INTERNSHIP PROJECT REPORT

Project Report Title : Keylogger with Encrypted Exfiltration

Internship at : Elevate Labs

Submitted by : Spandan Chavan

Duration of Internship : **04 August 2025 - 17 September 2025**

Submitted to : Elevate Labs Project Supervisor

TABLE OF CONTENT

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Elevate Labs** for providing me with the opportunity to undertake this internship. The experience of working on this project independently allowed me to develop my skills, face challenges, and gain practical knowledge in a professional environment.

I am grateful to the entire team at **Elevate Labs** for creating a supportive and resourceful environment that made this learning possible.

INTRODUCTION

In cybersecurity, **keylogging** refers to the practice of recording every key pressed on a computer or device, often for monitoring or auditing purposes. A keylogger can operate as a legitimate tool (for example, for debugging, parental controls, or authorized monitoring) or as a malicious spy tool. By definition, a keylogger “monitors device activity by recording every key that is pressed and then saves that data to a file. This type of program can be used as an effective security measure, as well as a malicious threat if it's in the wrong hands.” When used lawfully and with consent, keylogging can help organizations increase their security posture, such as tracking suspicious activity or debugging software behaviors.

This project explores the development of a stealthy keylogger with encrypted exfiltration to understand how such tools work and to demonstrate secure communication techniques. The client component captures keystrokes using the **pynput** library, encrypts them with a **Fernet (symmetric) session key**, and sends them over a network socket to a server. The server uses **RSA (asymmetric)** to securely receive the Fernet key and then decrypts incoming keystroke messages. **This hybrid design (combining RSA and Fernet)** reflects common practice in secure communications. The goal is to illustrate both the threat posed by keyloggers and the importance of encrypting data in transit.

OBJECTIVE OF THE INTERNSHIP

The primary objectives of this internship and project were:

•**Practical Experience**: Gain hands-on experience in cybersecurity development and network programming.

•**Keylogging & Encryption**: Learn how to capture keystrokes using Python and apply cryptographic techniques (symmetric and asymmetric encryption) to protect data in transit.

•**Stealth and Packaging**: Understand techniques for making a Python application stealthy, including hiding console windows and packaging code into executables.

•**Ethical Hacking Perspective**: Appreciate the ethical and security implications of keyloggers, emphasizing responsible and authorized usage.

•**Skill Development**: Improve skills in Python programming, use of libraries (pynput, cryptography), socket communication, and tools like PyInstaller.

Through these objectives, the internship aimed to bridge theoretical cybersecurity concepts with practical implementation.

PROJECT OVERVIEW

The core goal of the project was to develop a proof-of-concept keylogger that operates covertly on a client machine while securely transmitting logged keystrokes to a server. The project highlights the relevance of encryption and secure communication in the context of cyber threats. Keyloggers themselves are tools often associated with malware; understanding their design helps in developing countermeasures. Moreover, implementing encrypted exfiltration demonstrates how even maliciously captured data can be protected

The key objectives of the project included

:

•**Stealthy Keylogging**: Capture user keystrokes unobtrusively using a global listener without any visible user interface or windows.

•**Encrypted Transmission**: Secure captured data by encrypting it before sending. The project uses Fernet (symmetric encryption) for keystroke data and RSA (asymmetric encryption) for securely exchanging the Fernet key. In practice, this hybrid encryption approach is recommended: encrypt large data with a symmetric cipher (fast) and use RSA for secure key exchange.

•**Networking**: Set up a TCP client-server model where the client sends encrypted data to a listening server over a network socket.

•**Portability and Stealth**: Package the Python client into a standalone executable (using PyInstaller) that runs without a console window, making it appear as a background processes.

By the end of the project, the expected outcome was a working demonstration: the client logs keystrokes, encrypts them, and transmits them to the server, while the server decrypts and records the captured data. All components were developed within an isolated, permissioned environment to emphasize that this is a security exercise and not for unauthorized use.

TECHNICAL IMPLEMENTATION

Keylogging using pynput

The client application uses the pynput library to monitor and record keyboard input. Pynput provides a high-level interface to listen for global keyboard events. In Python, a keyboard listener can be created using pynput.keyboard.Listener and callback functions for on_press and on_release.

```
from pynput import keyboard


def on_press(key):
    try:
        char = key.char  # Alphanumeric key
    except AttributeError:
        char = str(key)  # Special key (e.g., Key.space)
    print(f"{char} pressed")

listener = keyboard.Listener(on_press=on_press)
listener.start()
```

This code snippet (adapted from the official pynput documentation) demonstrates how each key press invokes a callback. The listener runs in a separate thread, allowing the main program to continue execution concurrently. In the project, every captured keystroke (either as an alphanumeric character or a special key representation) is appended to a buffer in memory. Care was taken not to disrupt system performance; the listener simply logs keystrokes and is lightweight.

