates—through encryption, communication with command-and-control servers, and ransom note display—is essential for both prevention and mitigation. Hence, simulating ransomware in a safe and controlled environment is a critical approach in cybersecurity research and education.

2.2 Existing Simulation Tools

Several tools and frameworks exist for malware and ransomware analysis, though few are tailored specifically for educational simulations:

- Cuckoo Sandbox: An automated malware analysis system that provides insight into ransomware behavior through dynamic analysis in virtual environments. While effective for research, it focuses more on detection than controlled simulation (Guarnieri & Bremer, 2012).
- GamorSEC Ransomware Simulator: A lightweight tool that mimics ransomware behavior for training purposes. However, it is limited in customization and does not include realistic encryption or C2 communication models.
- RansomwareSim (by KnowBe4): Designed for awareness training, this tool simulates ransomware-style activity (e.g., file renaming and ransom notes) without real encryption. It's useful for enterprises but lacks technical depth required for research-level simulations.
- Custom Academic Simulations: Some academic projects develop custom Python-based ransomware models using libraries like cryptography for encryption and GUI libraries for ransom note simulation. These are often unpublished or used internally in universities for cybersecurity training (Alasmary et al., 2020).

2.3 Gap Analysis

Despite growing interest in ransomware defense, several gaps exist in current literature and tools:

- Lack of Open-Source Educational Simulators: Most existing simulators are either proprietary or limited in scope. There is a need for openly available ransomware simulation tools that allow students and researchers to study realistic attack behavior safely.
- Limited Focus on Encryption Implementation: While encryption is the core mechanism in ransomware, many simulators avoid implementing real encryption due to ethical concerns. As a result, there is a gap in understanding the actual impact and technical structure of encryption within simulated ransomware environments.
- Absence of Full Lifecycle Simulation: Few tools model the full ransomware lifecycle-encryption, ransom note display, countdown timer, and simulated communication with a C2 server. A

complete simulation model would provide deeper educational value and allow for more comprehensive defense training.

- Outdated or Simplified Behavior: Many tools simulate older ransomware variants or use simplified mechanisms that do not reflect modern ransomware threats. Incorporating features from recent ransomware families (e.g., encryption keys per file, delayed activation, or persistence mechanisms) is essential for up-to-date training.

3. Abbreviations and Acronyms

C2 - Command-and-Control
OS - Operating System
VM -Virtual Machine
GUI - Graphical User Interface
HTTP - Hypertext Transfer Protocol

4. Proposed Methodology

4.1 System Architecture

- The system architecture of the Ransomware Simulation project is designed to replicate the behavior of real-world ransomware in a controlled and educational environment. It consists of multiple interconnected modules that simulate the infection lifecycle, including file encryption, ransom note display, simulated communication with a command-and-control (C2) server, and file decryption. The simulation is built using Python and is executed within a virtual machine to ensure safety and containment.
- The architecture follows a modular approach where each component is responsible for a specific function. The flow begins with the encryption of targeted files on the system, followed by the display of a ransom note through a graphical user interface (GUI). The GUI includes a countdown timer to simulate urgency. The user is presented with the option to "pay" the ransom, which triggers communication with a simulated C2 server. Upon receiving the correct decryption key (simulated), the files are decrypted and restored to their original form.

4.2 Tools and Technologies Used

- The Ransomware Simulation project is developed using a combination of programming tools, libraries, and software environments. Each technology plays a vital role in enabling core functionalities such as encryption, user interface creation, file handling, simulated communication, and secure execution. The chosen tools are widely used in cybersecurity education and are suitable for building controlled malware simulations.

4.2.1 Python

- Description: Python is the primary programming language used to develop the entire simulation.
- Reason for Use: It offers extensive libraries, simple syntax, and strong community support, making it ideal for prototyping malware behavior in a controlled environment.

4.2.2 Cryptography Library (Fernet)

- Description: A module from Python's cryptography package that provides symmetric encryption using AES and Fernet.
- Reason for Use: Fernet ensures secure and authenticated encryption and decryption of files. It is easy to implement and sufficient for demonstrating ransomware-like behavior.

