

# Tunis business school



## Keylogger software

### IT 360: Information Assurance and Security

#### **Authors :**

Yesmine Srairi

Yassine Ben Omrane

Rahma Ellouze

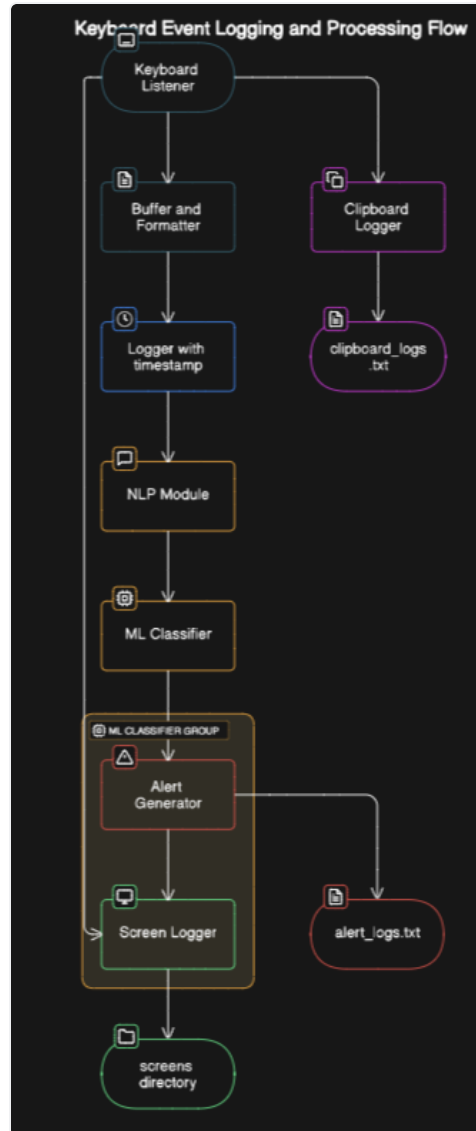
Yasmine Akik

#### **Submitted to :**

Dr.Manel Abdelkader

# Report

## 1. System Design Overview :



### 1. Keyboard Listener :

The Keyboard Listener is responsible for intercepting and recording every key pressed by the user, regardless of the appli

This is the initial point of the system pipeline. It runs as a background service from the moment the software is started.

Its primary role is to detect and capture all keyboard events in real time. It ensures that every keystroke, including special keys (like Enter, Backspace, Shift), is collected and sent downstream to

be processed. By operating at a low level in the system, it guarantees continuous monitoring without user awareness.

## **2. Buffer and Formatter :**

This module stores keystrokes temporarily in memory and applies basic structuring before the data is logged or analyzed further.

It immediately receives raw data from the keyboard listener for temporary storage and pre-processing.

The Buffer's role is to ensure that the system doesn't log individual characters one by one, which can be hard to analyze. Instead, it accumulates text over a time window or until a logical stopping point (like pressing Enter). The Formatter processes the buffer content by interpreting special keys (turning multiple Backspaces into character deletion), segmenting text into lines or sessions, and ensuring formatting consistency for logs and classifiers.

## **3. Logger with Timestamp :**

This module takes the cleaned, structured input from the formatter and writes it to a log file, attaching the exact time each input was made.

It is the first module to persistently store structured keystroke data.

The Logger's timestamping feature is essential for tracing user behavior chronologically. Whether for compliance, auditing, or behavior reconstruction, having a time-labeled sequence of input events provides valuable insights. It allows investigators to determine when a password was entered, what time a website was accessed, or when a user may have copied sensitive information.

## **4. NLP (Natural Language Processing) Module :**

It serves as a bridge between basic text logging and intelligent analysis.

The NLP Module applies linguistic and pattern analysis techniques to understand the context and meaning of the user's input.

It analyzes buffered text using techniques like tokenization, named entity recognition, and keyword detection. This helps extract semantic patterns such as the presence of names, dates, email addresses, or financial information. It might also detect intent, like if the user is filling out a form, searching for a bank login, or typing an email address. This extracted information is used as input features for the machine learning classifier.

## **5. Naïve Bayes Classifier (ML Classifier) :**

It receives the output of the NLP module for intelligent classification.

The classifier is a probabilistic machine learning model that applies Bayes' theorem to classify input as "normal" or "suspicious," assuming the independence of features.

Based on training data, the Naïve Bayes classifier calculates the probability that a particular text belongs to one of two categories: safe for normal user behavior or risky for potential threat or policy violation . It uses features such as the presence of sensitive keywords, typing patterns, or context tags from the NLP module. This module makes the keylogger more intelligent, allowing it to focus only on important or risky events, rather than flooding logs with irrelevant data.

## 6. Alert Generator :

This module is activated when the classifier predicts a suspicious event.

