Tunis business school

TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Keylogger software

IT 360: Information Assurance and Security

Authors :

Yesmine Srairi

Yassine Ben Omrane

Rahma Ellouze

Yasmine Akik

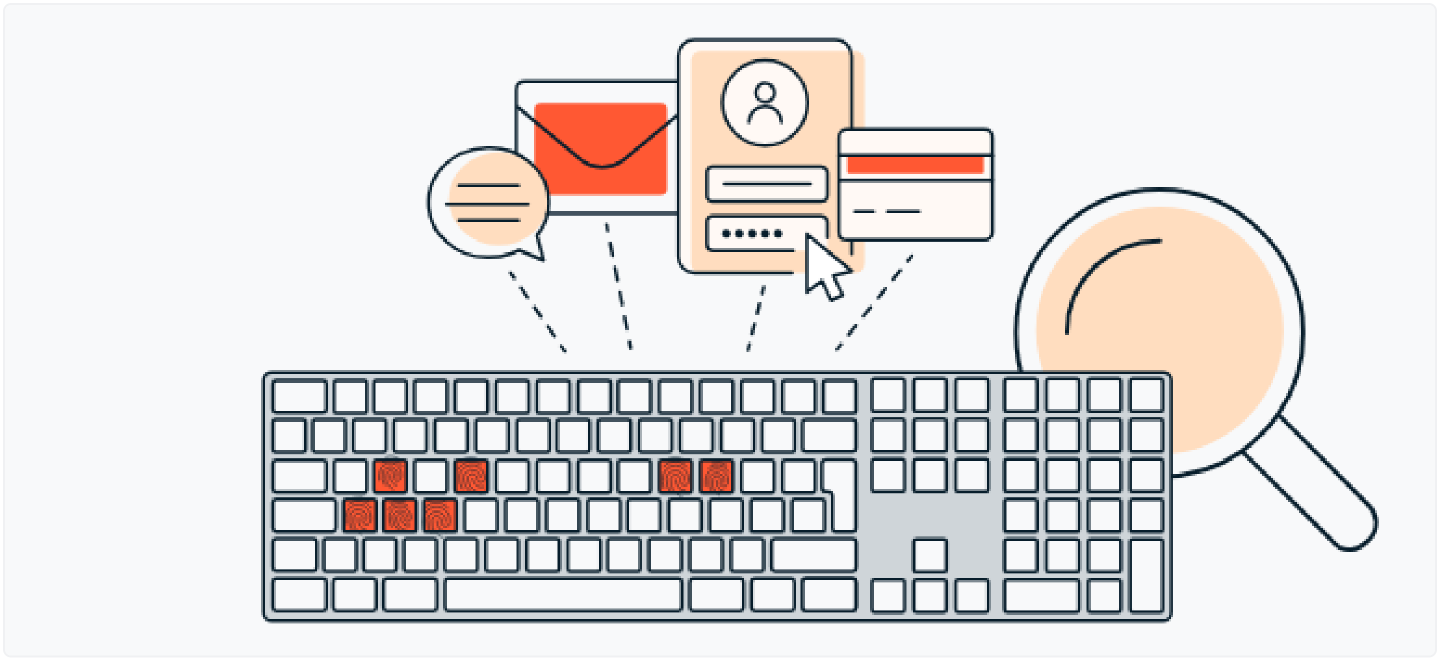
Submitted to :

Dr.Manel Abdelkader

Glossary

Keylogger Software Project Report

I- Introduction:



1- What is a keylogger software ?

A keylogger is a type of software designed to record every keystroke made by a user on their keyboard. It functions by running in the background and monitoring user input, typically saving the captured data in a log file for later analysis.

While keyloggers are often associated with malicious activity such as being part of ransomware or spyware attacks used by hackers to steal sensitive information like passwords and credit card detail they also have legitimate and ethical uses. For example, companies may use keylogging tools to monitor employee productivity, ensure compliance with security policies, or detect insider threats. In cybersecurity, ethical hackers and analysts might use keyloggers in controlled environments to test a system’s resilience against input-based threats.