4.2.3 Tkinter

- Description: Python's standard library for creating graphical user interfaces (GUI).
- Reason for Use: Tkinter is used to develop the ransom note window, providing buttons, labels, and a countdown timer. It requires no external installation and is suitable for simple GUI applications.

4.2.4 Socket / HTTP

- Description: Built-in Python modules used to simulate network communication between the ransomware and a fake command-and-control (C2) server.
- Reason for Use: These modules demonstrate how ransomware typically communicates with a remote server, but in this simulation, communication is restricted to a local environment for safety.

4.2.5 Virtual Machine Software

- Description: Virtualization tools used to create a sandboxed environment for running the simulation.
- Reason for Use: Ensures safety by isolating the simulation from the host operating system, preventing accidental damage or spread of the simulated ransomware.

4.2.6 OS and File Handling Libraries

- Description: Python's built-in modules for interacting with the file system.
- Reason for Use: Used to navigate directories, scan for specific file types, and perform file-level operations necessary for encryption and decryption.
- 4.2.7 Logging Module Description: Python's logging module for recording runtime events and activities.Reason for Use: Useful for tracking file encryption status, user interactions, and simulated server responses, aiding in debugging and analysis.

4.3 Functional Modules

The Ransomware Simulation project is structured into functional modules, each responsible for a specific stage of the ransomware lifecycle. This modular design promotes clarity, ease of testing, and maintainability. The following sub-sections describe each functional module and its role in the simulation.

4.3.1 File Encryption using Fernet

- This module initiates the core behavior of ransomware by encrypting selected files on the system. It searches predefined directories for target file types (e.g., .txt, .jpg, .pdf) and uses the Fernet symmetric encryption algorithm from Python's cryptography library to encrypt the contents.

Key Features:

- Generates or loads a symmetric encryption key.
- Scans directories and filters target file types.
- Encrypts files in-place and optionally removes the original versions.

Purpose:

- To simulate the disruption caused by real ransomware by rendering user data inaccessible without the decryption key.

4.3.2 Ransom Note GUI with Countdown Timer

Once encryption is completed, the program displays a ransom note through a graphical user interface developed using Tkinter. The GUI includes a message demanding "payment" and a countdown timer to simulate psychological pressure.

Key Features:

- Customizable ransom message text.
- Countdown timer that triggers consequences when time expires (e.g., Deleting the encrypted files).

Purpose:

- To demonstrate the visual and psychological tactics used by ransomware to coerce victims.
- Simulated Command and Control (C2) Communication

4.3.3 Simulated Command and Control (C2) Communication

This module mimics the network interaction between ransomware and an attacker's command-and-control server. It does not use real networking over the internet but instead simulates communication locally.

Key Features:

- Basic socket or HTTP-based client-server architecture.
- Sends the decryption script back to the client (locally) after simulated payment.

Purpose:

- To illustrate how real ransomware may contact external servers for key exchange, updates, or reporting without any actual malicious communication.

4.3.4 File Recovery (Decryption) Process

This module allows encrypted files to be restored to their original state using the correct decryption key. It validates the input key and decrypts each file encrypted during the earlier stage.

Key Features:

- Accepts a key from the user (or simulated C2 server).
- Decrypts files and restores original extensions and contents.
- Handles incorrect key entries gracefully.

Purpose:

- To demonstrate the process of recovery and reinforce the importance of encryption keys in both security and cyberattack contexts.

4.4 Simulation Workflow

The simulation workflow describes the step-by-step process through which the ransomware simulation executes, demonstrating the typical lifecycle of a ransomware attack in a controlled and educational environment. Each stage corresponds to one or more functional modules, collectively illustrating the full ransomware operation from infection to file recovery.

Step 1: Initialization and Target Selection

- The simulation starts by initializing necessary components such as generating or loading the encryption key.
- The encryption module scans predefined directories to identify target files based on their extensions (e.g., .txt, .jpg).
- This step sets the stage for the encryption process, ensuring only specified files are affected.