By using pynput, the keylogger works on multiple operating systems (Windows, Linux, macOS) since the library abstracts OS-specific details. The logged keystrokes are then periodically encrypted and sent to the server (see following sections), rather than being stored in plaintext on disk.

Encrypted Communication (RSA and Fernet)

Once keystrokes are captured, they must be securely transmitted. This project uses a hybrid encryption approach. First, a symmetric key is generated for bulk encryption of keystroke data using the Fernet class from the Python cryptography library. As documented, Fernet provides authenticated symmetric encryption (AES under the hood) and guarantees that any message encrypted by it “cannot be manipulated or read without the key”. The code to set up Fernet is:

```
from cryptography.fernet import Fernet

session_key = Fernet.generate_key()  # Generate a random 32-byte key
cipher_suite = Fernet(session_key)
token = cipher_suite.encrypt(b"keystroke_data")
plaintext = cipher_suite.decrypt(token)
```

Next, the symmetric Fernet key itself needs to be shared securely with the server. For this, the client encrypts the session_key with the server's RSA public key and sends it to the server. The server decrypts it with its private key. For example:

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

ciphertext = public_key.encrypt(
    session_key,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),
                 algorithm=hashes.SHA256(),
                 label=None)
)
plaintext = private_key.decrypt(
    ciphertext,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),
                 algorithm=hashes.SHA256(),
                 label=None)
)
```

This ensures that only the server (holding the private key) can recover the Fernet key. In practice, using RSA to exchange a symmetric key and then using the symmetric key to encrypt data is a recommended strategy. After the key exchange, the client encrypts the keystroke buffer with Fernet, and the server decrypts it, ensuring confidentiality and integrity of the data in transit.

Networking via socket

Python's socket module is used to establish the network connection between the client and server. A socket in Python is an endpoint for sending or receiving data across the network. In this project, the server creates a listening TCP socket on a known port, and the client connects to this server. The typical flow is:

1. **Server**: Create a socket (socket.socket()), bind to an IP and port, call .listen() and accept a client connection (.accept()).

2. **Client**: Create a socket, connect to the server's IP/port (.connect()).

3. **Data Transfer**: Once connected, the client first sends the RSA-encrypted Fernet key. The server receives and decrypts it. Then the client sends Fernet-encrypted keystroke messages in a loop. The server receives each token and decrypts it to recover the plaintext keystrokes.
For example, on the server side:

All data exchanged over the socket is already encrypted, so any intercepted network traffic is unintelligible. This use of sockets for client-server communication follows common practices in Python networking.

Packaging with PyInstaller

To achieve stealth and portability, the Python client script was packaged into a standalone executable using PyInstaller. PyInstaller analyzes a Python script, finds all its dependencies (including the interpreter), and bundles them into a single folder or executable. For the client, the **--onefile** option was used to create one executable file, and --windowed (or **--noconsole**) was used to suppress the console window.

For example:

>>>pyinstaller --onefile --windowed keylogger.py

This command tells PyInstaller to process keylogger.py with the onefile and windowed options, resulting in a single EXE. The documentation explains that with --windowed (alias --noconsole), no console window appears on Windows or macOS. The resulting executable contains the Python interpreter and the script bytecode, and can run on the target machine without a separate Python installation. This packaging allows the keylogger to run quietly in the background, fulfilling the "**stealth**" requirement.

After packaging, running the compiled keylogger.exe showed no visible console, as expected. Thus, PyInstaller effectively produced a stealthy, self-contained executable for the keylogger.

TOOLS AND TECHNOLOGIES USED

• **Python 3**: The development language for both client and server.

• **pynput**: Python library to capture global keyboard events (pynput documentation[3]).

• **cryptography**: Python library providing Fernet (symmetric AES) and RSA primitives.

•**socket**: Python's built-in networking library for TCP client-server communication.

• **PyInstaller**: Tool to bundle Python scripts into executables.

• **Operating System**: Any Operating System is suitable. However Linux is more preferable where the code was developed and tested.

• **IDE/Editor**: Any code code editor of user's choice for coding and debugging.

• **Other Libraries**: logging module for diagnostics, threading for concurrent execution, etc.

These tools and libraries were selected to implement keylogging, encryption, networking, and packaging as required by the project.

