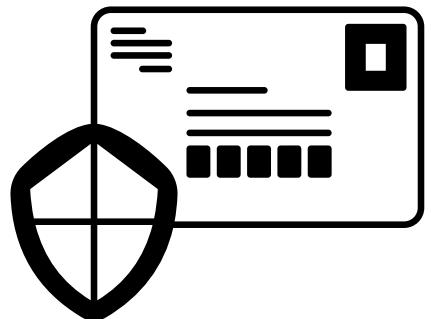


Digital Privacy Tool



By: Abdul Haseeb
29/06/2025

For my mother, Zubaida — who taught me to think in tiny ways, ignore the noise, stay sharp, and build things like this before I even knew I could.

Table of Content

1. Introduction
2. Requirements
3. Implementation & Setup
4. Working
5. Code
6. Conclusion & Future Scope

Introduction

DigitalPrivacy Tool: Real-Time Clipboard Security Monitor

Clipboard-based data leakage represents an overlooked attack vector in enterprise security environments. Employees in SOC teams, financial institutions, and customer service organizations routinely copy sensitive information—passwords, OTPs, customer IDs, payment card numbers—into their system clipboard, where this data remains temporarily accessible, often unencrypted and unmonitored. A single accidental paste into an unsecured channel (Slack, email, browser) can trigger credential compromise, identity theft, or compliance violations.

Technical Implementation: DigitalPrivacy Tool is a lightweight Python-based clipboard monitoring system that provides real-time detection of sensitive data through a hybrid classification approach. I built the core engine using pyperclip for continuous clipboard polling, implementing regex pattern matching for immediate identification of structured data types including email addresses (RFC 5322 compliant), Indian PAN numbers, payment card sequences (13-16 digits), and phone numbers. For unstructured or ambiguous content that evades regex detection, the system integrates local AI inference via Ollama's Mistral model, which performs contextual analysis to identify passwords, API keys, and other credential patterns without external data transmission.

The architecture maintains privacy-by-design principles: all processing occurs locally with zero cloud dependencies, ensuring sensitive clipboard content never leaves the user's device. When potentially risky data is detected, the system triggers threaded Tkinter popup alerts for immediate user notification while logging all clipboard activity with ISO 8601 timestamps to clipboard_log.txt for post-incident forensic analysis.

Key Technical Components:

- Multi-pattern regex engine covering PAN (Indian tax ID), payment cards (Luhn algorithm validation capable), email (RFC-compliant), and phone number formats
- Local LLM integration (Mistral 7B) via Ollama API for zero-latency, privacy-preserving inference
- Asynchronous alert system using Python threading to prevent UI blocking during active monitoring
- Companion log analysis tool (analyze_logs.py) providing statistical reporting on clipboard usage patterns, sensitive data frequency, and temporal analysis

Enterprise Security Applications: This tool addresses clipboard security gaps in high-risk operational environments including Security Operations Centers (SOC), banking terminals, healthcare administration systems, and customer support desks where PII exposure through copy-paste workflows creates compliance risks under regulations including GDPR Article 32 (security of processing), PCI DSS Requirement 3 (cardholder data protection), and India's DPDP Act 2023. The solution enables organizations to implement clipboard-level Data Loss Prevention (DLP) controls without deploying enterprise-grade endpoint security suites.

Impact & Validation: By operating entirely offline with local AI inference, the tool demonstrates feasibility of privacy-preserving security monitoring suitable for air-gapped environments and zero-trust architectures. The hybrid detection methodology (regex + LLM) achieves coverage of both structured and unstructured sensitive data patterns while maintaining sub-second response times for real-time user alerts, preventing accidental data exposure before it propagates to external systems.

This is where my prototype comes in — the DigitalPrivacyTOOL.

This tool is a lightweight, real-time clipboard analyzer designed to detect sensitive data such as emails, phone numbers, PAN numbers, and card details. It uses a combination of regex pattern matching (for local detection) and AI-powered classification using the Ollama + Mistral model. The tool provides popup alerts when sensitive data is detected and logs every event with timestamps for auditing via an `analyze_logs.py` script.