Step 2: File Encryption

- The encryption module processes each identified file by encrypting its content using the Fernet symmetric encryption algorithm.
- Encrypted files are saved with a modified extension (e.g., .locked), and original files are either overwritten or securely deleted.
- This renders the targeted files inaccessible to simulate the impact of a ransomware attack.
- If the ransom payment is not made within the specified time, the encrypted files are permanently deleted as a consequence.

Step 3: Display of Ransom Note with Countdown Timer

- Upon completion of the encryption, the GUI module launches a ransom note window using Tkinter.
- The ransom note displays a message demanding payment and includes a visible countdown timer to create urgency.

Step 4: Simulated Communication with Command-and-Control Server

- When the user simulates payment by clicking the corresponding button, the ransomware client module initiates communication with a locally hosted fake command-and-control (C2) server.
- The server validates the request and, upon "payment," sends back the decryption to the client.
- This step mimics the real-world ransomware behaviour of contacting an external server for key exchange.

Step 5: File Decryption and Recovery

- Using the decryption key obtained from the simulated C2 server, the decryption module begins restoring encrypted files to their original state.
- Files are decrypted in place, original extensions are restored, and access is regained.
- If an incorrect key is entered, the module handles the error gracefully, maintaining encryption.

Step 6: Simulation End and Logging

- The simulation logs key events such as encryption completion, user interactions, communication success, and file recovery.
- The simulation ends safely within the virtual machine environment, allowing repeated runs without risk to the host system.

5. Environment Setup

A secure and isolated environment is critical for safely developing and testing a ransomware simulation. This section outlines the steps and considerations taken to create a controlled setup where the ransomware simulation can operate without posing any risk to actual user data or systems.

1. Virtual Machine Installation

- Tool Used: Oracle VirtualBox (alternatively, VMware Workstation)
- Guest OS: Windows or Linux (depending on preference)
- Purpose: Provides an isolated sandbox where the simulation can be executed safely.
- Configuration**:**
  - Allocated memory: 2–4 GB RAM
  - Disk space: Minimum 40 GB
  - Networking: Host-only or NAT mode to prevent external access
  - Shared folders disabled to prevent host system interaction

2. Python Environment Setup

- Python Version: Python 3.10 or later
- Installation Method: Official Python installer or package manager (e.g., apt, choco, or pip)

3. Required Libraries and Dependencies

- The following Python libraries were installed within the virtual environment:
  - pip install cryptography
  - pip install tk
- Optional libraries (if using simulated C2 server with HTTP):
  - pip install flask

4. Project Directory Structure

```
Ransomware project/
│
├── countdown.py                ← [VM] Countdown timer module
├── decryption.py               ← [Host] Decryption tool for file recovery
├── encryption_key.txt          ← [Host] Stores the symmetric encryption key
├── Key_Server.py               ← [Host] Simulated C2 server for key exchange
├── ransom_note.py              ← [VM] GUI ransom note with timer
├── ransomware(encrypting).py   ← [VM] Main ransomware logic for file encryption
├── report_server.py            ← [Host] Fake reporting endpoint for victim data
└── victim_reports.txt          ← [Host] Log of victim reports (locally stored)
```

5. Safety Measures

- All test files are dummy or sample documents located in a non-critical directory.

- Host file system is completely isolated from the virtual machine to prevent unintentional damage.
- Networking is kept local-only to simulate C2 communication without real internet access.
- Encryption is reversible using the correct Fernet key to ensure recovery.

6. Testing Protocol

- All tests were conducted inside the virtual machine.
- Encrypted files were verified to ensure they could not be opened without decryption.
- The decryption process was tested multiple times to confirm full file recovery.
- The simulation was reset after each full run by restoring a VM snapshot.

6. Code Explanation

This section explains the full implementation of the ransomware simulation, which includes encryption logic, GUI behaviour, network communication with simulated servers, decryption capabilities, and logging mechanisms. The simulation is divided into two components:

- Virtual Machine (VM) side: Simulates victim behaviour.
- Host Server side: Simulates attacker infrastructure, such as key storage and victim tracking.