SECURITY AND ETHICAL CONCERNS

Throughout the project, ethical guidelines and security best practices were strictly followed. The keylogger and server were implemented and tested only in a controlled environment on machines where we had full authorization. Explicit disclaimer: This project and its code were developed solely for educational and research purposes. Unauthorized deployment of a keylogger on any system is illegal and unethical.

Keylogging inherently involves privacy concerns. In legitimate settings, keylogging can only be used when transparency and consent are ensured. Users should be informed if their keystrokes are monitored to avoid legal issues. For example, organizations must tell employees about monitoring to avoid legal responsibility. In our project, no real user data was captured without consent; only sample inputs were logged. All code includes documentation and disclaimers (e.g., “for educational use only”). Industry references emphasize that keyloggers “should be used correctly” and transparently in authorized settings, otherwise they are considered malicious spyware.

In summary, the ethical measures included: - Ensuring all testing was authorized by the host organization.

- Including clear disclaimers about educational use.
- Discussing privacy considerations and emphasizing transparency.
- Emphasizing responsible usage only in authorized contexts.

RESULTS AND OUTCOMES

The project successfully demonstrated a working prototype of the stealthy keylogger with encrypted exfiltration. The client reliably captured keystrokes; the buffer was then encrypted with Fernet and transmitted to the server. The server decrypted the messages and recorded the keystrokes. In testing, even if the network traffic was intercepted, the keystrokes remained confidential (the ciphertext did not reveal any information). This confirmed that the encryption steps (Fernet + RSA) effectively protected the data.

Key outcomes included: - The Fernet encryption ensured that any tampering with the ciphertext would be detected; only the correct key on the server could recover the plaintext.

- The RSA key exchange successfully allowed the server to receive the session key securely.

- The PyInstaller executable ran without a console window, meeting the stealth requirement.

- Overall, all core project goals were achieved: covert keylogging, secure transmission, and correct decryption on the server side.

These results show that the system operates as intended, fulfilling the objectives of capturing, encrypting, sending, and decrypting keystroke data in a hidden manner.

CONCLUSION

This internship project provided a comprehensive experience in developing a simple yet instructive cybersecurity tool. By building a stealthy keylogger with encrypted exfiltration, I gained practical insight into cryptography, network programming, and software deployment. The project demonstrated how symmetric (Fernet) and asymmetric (RSA) encryption can work together to secure data, and how tools like PyInstaller can make Python programs portable and stealthy.

Overall, the project met its objectives. The final system successfully captured and transmitted keystrokes securely, and key learnings were applied in each step. This endeavor also highlighted the broader lesson that powerful tools must be used ethically. Future work could involve enhancing the program (e.g., adding multiplexing or more secure key management) and exploring defenses against such keyloggers. The knowledge acquired here builds a foundation for further exploration in cybersecurity and ethical hacking.

References

- Pynput keyboard monitoring documentation[3].
- Cryptography library documentation (Fernet symmetric encryption)[4].
- Cryptography library documentation (RSA public-key encryption)[8].
- Nepali, S., “Complete Guide to Encryption and Decryption in Python (For Beginners)” (Medium, 2025)[2][12].
- Jennings, N., “Socket Programming in Python” (Real Python, 2024)[5].
- PyInstaller documentation, “What PyInstaller Does and How It Does It”[6][9], and usage options[7].
- University of Phoenix, “What is keylogging?” (Phoenix.edu, 2023)[1][11].
- TimeChamp blog, “Keylogger Software & Legal Implications” (2024)[10].

---

[1] [11] What is Keylogging? | University of Phoenix

https://www.phoenix.edu/articles/it/what-is-keylogging.html

[2] [12] Complete Guide to Encryption and Decryption in Python (For Beginners) | by Sunil Nepali | Jun, 2025 | Medium

https://medium.com/@sunilnepali844/complete-guide-to-encryption-and-decryption-in-python-for-beginners-61c2343c3f2b

[3] Handling the keyboard — pynput 1.7.6 documentation

https://pynput.readthedocs.io/en/latest/keyboard.html

[4] Fernet (symmetric encryption) — Cryptography 46.0.0.dev1 documentation

https://cryptography.io/en/latest/fernet/

[5] Socket Programming in Python (Guide) – Real Python

https://realpython.com/python-sockets/

[6] [9] What PyInstaller Does and How It Does It — PyInstaller 6.15.0 documentation

https://pyinstaller.org/en/stable/operating-mode.html

[7] Using PyInstaller — PyInstaller 6.15.0 documentation

https://pyinstaller.org/en/latest/usage.html

[8] RSA — Cryptography 46.0.0.dev1 documentation

https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/