Built using Python, pyperclip, tkinter, and local AI integration, this project is both a proof of concept and a personal attempt at addressing clipboard-level data negligence. The goal is to raise awareness that even small digital behaviors, like copying a password, can become major security risks if left unchecked.

I hope this project sheds light on this under-discussed threat vector and inspires better habits and tools to secure even the smallest parts of our digital lives.

Requirements

Physical requirements

- A computer/laptop running Windows, Linux, or macOS
- Minimum 4GB RAM
- At least 100MB of free storage space
- Internet connection (required for AI-based classification via Ollama)

Software requirements

- Python 3.8+ installed and added to your system PATH
- Ollama installed and running locally:
 - → Download from <https://ollama.com>
- Ensure the mistral model is pulled using:
“ollama run mistral”
- The following Python libraries:
 - pyperclip (clipboard access)
 - requests (sending data to the AI API)
 - tkinter (for GUI popup alerts /comes pre-installed with Python in most systems)

Quick Setup (if physical requirements are met)

If your system meets the above, you can directly install the dependencies using the provided requirements.txt file:

- `pip install -r requirements.txt`

Implementation & Setup

STEP 1 : Clipboard Access using Python

- First thing I needed: a way to constantly check what a user copies.
- I used the pyperclip module:

```
import pyperclip  
text = pyperclip.paste()
```

I wrote a loop to keep checking for changes every second:

```
prev_text = ""  
while True:  
    text = pyperclip.paste()  
    if text != prev_text:  
        print("New clipboard copy:", text)  
        prev_text = text
```

STEP 2: Added Regex to Detect Sensitive Stuff

- I wanted to catch emails, phone numbers, PAN, card numbers etc.
- So I wrote regex like this:

```
patterns = {  
    "Email": r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}",  
    "Phone": r"\b\d{10}\b",  
    "Card": r"\b(?:\d[-]*?){13,16}\b",  
    "PAN": r"\b[A-Z]{5}[0-9]{4}[A-Z]\b"  
}
```

Then looped through to see if the copied text matched any:

```
for label, pattern in patterns.items():  
    if re.search(pattern, text):  
        print(f"{label} Detected: {text}")
```

STEP 3: Logging Everything with Timestamps

- I didn't want to just print stuff, so I created a log file:

```
with open("clipboard_log.txt", "a") as f:  
    f.write(f"[{timestamp}] Copied: {text}\n")
```

Used datetime to get the time of copy:

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

STEP 4: Popup Alert using Tkinter

- I wanted the tool to alert the user if something sensitive got copied.
- So I used tkinter to make a small GUI window pop up:

```
def show_popup_alert(message):  
    root = tk.Tk()  
    root.title("Sensitive Alert")  
    root.geometry("400x100")  
    label = tk.Label(root, text=message)  
    label.pack()  
    button = tk.Button(root, text="OK", command=root.destroy)  
    button.pack()  
    root.mainloop()
```

Wrapped this in a thread so it doesn't freeze the main loop:

```
from threading import Thread  
Thread(target=popup).start()
```

STEP 5: AI Integration via Ollama

- This is where I added AI to analyze clipboard content if regex missed something.
- I used requests to connect to a locally running LLM via Ollama:

```
response = requests.post(  
    'http://localhost:11434/api/generate',  
    json={  
        "model": "mistral",  
        "prompt": f"Analyze this clipboard content: {text}. Is it sensitive?",  
        "stream": False  
    }  
)  
result = response.json()["response"]
```

Only if regex didn't detect anything, I sent it to AI:

if not found:

```
ai_response = ask_ollama(text)  
if "Sensitive" in ai_response:  
    show_popup_alert(ai_response)
```

STEP 6: Built Log Analyzer Tool

- I wanted a second script to read the logs and summarize usage.
- Made a new Python file analyze_logs.py:

```
with open("clipboard_log.txt", "r") as f:  
    logs = f.readlines()
```

- Counted how many entries were Sensitive / Not Sensitive using conditions.
- Printed a small CLI report like:

Total Entries: 50

Sensitive: 22

Not Sensitive: 28

STEP 7: Project Clean-Up & Documentation

- Wrote a requirements.txt:
- pyperclip
- requests
- tk

created .gitignore to ignore logs and env files:

```
__pycache__/  
clipboard_log.txt  
*.pyc
```

It constantly monitors your clipboard. When you copy something, it first checks for known patterns like email or card number. If none are found, it asks an AI model running locally (Mistral via Ollama) to determine if the content is sensitive. If flagged, it immediately shows a popup alert and logs the entry. Later, users can audit their logs using a separate analyzer script.