Thus, the intention and context of use determine whether a keylogger is harmful or helpful.

2- Project objectives and scope:

In this project, we will develop a basic keylogger software that captures and records keystrokes made on a keyboard. The goal is to understand the fundamental mechanisms behind input monitoring tools, and to explore how such software can be implemented using high-level programming languages like Python. Our keylogger will run in the background, log each key pressed by the user, and store the data in a local file for review.

> ***Note:* Our version will remain simple and ethical, limited to local logging for learning purposes only**

II- Main components:

1- Keyboard Event Listener :

This part of the software is responsible for detecting when a key is pressed on the keyboard. It works by constantly "listening" for keyboard activity. When you press a key, the event listener picks it up and passes it to the rest of the program. In programming languages like Python, there are libraries such as `pynput` or `keyboard` that make this easy to implement. The listener stays active in the background and runs until the program is stopped.

2- Logging System :

Once the event listener detects a key press, the logging system saves that information somewhere — usually in a file on the computer. For example, it might write each key that was pressed into a `.txt` file, one after another. This way, all the keys you press are recorded in one place. The system can also add useful details like the date and time of each keystroke to make the data more organized.

3- Data Management:

This part handles how the data is processed after being captured. It can clean up the raw data to make it easier to read, like changing symbols into readable words (turning a space bar press into “[SPACE]”). It can also be used to remove duplicate entries, group the keystrokes by session, or even encrypt the data so that it’s protected from being seen by others. This step helps keep the log organized and secure.

4- User Interface:

Some versions of keylogger software include a simple interface that lets you adjust how the program works. For example, you might be able to choose where to save the logs or turn the keylogger on and off with a button. This is not required for basic versions of the program, but it can make it easier to use if you're managing it on your own computer.

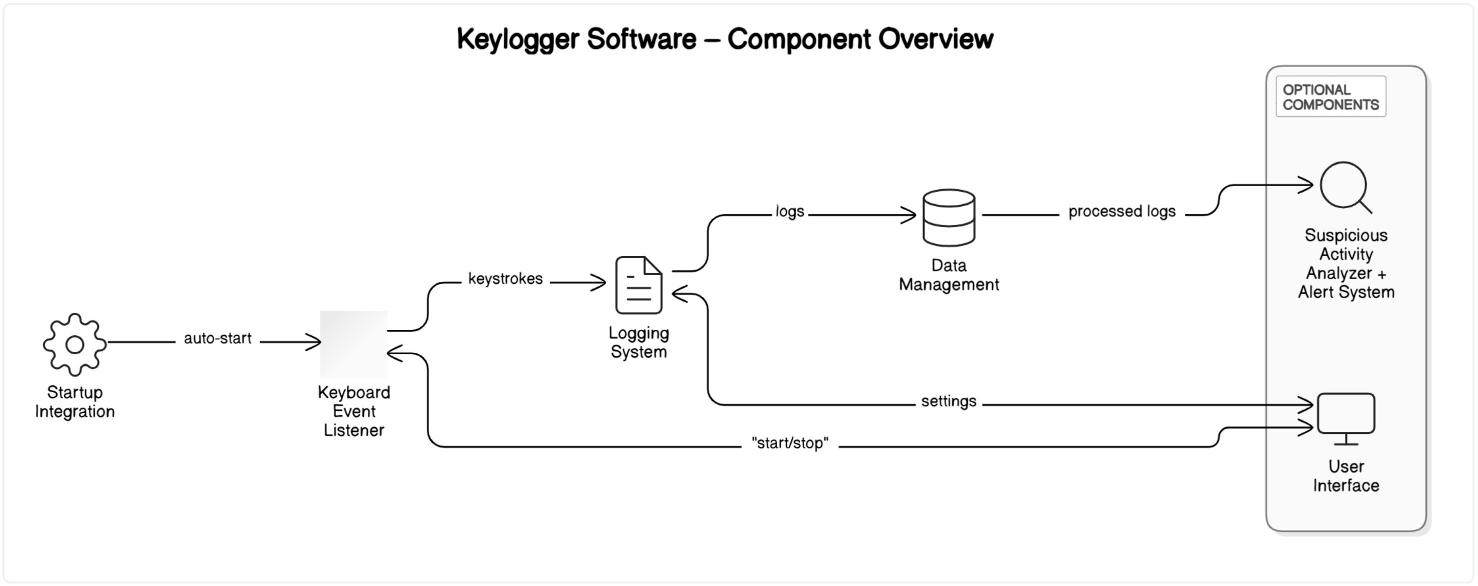
5- Startup Integration:

This is a feature that allows the keylogger to start running automatically every time the computer is turned on. It works by adding the program to the system's list of startup applications. Once this is done, you don't need to open the program manually each time — it runs in the background from the beginning. This is more common in advanced setups, especially when long-term tracking is needed.

6- Suspicious Activity Analyzer with Alert System

This part of the software is in charge of going through the logged keystrokes and checking if anything unusual is happening. To do this, we can use either simple keyword-based rules or a basic machine learning model trained to detect suspicious writing patterns.

III - Functional flow:



1-Start the Program

The user (or the system) runs the keylogger script. At this point, the program gets everything ready in the background.

2- Initialize the Keyboard Listener

The keyboard event listener is activated. This is the part of the code that waits and listens for any key pressed by the user. It also loads any configuration settings, like where the log file should be saved or how special keys should be formatted.

3- Capture Each Keystroke

Whenever a key is pressed, the listener immediately detects it. The event gets passed to a small function that takes care of the rest of the processing.

4- Process the Keystroke :

The key that was pressed is turned into a readable format. For example, if you press the spacebar or Enter, the program might translate that into “[SPACE]” or “[ENTER]” so that the logs make sense when read later. It also makes sure symbols, numbers, and letters are clearly logged.

5- Log the Keystroke :

The processed key is written into a text file, usually by appending it at the end of the file. This log file keeps growing as more keys are pressed. Optionally, the software can also add the **timestamp** (date and time) next to each entry to show when it happened.

6- Format or Encrypt the Data :

If needed, the keystroke data can be cleaned, formatted for readability, or even encrypted to make it more secure. This is useful for privacy and security, especially if the logs are stored for later analysis.

7- Analyze the Keystrokes :

At this stage, the program can use a built-in analyzer (a rule-based system or a machine learning model) to check if the logged words or patterns look suspicious. If anything is detected, the program can raise an alert like sending an email, logging the event in a separate alert file, or showing a warning message.

8- Keep Running in the Background :

The keylogger continues listening for new keystrokes and repeating steps 3–7 until it is manually stopped or the computer is shut down. The entire process is invisible to the user unless intentionally made visible.

IV - Theoretical background

This part serves as the foundational framework for understanding the key concepts, technologies, and scientific principles behind the development of a keylogger system. Including this section in our report ensures that the practical implementation is not approached as a "black box," but rather as a well-informed, technically grounded solution.

1- System-Level Fundamentals:

a) Keyboard Input Events

Every time a key is pressed, it generates an interrupt signal, which is handled by the operating system. This signal is queued and eventually interpreted by the active application. To capture this signal before the application does, a program must hook into the event stream.

b) Hooking Mechanism

Hooking is a method of intercepting system-level events, such as keyboard input, before they reach their destination.

Types of Hooks:

- *Local hooks*: limited to one application
- *Global hooks*: intercept all system-wide input
- *Low-level hooks*: operate at the OS level

In our project, we will be using global hooks is used to monitor all keystrokes entered by the user.