The Alert Generator is a reactive submodule that creates and logs alerts based on the classifier's output.

Once the classifier identifies a risky event, the Alert Generator logs a structured alert entry. This may include a timestamp, risk type ( "possible password entry"), and the snippet of text or behavior that triggered it. Additionally, it may trigger secondary surveillance actions like taking a screenshot or flagging the session for review.

## 7. Screen Logger :

Triggered by the Alert Generator upon suspicious classification.

This module captures an image of the user's screen at the moment an alert is generated.

Screenshots provide valuable visual context that text logs alone cannot capture. For instance, while a typed password is logged, the screenshot may show that it was typed into a banking website or a password manager. This makes forensic investigation more thorough and supports evidence-based decisions. Captured images are stored for later analysis in a dedicated directory.

## 8. alert\_logs.txt :

It is the main storage file for high-risk event records.

A dedicated plain-text log file where all alerts generated by the Alert Generator are written.

This file acts as a focused summary of all flagged events, separate from the full keystroke logs. It allows the administrator to quickly identify and review moments of concern without sifting through all user activity. Each alert contains contextual data such as the triggering text, timestamp, and possibly even the name of the active window.

## 9. Clipboard Logger :

Runs in parallel to the keystroke pipeline and monitors a different vector of input.

The Clipboard Logger continuously monitors and logs content copied to or from the system clipboard.

Often, users paste sensitive information like passwords, card numbers, or API keys rather than typing them. This module ensures that such data isn't missed. Like keystrokes, clipboard data is timestamped and stored in a separate log file ( `clipboard_logs.txt` ), ensuring full coverage of input activity.

## 10. screens directory :

Acts as a dedicated storage folder for visual evidence.

This is a file system directory where screenshots captured by the Screen Logger are saved.

Screenshots provide visual verification of user activity. Each image file can be linked to the corresponding entry in `alert_logs.txt` based on timestamps. Investigators or administrators can review this directory to gain a clearer picture of what the user was doing when a suspicious event occurred. Screenshots might show visited websites, typed forms, email drafts, or sensitive files being accessed.

## 2. Roles :

### 1. Admin / Owner :

The Admin or Owner is the rightful user who is in charge of the whole lifecycle of the keylogger software , from installation to management and maintenance. This encompasses installing the program on the target PC, configuring what data to capture ( keystrokes, clipboard, and screenshots), and adjusting system settings like alert sensitivity. The Admin also manages the machine learning components, including retraining or updating the Naïve Bayes classifier and adjusting NLP rules, to improve threat detection. They review system outputs, such as `alert_logs.txt`, `clipboard_logs.txt`, and screenshots in the screens directory, periodically to identify malicious activity and act accordingly. They ensure that the system works correctly, adapts to emerging threats, and supports investigative or security actions when necessary

### 2. End User :

The **End User** is the individual using the device on which the keylogger is installed. This person is being monitored—**often without their direct knowledge** in surveillance scenarios

## 3. Functional Workflow :

### 1. Start Program

The software launches and initializes all components

### 2. Initialize Modules

Sets up:

- Keyboard Listener
- Clipboard Logger (if enabled)
- NLP & ML pipelines
- Logging mechanisms

### 3. Capture Keypress :

The keyboard listener detects a keystroke.

### 4. Process Key :

Keystroke is buffered and formatted for readability and context (e.g., detect backspace, shift, etc.).

### 5. Log to File :

The processed keystroke, along with a timestamp, is written to the log file.

### 6. Buffer Keystrokes :

Inputs are temporarily stored in chunks (e.g., by sentence or time window) for analysis.

### 7. NLP Processing :

Natural Language Processing module interprets the semantic meaning of the buffered keystrokes.

### 8. ML Classifier :

Analyzes the processed input to detect suspicious or sensitive behavior.


### 9. If Suspicious ==> Alert :

Generates an alert if the input is flagged as suspicious.

Alert is recorded in `alert_logs.txt`.

### 10. Clipboard / Screenshot Logging :

- Clipboard logger saves copied data to `clipboard_logs.txt`.
- Screen logger captures the screen context and stores images if alert is triggered.

 All modules continue to operate silently until manually stopped by the admin or system restart

## 4. Component description and communication

### 1. Keyboard Listener:

**Data Exchanged:**

Raw keystrokes (e.g., "a", "b", "Shift", "Ctrl+V", etc.)

->Think of this as the ears of the system. It picks up every keystroke as soon as it's pressed. At this stage, the data is raw and unprocessed,it doesn't say when or where the key was pressed, just that it was. It's like listening to someone speak without context.

## 2. Logger :

### Data Exchanged:

Formatted keystrokes with timestamps

-> Imagine someone writing down everything they hear in a notebook, but also jotting down the time and the room they were in when they heard it. That's what this component does,it turns raw letters into structured information, making the data much easier to analyze later.