Documented the entire project with setup instructions, purpose, screenshots, file structure, and sample logs. (README)

{The syntax code is on page 18}

Working

Once the implementation process is completed without any syntax errors, the DigitalPrivacyTool functions as follows:

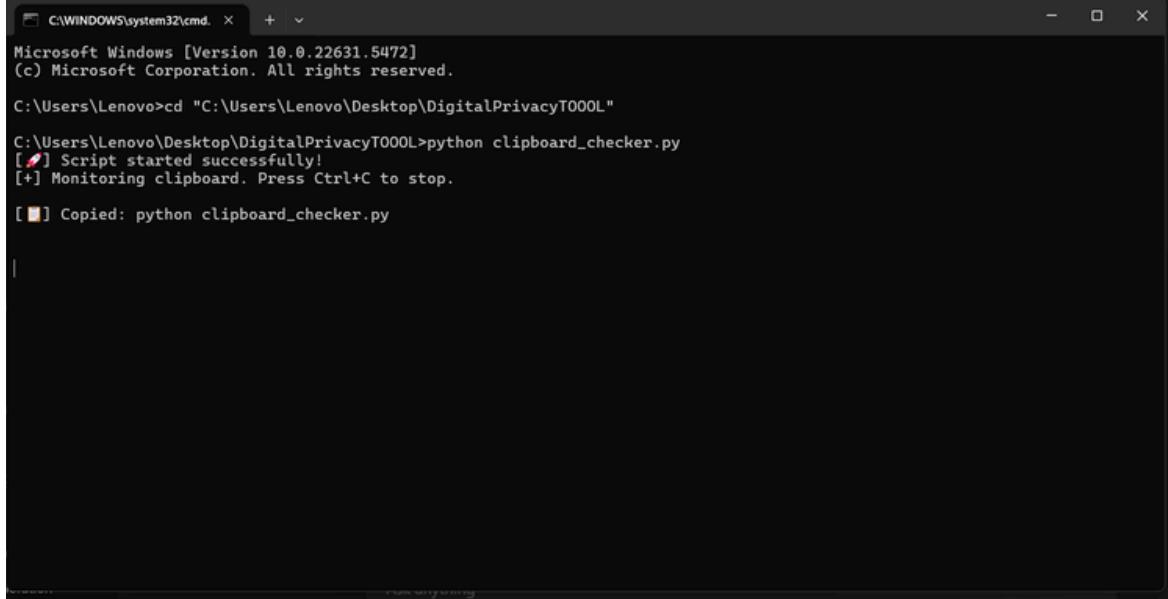
Step 1: Run the Script

Open Command Prompt (CMD) and navigate to the folder where you saved the DigitalPrivacyTool.