> *You can find more Implementation details related to the keyboard event listener covered in the technical implementation part*

2. Programming Concepts Behind a Keylogger:

To log keystrokes effectively, the following core concepts are utilized:

a) Event Listeners & Callbacks:

Functions that react to `KeyDown` and `KeyUp` events. In Python, the `pynput` library is used to attach listeners to the system.

b) Data Logging:

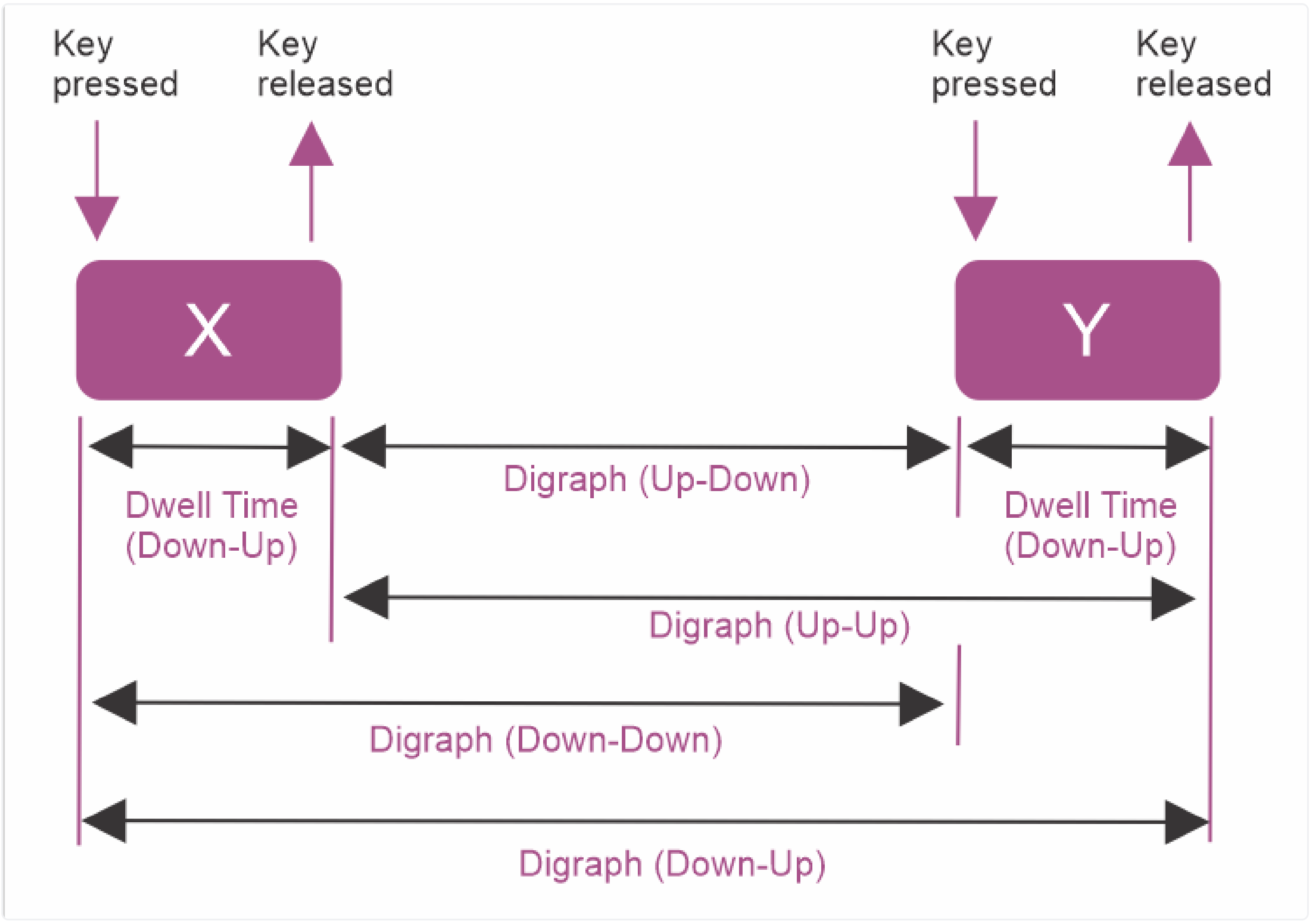
Keystrokes are stored locally in a log file, possibly with time stamps and formatting.

c) File Handling & Persistence:

Ensuring logs are stored securely and reliably.