## 3. Data Formatter / Encryptor :

### Data Exchanged:

Encrypted or secured logs (e.g., encoded strings)

-> Since the logs can contain sensitive information like passwords or messages, this component acts like a security guard. It ensures the data is either encrypted or encoded so no one can just open the file and read it easily.

## 4. NLP Module :

### Data Exchanged:

Flagged keywords, phrases, or analyzed text segments

-> This component adds intelligence to the system. It's not just about logging anymore—it's about understanding.

## 5. ML Model :

### Data Exchanged:

Flags or alerts (e.g., "trigger alert", "safe input")

-> Think of this as the brain making decisions. It looks at the words, patterns, and typing behavior to figure out: is the user typing something dangerous or unusual? It helps reduce false alarms by learning from past data.

## 6. Alert Logger :

### Data Exchanged:

Alert messages or flagged data sequences

->This is like the alarm system. When something crosses a certain threshold of risk (determined by the ML model), it goes off and creates a separate entry. It could even trigger further actions, like taking a screenshot.

## 7. Screen Logger :

**Data Exchanged:**

Image files (with timestamps)

-> It works like CCTV for your computer screen. If the system thinks something interesting or risky is happening, it takes a visual snapshot for evidence or further analysis.

## 8. Clipboard Logger :

**Data Exchanged:**

Text copied to clipboard

-> Many users copy and paste sensitive data—like passwords, email addresses, or bank details. This component quietly watches the clipboard and records anything that might be useful or sensitive. It's a way to catch things that might not be typed directly.

# 5. Tools and Libraries

In our project, we selected a set of tools and libraries that will support each key functionality of the keylogger. These libraries were chosen for their reliability, ease of integration, and suitability for a Windows-based environment. Each library plays a specific role in capturing, processing, or analyzing the data.

## 1. General System & Functionality Libraries

These libraries handle core functions like capturing data, managing files, timestamps, and running background processes:

### 1.1. pynput:

Captures keyboard input in real-time.

### 1.2 datetime:

Adds timestamps to logged keystrokes.

### 1.3 pyperclip:

Monitors clipboard content.

### 1.4 PIL.ImageGrab / pyautogui:

Captures screenshots for screen logging.

### 1.5 os:

Manages file operations (e.g., creating folders, accessing file paths).



### 1.6 time:

Controls delays and scheduling.

### 1.7 threading:

Runs multiple functions in parallel (e.g., listening for keystrokes while processing NLP analysis in the background).

## 2. Machine Learning & NLP-related Libraries

These libraries handle the interpretation, classification, and analysis of typed text:

### 2.1 nltk:

Used for text preprocessing (tokenization, stopword removal, stemming) to prepare keystroke data for analysis.

### 2.2 scikit-learn

Provides machine learning algorithms (like Naive Bayes) to classify user input and detect patterns or categories of activity.

## 6. Development Phases

To build our keylogger software effectively and responsibly, we've divided the process into multiple development phases. Each phase focuses on one specific functionality, starting from basic keystroke capturing to intelligent behavior classification and alerting. Here's how we approached the implementation.

### Phase 1: Core Keylogger

In this first phase, our goal is to capture keystrokes and save them with timestamps.

#### Task 1: Set up the key listener using pynput

We'll use the `pynput` library to listen for keyboard inputs:

```
from pynput.keyboard import Listener

def on_press(key):
    print(f"Key pressed: {key}")

with Listener(on_press=on_press) as listener:
    listener.join()
```

## Task 2: Log each keystroke with the current date and time

To keep track of when each key was pressed, we'll add timestamps using `datetime`:

```
from datetime import datetime

def on_press(key):
    with open("logs.txt", "a") as f:
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        f.write(f"{timestamp}: {key}\n")
```

Finally, we'll combine them so every key gets logged:

```
def on_press(key):
    log_key(key)
```

## Phase 2: Data Formatting & Encryption

Once keystrokes are captured, we need to **make the logs more readable and secure sensitive data**.

### Task 1: Translate special keys into readable symbols

For example, replacing `Key.space` with an actual space:

```
def format_key(key):
    if str(key) == 'Key.space':
        return ' '
    elif str(key) == 'Key.enter':
        return '\n'
    else:
        return str(key)
```

### Task 2: Add encryption to protect log files

We can use the `cryptography` library's Fernet encryption to secure the logs:

```
from cryptography.fernet import Fernet

key = Fernet.generate_key() # Save this key somewhere safe!
cipher = Fernet(key)

with open("logs.txt", "rb") as f:
    data = f.read()

encrypted_data = cipher.encrypt(data)

with open("logs_encrypted.txt", "wb") as f:
    f.write(encrypted_data)
```

==> This phase ensures the logs are clean and optionally encrypted to protect sensitive content.