Type the following command and press Enter:

```
python clipboard_checker.py
```

Once executed, the script will start running, and you will see the following output on your screen.



```
C:\WINDOWS\system32\cmd. + 
Microsoft Windows [Version 10.0.22631.5472]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd "C:\Users\Lenovo\Desktop\DigitalPrivacyT000L"
C:\Users\Lenovo\Desktop\DigitalPrivacyT000L>python clipboard_checker.py
[!] Script started successfully!
[+] Monitoring clipboard. Press Ctrl+C to stop.

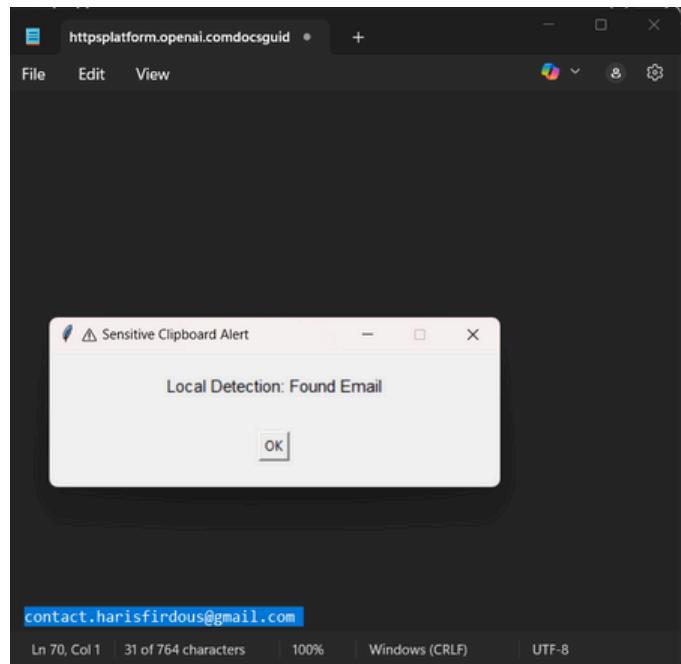
[!] Copied: python clipboard_checker.py
```

Step 2: Test Clipboard Monitoring

Once your script is running, try copying any text to your clipboard. For example, as shown in the screenshot below, I copied my friend's email address:

contact.harisfirdous@gmail.com

As soon as the email is copied, a pop-up alert appears on the screen, detecting and identifying the content as an email address.

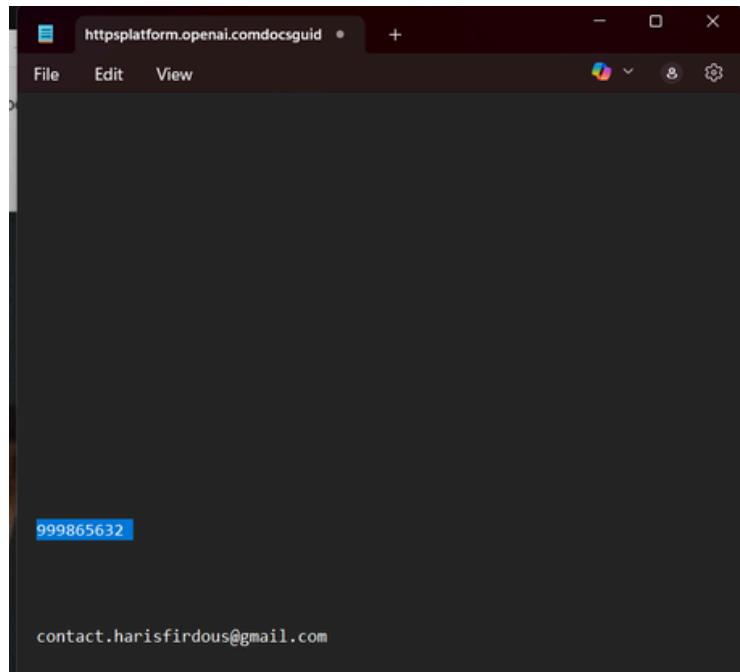


Step 3: Test with Numeric Data

Now, try copying a number or a sequence of digits to test how the tool handles numeric data.

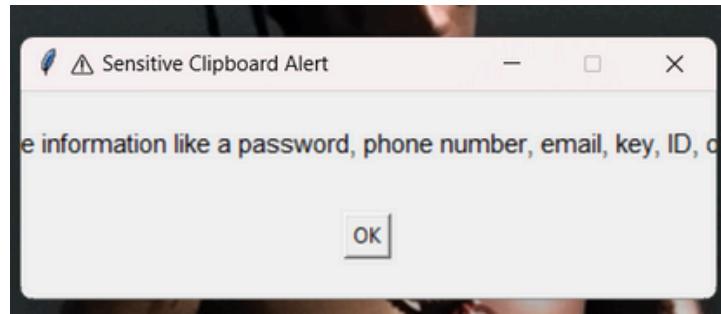
As shown in the screenshot below, I copied a random string of numbers to see if the DigitalPrivacyTool can detect and classify the content.