6.1 Encryption Logic (ransomware(encrypting).py)

- Purpose: Encrypts user files within the virtual machine using symmetric encryption (Fernet).
- Key Functionalities:
  - Key Generation: A secure key is generated using Fernet.generate_key().
  - File Selection and Encryption: Recursively searches specified directories and encrypts files (e.g., .txt, .pdf) using the Fernet key.
  - Key Transmission: The key is sent to a simulated Command and Control (C2) server (Key_Server.py) using an HTTP GET request.
  - Logging: Logs encrypted files locally for later analysis.
  - File Deletion (on Non-Payment): If the countdown timer expires without payment, the script traverses the encrypted directory and permanently deletes all encrypted files using os.remove().

6.2 GUI Design and Timer Functionality (ransom_note.py and countdown.py)

- Purpose: Displays a ransom note with a countdown timer, mimicking the behaviour of real ransomware.
- Key Functionalities:
  - Graphical User Interface (GUI): Uses Tkinter to create a modal window showing the ransom message.

- Countdown Timer: Implements a ticking timer to simulate urgency using a timer loop in countdown.py.

6.3 Unclosable Window: Prevents users from closing the ransom note manually.

- Simulated Command and Control (C2) Communication

1. Key Storage Server (Key_Server.py)

- Purpose: Receives and stores encryption keys from infected virtual machines.
- HTTP Methods:
  - GET for retrieving keys during encryption & decryption.
- Functionality:
  - Maps keys to IP addresses or machine identifiers.
  - Saves keys in a text file (encryption_key.txt) for recovery.

2. Victim Report Server (report_server.py)

- Purpose: Logs infection reports from virtual machines.
  - Functionality:
  - Receives POST requests with victim hostname, timestamp and IP address of Victim.
  - Stores logs in victim_reports.txt for analysis.

3. HTTP Method Design Note

- POST (store/report): Used for submitting data (e.g., encryption key, victim report).
- GET (key retrieval): Used for retrieving data (e.g., fetching the encryption key).
- This mirrors real-world malware design, where the malware posts stolen info and retrieves commands or keys via GET

6.4 File Recovery (Decryption) Process (decryption.py)

- Purpose: Recovers encrypted files using the correct decryption key retrieved from the server.
- Functionality:
  - Searches for .encrypted files.
  - Uses Fernet to decrypt and restore the original files.

6.5 Logging and Monitoring

- Purpose: Provides transparency for simulation analysis and debugging.
- Log Types:
  - Local logs on the VM: Encrypted file paths.
  - Remote logs on the host:
    - encryption_key.txt: Encryption Key For Decryption

- victim_reports.txt: Hostname, timestamp and IP address of infection.

- Usage: Useful for demonstrating how real attackers track infections and manage victims.

7. Results and Analysis

This section presents the outcomes of the ransomware simulation, evaluates the behaviour of the simulated attack, and discusses its effectiveness and implications for cybersecurity education. The simulation was executed under controlled conditions within a virtual machine, with supporting host-side infrastructure.

7.1 Simulation Demonstration

The ransomware simulation was successfully executed as designed. The following sequence of actions occurred during a typical run:

7.1.1 File Encryption:

- Target files in a designated folder were located and encrypted using a unique Fernet key.
- Encrypted files were renamed with a .encrypted extension.
- Original files were securely deleted after encryption.
- If the “Payment” is not given then it deletes the encrypted files as consequence.

7.1.2 Key Transmission:

- The generated key was sent to the simulated Ransomware in Virtual Machine using an HTTP GET request.
- The key was stored on the server and mapped to the victim’s IP address for later retrieval.

7.1.3 Ransom Note Execution:

- A GUI ransom note appeared on the victim's screen using Tkinter.
- A countdown timer simulated a deadline for payment or decryption.
- The GUI remained persistent and unclosable, replicating real ransomware behaviour.

7.1.4 Victim Reporting:

- The infected VM reported its hostname and infection timestamp to a separate reporting server (report_server.py) via an HTTP POST request.
- The report was logged for analysis

7.1.5 Decryption and Recovery:

- The VM-side decryption script (decryption.py) successfully retrieved the key via an HTTP GET request.
- Encrypted files were decrypted and restored to their original form without data loss.