> *Full details on the implementation are described in the "Technical Implementation" section.*

3. Keystroke Dynamics and Behavioral Profiling:



Beyond logging raw keys, we can analyze how people type. This is known as **keystroke dynamics**, a subset of **behavioral biometrics** which is is a method of identifying or authenticating individuals based on unique patterns in how they interact with devices or systems rather than physical traits.

a) Key Metrics in Keystroke Dynamics:

Dwell Time:

- The amount of time a key is held down from key press to key release.

=> This metric matters since each user tends to have a unique "touch" when typing. Some type aggressively, holding keys longer; others have a lighter, quicker touch.

Typing patterns :

The overall rhythm, speed, common sequences and consistency of typing.

=>This metric matters since everyone has a natural "typing signature". Monitoring changes helps in detecting:

- Unauthorized access
- Stress/fatigue (in high-security contexts)
- Bots/scripts mimicking input

Purpose:

- Sudden changes in typing speed or rhythm.
- Unusual commands or word patterns (e.g., typing `sudo rm -rf /`).
- Multiple users using the same credentials but with different typing behaviors.

Coupled with an alert system, it can notify system admins or trigger defensive protocols.

4. Machine Learning in Keystroke Analysis :

a)Natural Language Processing :

In order to detect suspicious activities based on the content of the typed keystrokes, it is essential to process and understand the raw text generated by the user. This is the role of Natural Language Processing .

NLP provides a set of techniques to clean, transform, and extract meaningful patterns from human language data. In the context of keylogger-based suspicious activity detection, we apply basic NLP steps such as:

- **Tokenization:** Breaking down the captured keystroke stream into individual words or tokens.
- **Stop Word Removal:** Eliminating common words (like "the", "and", "a") that do not carry meaningful information.
- **TF-IDF :** Measuring how important a word is relative to a collection of texts. This helps identify unusual or significant terms.
- **Feature Extraction:** Transforming the processed text into a numerical representation suitable for machine learning models.

Once the text is processed through NLP, it becomes structured input for classification algorithms,

This part leverages key concepts from our Data Mining course, such as feature extraction, classification algorithms, and anomaly detection techniques, to identify and flag suspicious activity based on both typing behavior and textual content.

we wish to go beyond recording and into analyzing keystrokes ,That's why machine learning is essential.

we will be using ML to analyze the content of typed keystrokes in order to detect:

- **What** the user types content-based analysis)
- **How** they type (behavioral biometric patterns)

b) Feature Engineering:

b.1. Behavioral Biometrics Features (How You Type):

These features capture the **physical interaction** with the keyboard.

Feature	Description
Dwell Time	How long a key is held down (KeyDown → KeyUp)
Typing Speed	Characters typed per second
Sequence Rhythm	Pattern in timing between key pairs or trigrams

These features are numerical and help identify abnormal typing rhythms such as bots or impersonators.

b.2. Content Based Features (What You Type):

These are derived from the actual keystroke**s** to detect suspicious intent.

Feature	Description
TF-IDF Vectors	Highlight rare but important terms like “drop table” or “sudo”
N-grams (1–3)	Capture word/command sequences such as `rm -rf`, `password`, `SELECT *`
Keyword Flags	Presence of specific known-risk terms (e.g., “hack”, “exploit”, “admin credentials”)
Text Length & Entropy	Unusually short/long text or random strings (e.g., encoded payloads)

These features are **textual**, used to model intent and possible malicious actions.

c) Model Choice : Naive Bayes:

Naive Bayes is a probabilistic classifier well-suited for text classification, especially when:

- The input space is large and sparse (e.g., many n-grams, TF-IDF values)
- Interpretability and speed are priorities
- There is a combination of categorical and continuous features (like keyword flags + typing time)

c.1. How It Works:

Naive Bayes uses Bayes' Theorem to compute the probability that a given input belongs to a specific class

$$P(Class|Features) = \frac{P(Features|Class) \cdot P(Class)}{P(Features)}$$

It assumes feature independence, which is acceptable for high-dimensional keystroke/text data and simplifies computation.

c.2. Work Flow:

1. Data Collection

- Raw keystrokes, with timestamps for behavioral metrics
- Typed phrases, segmented and labeled as benign/suspicious

2. Preprocessing

- Normalize timing features (e.g., scale between 0 and 1)
- Convert text to TF-IDF (Term Frequency – Inverse Document Frequency). or Bag-of-Words

  => we can compute the TF-IDF as follows

$$TF(t) = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}}$$

$$IDF(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents with term t}}\right)$$

TF-IDF score = TF x IDF

- Tokenize suspicious command patterns using n-grams

3. Training Phase

- Train a **Naive Bayes classifier** on labeled examples
  - Input = [dwell time, flight time, TF-IDF vector, keyword flags]
  - Output = `benign` or `suspicious`

4. Real-Time Inference

- As the user types:
  - Behavioral features are captured dynamically
  - Content is evaluated for known-risk terms and statistical anomalies
- If probability of `suspicious` > threshold → Raise alert

V - Technical implementation:

1. Keystroke Detection:

a)Pynput Library :

The Python library allows you to control and monitor input devices. It contains sub packages for input devices supported: a mouse or trackpad and keyboard.

to import mouse controlling classes : **pynput.mouse**

to import keyboard controlling classes : **pynput.keyboard**

 =>All modules are automatically imported into the pynput package. To use any of them, import them from the main package

pynput.keyboard Module :

```
from pynput import  keyboard
```

Listener module :

Acts as the component responsible for continuously monitoring and capturing every keystroke made by the user. It runs in the background and logs each keypress as it occurs, enabling real-time recording of keyboard activity.

```
from pynput.keyboard import Listener
```

Key module :

Provides a structured way to identify and handle special keys such as shift , enter , backspace ,etc ...

It distinguishes between regular character inputs and functional keys, ensuring that complex key combinations

```
from pynput.keyboard import key
```

> ***==> Together, they allow for the effective and precise logging of all keyboard interactions `pynput` is a cross-platform library, but its compatibility varies slightly across operating systems***

2.Logging Strategy:

The logging strategy is a crucial part of the keylogger software. It ensures that every captured keystroke is stored accurately, securely, and in a structured format for future analysis or sending.

a)Strategy :

Step	Description
1. Capture	Detect keypress events in real-time using different techniques
2. Format	Attach a timestamp and format each log entry consistently.
3. Store	Save log entries to a local file, either in plaintext or encrypted format.

a.1. Keystroke Capture:

The Detection of keypress events in real-time can be fulfilled through the **pynput** library since it provides an event-driven, lightweight, and efficient keystroke capture solution,

Captured using the `on_press()` function with the `pynput.keyboard` module (python)

```
    from pynput.keyboard import Listener
    def on_press(key):
        try:
            print(f"Key {key.char} pressed")
        except AttributeError:
            print(f"Special key {key} pressed")
    with Listener(on_press=on_press) as listener:
        listener.join()
```

a.2. Timestamp Formatting :

Timestamp formatting is the process of adding the exact date and time to each keystroke recorded. It provides significant temporal context that makes logs more understandable, traceable, and meaningful.

Python datetime module:

The Python `datetime` module is used to timestamp each captured keystroke, helping track when sensitive actions occur. Timestamps make the logs more valuable for incident reporting, employee monitoring, and spotting suspicious activity patterns like late-night access.

```
    from datetime import datetime
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

a.3.Storage:

In our project, the storage strategy defines where, how, and in what format the captured keystrokes will be saved.

For our project, we chose local text file logging as the basic method for saving keystrokes. This ensures that data remains available even after the program is closed or the system restarts. We use Python's built-in `open()` function in append mode ("a") to continuously add new keystrokes to a file named logs.txt.