If the number matches a recognizable pattern (like a phone number, credit card number, etc.), the tool will trigger a pop-up alert accordingly.



As soon as I copy a number to my clipboard, a new pop-up alert appears — identifying the copied content as a password, phone number, or email based on pattern recognition.

In this case, since I copied a random sequence of numbers, the tool flagged it as potentially sensitive data.



Step 5: AI-Powered Analysis in CMD

Returning to the Command Prompt (CMD) window — as shown in the screenshot below — you'll see the complete analysis generated by Mistral AI.

This analysis evaluates the copied text and determines whether it is classified as sensitive or non-sensitive based on its content.

```
C:\WINDOWS\system32\cmd. x + v
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd "C:\Users\Lenovo\Desktop\DigitalPrivacyT000L"
C:\Users\Lenovo\Desktop\DigitalPrivacyT000L>python clipboard_checker.py
[!] Script started successfully!
[+] Monitoring clipboard. Press Ctrl+C to stop.

[!] Copied: python clipboard_checker.py

[!] Mistral says: Not Sensitive. The provided text appears to be a command for running a Python script named "clipboard_checker.py", which itself does not contain sensitive information such as passwords, phone numbers, emails, keys, IDs, financial info, etc. However, it's important to note that the actual function of this script may or may not involve handling sensitive data. It's always good practice to ensure you understand what scripts are doing before running them, especially if they have access to your system's clipboard.

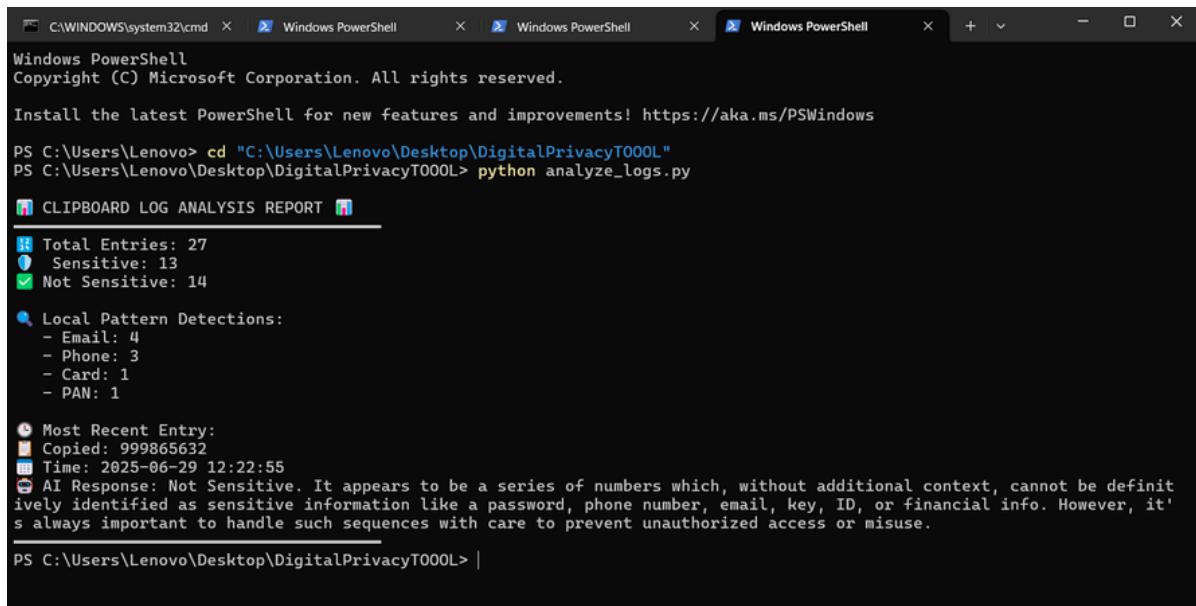
[!] Copied: contact.harisfirdous@gmail.com
[!] Local Detection: Found Email
[!] Copied: 999865632
[!] Mistral says: Not Sensitive. It appears to be a series of numbers which, without additional context, cannot be definitively identified as sensitive information like a password, phone number, email, key, ID, or financial info. However, it's always important to handle such sequences with care to prevent unauthorized access or misuse.

[!] Stopped by user.
```

Final Step: Analyze Clipboard Logs

You can also review and analyze the logged clipboard activity by running the file `analyze_log.py` in a separate Command Prompt window. This script helps you track everything that has been copied to the clipboard over time.