7.2 Behavior Analysis

Aspect	Observation
Encryption Speed	Small files (<1MB) were encrypted almost instantly; large files (~10MB+) took noticeable time (~0.5s per file).
GUI Persistence	The Tkinter ransom note reliably stayed on top and blocked user attempts to close it manually.
Timer Functionality	Countdown operated as intended, decrementing each second and displaying time pressure.
Key Management	Keys were accurately logged and retrieved using victim IP, ensuring decryption was possible.
Server Stability	Flask-based servers (Key_Server.py, report_server.py) handled requests without crashes during multiple tests.
Reversibility	All encrypted files were successfully recovered using the stored key, validating simulation safety.

7.3 Educational and Research Implications

- The simulation provided valuable insights into how ransomware typically operates, and how:
  - File encryption works at a system level using symmetric cryptography.
  - Attackers manage keys and control decryption access through C2 infrastructure.
  - Victims are pressured through psychological manipulation (countdowns, GUI lock-ins).
  - Security researchers can trace infections via logging and reporting mechanisms.

- This model supports educational use cases in:
  - Cybersecurity training labs
  - Digital forensics practice
  - Threat modeling exercises
  - Demonstrating attack vectors in ethical hacking courses

8. Discussion

8.1 Effectiveness of the Simulation

- The project successfully demonstrates the core mechanics of ransomware in a safe, reproducible manner:
  - It models end-to-end ransomware behaviour — from file encryption to key storage, victim notification, and recovery.
  - The simulation creates a realistic pressure scenario through a graphical ransom note and countdown timer.
  - The use of actual file encryption with Fernet provides authentic hands-on exposure to cryptographic techniques.
  - Integration with a simulated command-and-control infrastructure allows testing of communication channels without real-world risk.
- The simulation fulfils its educational and illustrative goals effectively, providing both technical depth and usability for demonstration.

8.2 Practical Implementation and Feasibility

- The system is modular and easy to deploy:
  - Environment: Runs entirely within a virtual machine, using Python and standard libraries (Flask, Fernet, Tkinter).
  - Minimal Dependencies: Requires no complex frameworks, making it ideal for classroom and lab use.
  - Clear File Separation: VM-side and host-side components are isolated for security and transparency.
  - Customizable: Instructors or researchers can adapt file paths, encryption targets, GUI text, and server behaviour.
- This practical design ensures that the project is feasible for academic labs, cyber drills, and ethical hacking workshops.

8.3 Educational and Research Impact

- The simulation provides tangible benefits in cybersecurity education:
  - Hands-on Cyber Threat Emulation: Gives students a safe way to understand the mechanics of modern malware.
  - Training in Incident Response: Allows practice of decryption, key retrieval, and log analysis.

- Research Platform: Can serve as a foundation for advanced topics like ransomware detection, behaviour analysis, or countermeasure testing.

- By experiencing the inner workings of ransomware, learners can develop critical thinking and analytical skills needed in real-world cybersecurity roles.
- Impact: It bridges the gap between theory and practical threat modeling, enhancing learning outcomes.

8.4 Limitations

Limitation	Description
No Real Attack Vectors	It doesn't simulate phishing emails, drive-by downloads, or remote code execution.
Single-Device Scope	The simulation is confined to a single machine and does not demonstrate ransomware propagation across networks.
Simplified Cryptography	Uses symmetric Fernet encryption, while real ransomware often employs hybrid encryption (AES + RSA).
No Anti-Forensic Behaviour	The ransomware doesn't use obfuscation, privilege escalation, or anti-debugging tactics.
No Financial Simulation	The ransom note is symbolic and does not involve cryptocurrency wallets or payment tracking.

9. Conclusion

9.1 Summary of Findings

- The simulation effectively demonstrates:

  - Real-world ransomware behaviour in a controlled virtual environment.
  - Use of symmetric key encryption (Fernet) to simulate file locking.
  - A persistent GUI-based ransom note with a countdown mechanism to simulate urgency.
  - Command and Control (C2) server simulation, managing key storage and victim reporting.