```
with open("logs.txt", "a") as f:
    f.write(f"{entry}¥n")
```

3.Technical Components of Suspicious Activity Detection:

In this section, we outlined the key components involved in detecting suspicious activity within the keylogging system. Rather than relying on predefined trigger words, we leverage Natural Language Processing (NLP) techniques to analyze and classify keystroke patterns in real-time.

a) Keystroke Buffering:

Keystrokes are not analyzed one-by-one in isolation, A **temporary buffer** stores recent characters typed. This buffer is used to detect complete words or sequences like `”password”` or `”bank”`.

b)Natural Language Processing:

Rather than relying on a predefined list of trigger words, NLP will be employed to analyze the typed content. The keystrokes will be processed using NLP techniques such as tokenization, TF-IDF (Term Frequency-Inverse Document Frequency), and contextual classification *(you can find further information on this topic in the theoretical background section)* to identify suspicious activities or sensitive terms based on the context of the conversation.

the following libraries will be used in order to process the keystrokes:

1.NLTK (Natural Language Toolkit)

For basic NLP tasks like tokenization and text classification. `pip install nltk`

2.scikit-learn

For machine learning models and text vectorization

c) Detection Mechanism

The buffer is continuously analyzed by an NLP model, which evaluates the context of the typed content. If the content is classified as suspicious or abnormal, an alert is triggered

```
#the implementation can be fulfilled through this code
def on_press(key):
    global buffer
    try:
        buffer += key.char
    except AttributeError:
        buffer += ” ”



    if nlp_model.predict(buffer):
        print(f”⚠️ Suspicious activity detected: {buffer}”)
        log_alert(buffer)
        buffer = ””
```

d) Logging Behavior Based on Triggers:

Once a trigger is detected, , the system logs the event using a secure storage strategy:

- Log the event into a separate alert file: `alert_logs.txt`
- Encrypt the keystrokes for sensitive data.
- Send a notification (system, email, or server callback). Add a flag or label to highlight the eventù

- Send a notification (system, email, or server callback).

```
with open("alert_logs.txt", "a") as f:
    f.write(f"[ALERT] {datetime.now()} → {buffer}¥n")
```

d) Optional Components:

d.1. Screen Logging Module:

Screen logging captures the current visual state of the user's screen at specific intervals or when triggered by sensitive actions for example when a suspicious keyword is typed. This gives valuable context to the keystroke especially useful in forensic analysis.

Library Used:

`PIL.ImageGrab` or `pyautogui.screenshot()` in Python

```
from PIL import ImageGrab
from datetime import datetime
import os

def capture_screenshot():
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"screens/screenshot_{timestamp}.png"
    screenshot = ImageGrab.grab()
    screenshot.save(filename)
```

Triggering events :

- At fixed intervals
- On detection of trigger keywords
- On special key combinations

Storage Strategy:

Screenshots are stored in a dedicated folder (`/screens`) and named with timestamps. Optionally, the files can be:

- Compressed to reduce size
- Encrypted for confidentiality

2. Clipboard Logging Module :

is the act of recording and/or capturing what users cut or copy (Ctrl+C, Ctrl+X) to the system clipboard.

The clipboard often contains **sensitive data** like passwords, credit card numbers, email addresses, and confidential text, making it a **critical target** for monitoring in advanced keylogging software.

In clipboard monitoring, polling is commonly used to periodically check the clipboard's content for any changes

Library Used:

`pyperclip` or `tkinter`

The logger continuously monitors the clipboard for changes.