## Phase 3: Activity Analyzer

Here, we'll start using Natural Language Processing (NLP) to analyze what the user is typing.

## Task 1: Tokenize and clean the logged text

We'll use `nltk` to split the text into words and remove common words (stopwords):

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords

nltk.download('punkt')
nltk.download('stopwords')

with open("logs.txt", "r") as f:
    text = f.read()

tokens = word_tokenize(text)
filtered_tokens = [word for word in tokens if word.lower() not in stopwords.words('english')]
```

## Task 2: Classify the activity using a machine learning model (Naive Bayes)

We'll define simple training data and let the model decide if the content is normal or sensitive:

```
from nltk import NaiveBayesClassifier

training_data = [
    ({'contains(password)': True}, 'sensitive'),
    ({'contains(bank)': True}, 'sensitive'),
    ({'contains(hello)': False}, 'normal')
]

classifier = NaiveBayesClassifier.train(training_data)

features = {'contains(password)': 'password' in filtered_tokens}
result = classifier.classify(features)

print(f"Activity detected: {result}")
```

==> This allows the software to decide dynamically if the user is typing sensitive information, instead of relying on hardcoded trigger words.

## Phase 4: Alert System

If sensitive activity is detected, we want the system to respond appropriately.

### Task 1: Log sensitive events to a separate alert file

```
from datetime import datetime

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

## Task 2: Print a real-time alert in the console

```
if result == 'sensitive':  
    log_alert(text)  
    print("⚠ Sensitive input detected!")
```

==> This keeps sensitive data highlighted and separated from normal logs.

## Phase 5: Optional Extensions

For more advanced monitoring, we can add clipboard and screen logging.

### Task 1: Monitor clipboard using `pyperclip`

```
import pyperclip  
import time  
  
last_clipboard = ""  
while True:  
    current_clipboard = pyperclip.paste()  
    if current_clipboard != last_clipboard:  
        with open("clipboard_logs.txt", "a") as f:  
            f.write(f"{datetime.now()} → {current_clipboard}¥n")  
        last_clipboard = current_clipboard  
    time.sleep(5)
```

### Task 2: Capture screenshots with `PIL.ImageGrab`

```
from PIL import ImageGrab  
  
screenshot = ImageGrab.grab()  
screenshot.save(f"screenshot_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png")
```

==> These are optional features that enhance monitoring beyond just keystrokes.

## Phase 6: Testing and Ethical Safeguards

Finally, we need to make sure the software is ethical, transparent, and reliable.

### Task 1: Print a consent message before starting

```
print("This software records keystrokes. By continuing, you agree to its operation.")  
consent = input("Do you agree? (y/n): ")  
if consent.lower() != 'y':  
    exit("Program exited due to lack of consent.")
```

### Task 2: Test that logs, encryption, NLP, and alerts work as intended

We'll do trial runs to check:

- Are all keys being captured?

- Are special keys translated properly?
  - Does the NLP classifier give accurate results?
- ==> This phase ensures ethical compliance and functionality.

## 7. Use cases

### 1. Employee Monitoring in Enterprises

#### Purpose:

Beyond enforcing compliance, keyloggers help detect inefficiencies, insider threats, and unauthorized access to sensitive systems. Monitoring keystrokes can reveal if employees are accessing non-work-related platforms or sharing confidential information through unapproved channels.

#### Example:

- Detecting repeated attempts to access restricted systems.
- Identifying copy-pasting of client databases into personal storage or email.
- Monitoring use of messaging platforms (e.g., Slack, Teams, WhatsApp Web) for data exfiltration.

### 2. Parental Control

#### Purpose:

In addition to content filtering, keystroke logging gives insight into private communications or search queries children may attempt to hide. It offers more context than just URL or app logs.

#### Example:

- Monitoring for signs of cyberbullying ("kill myself," "nobody likes you," etc.).
- Detecting attempts to bypass parental controls (e.g., searching "how to disable SafeSearch").
- Logging activity in incognito/private browser modes that wouldn't be captured by history logs.

### 3. IT Forensics and Security Auditing

#### Purpose:

In a post-incident investigation, keylogs can provide granular evidence of who typed what and when, which is useful when other audit trails are deleted or corrupted. It's a reconstructive tool.

### **Example:**

- Identifying if an internal user typed a suspicious command (e.g., `rm -rf` or `netstat`) at the time of a breach.
- Verifying if an attacker used stolen credentials to log into a system and issue commands manually.

## **4. Usability Testing & Behavioral Analysis**

### **Purpose:**

Keyloggers help understand how real users interact with software, highlighting friction points, typographical errors, and mental models behind user actions.

### **Example:**

- Measuring how long it takes a user to complete a form or how often they backspace.
- Analyzing the order of field completion (e.g., skipping address line, returning later).
- Tracking hesitation or confusion through prolonged inactivity or erratic typing.