- Safe decryption and recovery process, ensuring no permanent file loss.
- Logging and traceability, offering transparency and facilitating analysis.

9.2 Contributions

- The project contributes the following to cybersecurity education and research:

  - A modular ransomware simulation framework written in Python.
  - A safe and reversible testbed for teaching cryptographic malware behaviours.
  - Insight into key management, encryption logic, and GUI tactics used by ransomware.
  - A clear separation between attacker and victim components, supporting safe lab deployment.
  - Groundwork for building more advanced simulations or detection experiments.

9.3 Future Scope

- Hybrid Encryption: Integrating asymmetric (e.g., RSA) encryption alongside symmetric encryption to simulate real-world ransomware's key exchange methods.
- Network Propagation Simulation: Extending the simulation to show how ransomware can spread across local networks or shared drives.
- Realistic Delivery Vectors: Simulating infection through phishing emails, malicious attachments, or dropper scripts to represent the full attack lifecycle.
- Behavioural Logging and Detection: Adding advanced logging and system monitoring for use in intrusion detection or machine learning research.
- Visualization Dashboard: Developing a web-based interface for instructors or researchers to monitor simulation behaviour, timers, and reports.
- Single Executable Deployment: Combining all components (encryption logic, ransom GUI, key communication) into one self-contained executable file (e.g., via PyInstaller). This would simulate how real ransomware typically operates as a single binary, while still running safely in a virtual environment.

9.4 Final Remarks

The Ransomware Simulation project demonstrates that even complex cyber threats can be broken down, understood, and safely modeled in a way that is both educational and ethical. By simulating a ransomware attack in a controlled virtual environment, this project empowers learners and researchers to explore malware behavior without risk.

Key takeaways include:

- A hands-on understanding of file encryption, key exchange, and C2 mechanisms.
- Realistic implementation using Python and open-source tools.

- Safe recovery paths that reinforce responsible cybersecurity practices.
- A platform that can be extended for deeper learning, experimentation, and detection research.

10. Acknowledgement

I wish to express my sincere gratitude to my parents, friends and all the members of the family for their precious support and encouragement which they had provided the necessary suggestions and advices along with their valuable co-ordination in completion of my work.

11. Authors' Biography

Mr. Hard Pandya is an undergraduate student specializing in Cybersecurity at Gujarat University, CPC Department. With a keen interest in network security and ethical hacking, Hard Pandya actively engages in research and practical applications in the field of cybersecurity. Passionate about developing secure digital environments, he strives to contribute innovative solutions to modern cyber threats.

Ms. Vedrucha Pandya is an Assistant Professor at Aditya Silver Oak University. With a strong background in cybersecurity and information technology, she enjoys teaching and guiding students in the field. Her goal is to help students develop skills to tackle modern cybersecurity challenges.

References

1. Asmare, A. G., et al. (2021). Covid-Lock Ransomware Simulation in the Age of Pandemics. In Advances in Cybersecurity (pp. 33–49).
2. Bakke, K. O., & Aarnes, H. (2023). Detection and Mitigation of Ransomware with EDR Techniques. University of Agder.
3. Kherwa, R., et al. (2023). Blockchain-based Detection of Ransomware using Machine Learning. IEEE Xplore.
4. Segura, J. (2016, March). Cerber Ransomware: New but Mature. Malwarebytes.
5. Abrams, L. (2023). New RaaS Portal Preparing to Spread Unlock26 Ransomware. BleepingComputer.
6. Adepu, S., & Mathur, A. P. (2019). Understanding Ransomware Behavior. Cybersecurity, 2(6).
7. Munyite, M. (2022). Simple Ransomware Scr ipt in Python. DEV Community.
8. Kuhi, T. (2020). Writing a Ransomware in Python – Practical Demo. Kuhi's Blog.
9. Wababu, P. (2021). How to Make Ransomware with Python (Windows, Mac, and Linux). DEV Community.
10. EasyRead. (2021). Understanding Ransomware: A Simple Demonstration Using Python's Cryptography Library. Medium.
11. OSINT Team. (2021). Writing Your Own Ransomware for Windows in Python. OSINT Team Blog.