```
import pyperclip
import time

last_data = ""def monitor_clipboard():
    global last_data
    while True:
        data = pyperclip.paste()
        if data != last_data:
            print(f"[CLIPBOARD] {data}")
            with open("clipboard_logs.txt", "a") as f:
                f.write(f"[{datetime.now()}] {data}¥n")
            last_data = data
        time.sleep(1)
```

4. Supported Operating Systems

Windows :

For the initial version of our keylogger software, **Windows** will be the only supported operating system.

The program will rely on native **Windows APIs** for both **keyboard** and **mouse** monitoring and control, ensuring full compatibility and efficient performance.

VI - Advantages and limits of existing solutions

1.Advantages of Existing Keylogger Solutions :

Comprehensive Monitoring:

Most keyloggers can capture all types of input : passwords, chats, emails, documents offering complete visibility over user activity.

Stealth Operation:

Advanced keyloggers are highly discreet : they run invisibly without impacting system performance, making them hard to detect ;which is useful for both security testing and malicious purposes.

Simple Deployment:

Many keyloggers are lightweight and easy to install, sometimes requiring only a few minutes to set up on a target machine.

Data Collection for Security Audits:

In ethical contexts, companies can gather important data to analyze employee behavior, detect policy violations, or investigate breaches.

Support for Remote Access:

Some keyloggers can send captured data remotely via email, FTP, or cloud dashboards, making monitoring scalable.

Customizable Features:

Modern solutions allow filtering (for example: log only specific applications, exclude safe apps) and periodic screenshots, offering richer data beyond keystrokes.

2.Limitations of Existing Keylogger Solutions :

Detection by Antivirus Software:

Most updated antivirus and endpoint protection solutions can now detect even well-hidden keyloggers, limiting their stealth effectiveness.

Legal and Ethical Risks:

Deploying a keylogger without explicit consent can lead to serious legal consequences

False Positives:

Security software sometimes incorrectly flags legitimate monitoring tools as malware, complicating deployment for ethical purposes.

Encryption Challenges:

Some keyloggers fail to capture data entered into encrypted fields (like passwords entered in secure browsers), reducing their effectiveness.

High Volume of Data:

Capturing everything can lead to overwhelming amounts of irrelevant data, making analysis tedious and resource-consuming.

Resistance Mechanisms:

Modern operating systems, browsers, and software employ countermeasures like sandboxing, secure input fields, or virtual keyboards to block or confuse keyloggers.

System Instability:

Poorly designed or outdated keyloggers can cause software crashes, system lags, or even expose the monitored system to other security risks.

VII- Cybersecurity & Ethical Considerations:

The development and use of keyloggers raise significant ethical and legal concerns. While they can be used for legitimate purpose such as employee monitoring, parental controls, or security auditing they also have the potential to be misused for malicious intent, such as identity theft, data breaches, or unauthorized surveillance.

a) Key Ethical Principles:

a.1.Informed Consent:

Users should be made aware when their keystrokes are being recorded. Transparent policies are critical in any ethical implementation.

a.2.Purpose Limitation:

Keylogger use should be confined strictly to its declared objective, such as detecting suspicious behavior within an organization.

a.3.Data Privacy:

Logged data must be stored securely, anonymized where possible, and protected from unauthorized access.

b.4. Compliance with Law:

All implementations must follow cybersecurity laws and regulations (e.g., GDPR, HIPAA, or local data protection laws).

b) Responsible Usage

In this project, the keylogger is studied and developed in an educational context for learning and security research purposes. Any application outside this scope must be clearly documented, ethically justified, and legally compliant.

VIII-Resources & References:

a) Academic & Technical Papers

- Ahmed, M., Mahmood, A. N., & Hu, J. (2016). *A survey of network anomaly detection techniques*. Journal of Network and Computer Applications, 60, 19–31. https://doi.org/10.1016/j.jnca.2015.11.016
- Teh, P. S., Teoh, A. B. J., & Yue, S. (2013). *A survey of keystroke dynamics biometrics*. https://onlinelibrary.wiley.com/doi/10.1155/2013/408280

b) Machine Learning & Data Mining:

- Scikit-learn: Naive Bayes Classifiers
  https://scikit-learn.org/stable/modules/naive_bayes.html

c) Programming Libraries And Tools:

- Pynput Documentation:
-

d) Cybersecurity And Ethical Considerations:

- SANS Institute:
- https://www.sans.org
- OWASP Foundation (Security Standards):

https://owasp.org