As shown in the screenshot below, the log provides a summary report — listing all copied items and classifying them as either sensitive or non-sensitive, along with the total count for each.



```
C:\WINDOWS\system32\cmd  X  Windows PowerShell  X  Windows PowerShell  X  Windows PowerShell  X
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Lenovo> cd "C:\Users\Lenovo\Desktop\DigitalPrivacyT000L"
PS C:\Users\Lenovo\Desktop\DigitalPrivacyT000L> python analyze_logs.py

CLIPBOARD LOG ANALYSIS REPORT

Total Entries: 27
Sensitive: 13
Not Sensitive: 14

Local Pattern Detections:
- Email: 4
- Phone: 3
- Card: 1
- PAN: 1

Most Recent Entry:
Copied: 999865632
Time: 2025-06-29 12:22:55
AI Response: Not Sensitive. It appears to be a series of numbers which, without additional context, cannot be definitely identified as sensitive information like a password, phone number, email, key, ID, or financial info. However, it's always important to handle such sequences with care to prevent unauthorized access or misuse.

PS C:\Users\Lenovo\Desktop\DigitalPrivacyT000L> |
```

Syntax Code

```
# 🤖 Ollama / Mistral AI Fallback
def ask_ollama(text):
    try:
        response = requests.post(
            'http://localhost:11434/api/generate',
            json={
                "model": "mistral",
                "prompt": f"You're a cybersecurity expert. Analyze the following clipboard text: \'{text}\'". Determine if it is sensitive (e.g., password, phone number, email, key, ID, or financial info). Respond with just: Sensitive or Not Sensitive and give a short reason.",
                "stream": False
            }
        )
        result = response.json()
        return result["response"].strip()
    except Exception as e:
        return f"[ERROR] Could not reach local AI: {e}"

def show_popup_alert(message):
    def popup():
        root = tk.Tk()
        root.title("⚠️ Sensitive Clipboard Alert")
        root.geometry("400x120")
        root.resizable(False, False)
        label = tk.Label(root, text=message, padx=20, pady=20, font=("Arial", 11))
        label.pack()
        button = tk.Button(root, text="OK", command=root.destroy)
        button.pack(pady=10)
        root.mainloop()

    Ti_response) # 🎙 Show popup for AI result

    prev_text = text
    time.sleep(1)
except KeyboardInterrupt:
    print("\n[!] Stopped by user.")
    break
if __name__ == "__main__":
    monitor_clipboard()
```

```
hread(target=popup).start()

# 📋 Clipboard Monitoring
def monitor_clipboard():
    print("[+] Monitoring clipboard. Press Ctrl+C to stop.\n")
    prev_text = ""
    while True:
        try:
            text = pyperclip.paste()
            if text != prev_text and text.strip() != "":
                print(f"[📋] Copied: {text}\n")

        # Run Regex Detection First
        found = detect_sensitive_data(text)
        if found:
            local_detection = f"Local Detection: Found {', '.join(found)}"
            print(f"[⚠] {local_detection}\n")
            log_to_file(text, local_detection)
            show_popup_alert(local_detection) # 🎙 Show popup

        else:
            ai_response = ask_ollama(text)
            print(f"[🤖] Mistral says: {ai_response}\n")
            log_to_file(text, ai_response)
            if "Sensitive" in ai_response:
                show_popup_alert(a
```

Log Analyzer

```
import re
from collections import Counter

def analyze_logs(filename="clipboard_log.txt"):
    try:
        with open(filename, "r", encoding="utf-8") as file:
            data = file.read()

        copied_items = re.findall(r"Copied: (.+)", data)
        ai_responses = re.findall(r"AI Response: (.+)", data)
        timestamps = re.findall(r"\[(\d{4}-\d{2}-\d{2}\d{2}:\d{2}:\d{2})\]", data)

        print("\n📊 CLIPBOARD LOG ANALYSIS REPORT 📊")

    print("-----")
    print(f"13 Sensitive: {sens_count}")
    print(f"<span style='color: #28a745; font-weight: bold;">14 Not Sensitive: {not_sens_count}")

    # Local pattern counts
    patterns = {
        "Email": r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}",
        "Phone": r"\b\d{10}\b",
        "PAN": r"\b[A-Z]{5}[0-9]{4}[A-Z]\b",
        "Card": r"\b(?:\d[ -]*?)\{13,16\}\b"
    }
```

```
local_counts = Counter()
for item in copied_items:
    for label, pattern in patterns.items():
        if re.search(pattern, item):
            local_counts[label] += 1

print("\n🔍 Local Pattern Detections:")
for label, count in local_counts.items():
    print(f"- {label}: {count}")

if copied_items:
    print("\n🕒 Most Recent Entry:")
    print(f"📅 Copied: {copied_items[-1]}")
    print(f"📅 Time: {timestamps[-1]}")
    print(f"🤖 AI Response: {ai_responses[-1]}")

print("-----")
except FileNotFoundError:
    print("[✗] Log file not found!")
except Exception as e:
    print(f"[⚠️] Error during analysis: {e}")

if __name__ == "__main__":
    analyze_logs()
```

Conclusion & Futurescope

DigitalPrivacy Tool addresses an underestimated security vulnerability: clipboard-based data exposure through routine copy-paste operations. By providing real-time detection and user alerts before sensitive information is accidentally shared, the tool transforms clipboard monitoring from a passive logging mechanism into an active defense layer. Built with privacy-first principles (local processing, offline AI inference), the system demonstrates that effective security controls can be implemented without compromising user data sovereignty. This proof-of-concept validates the viability of clipboard-level DLP for resource-constrained environments including small businesses, SOC analyst workstations, and security-conscious individual users who lack access to enterprise endpoint protection platforms.

Future Scope:

Enterprise Integration:

- Develop Windows Service / systemd daemon for persistent background monitoring without user intervention
- Create centralized logging dashboard for SOC teams to aggregate clipboard activity across organizational endpoints
- Integrate with SIEM platforms (Splunk, ELK Stack) via syslog forwarding for correlation with other security events
- Build policy enforcement engine allowing administrators to block clipboard operations when sensitive patterns are detected

Enhanced Detection Capabilities:

- Expand regex library to cover IBAN, SWIFT codes, social security numbers (SSN), Aadhaar numbers, passport IDs, and cryptocurrency wallet addresses
- Implement machine learning-based anomaly detection to identify unusual clipboard usage patterns indicating potential insider threats
- Add support for image-based sensitive data (OCR analysis of screenshots containing credentials)
- Integrate with breach intelligence feeds to cross-reference copied credentials against known compromised databases

Cross-Platform & UX:

- Port to macOS and Linux with native clipboard API integration
- Develop browser extension to monitor web-based clipboard operations (JavaScript navigator.clipboard API)
- Create configurable sensitivity levels (strict/moderate/permissive) for different organizational risk appetites
- Add encrypted clipboard vault feature for temporarily storing sensitive data with automatic expiration

Compliance & Audit:

- Generate automated compliance reports mapping clipboard incidents to regulatory requirements (GDPR, HIPAA, PCI DSS)
- Implement tamper-evident logging with cryptographic signatures for forensic integrity
- Add user training mode that explains why specific clipboard content was flagged, promoting security awareness

Thank You.

About the Creator

Abdul Haseeb is a 22-year-old cybersecurity professional from Hyderabad, currently based in Bangalore, India. He works as a SOC Analyst with a focus on digital privacy, ethical technology, and AI-assisted problem-solving. Passionate about learning and exploring the ever-evolving landscape of cybersecurity, he enjoys music, building tools that make digital systems more secure, transparent, and user-aware.

For collaborations, thoughts, or feedback:

 contact.haseebabdul@gmail.com