Purple Keylogger Project (aka Snakey)

Project	Status	Related files	Notes
Research	Launched	File	
Coding	In progress	File	
Testing	In progress	File	

Abstract

Keyloggers, traditionally recognized as malicious tools for unauthorized data capture, or “red team”, are often associated with cyberattacks such as credential theft and espionage. However, within the scope of *blue-teaming*, these tools can serve legitimate and strategic purposes when utilized ethically and responsibly.

The research explores the deployment of a keylogger to monitor user behavior, detect anomalous activity, and respond to insider threats while adhering to ethical and legal frameworks and playbooks alike. By analyzing their technical implementation and integrating them into incident response and monitoring workflows, this study seeks to identify practical applications and address potential risks associated with insider threat actors or non-beneficial behaviors within a company. Special attention is given to the balance between organizational security needs and requirements, and the preservation of individual privacy rights according to compliance playbooks, namely “**Código de Integridade da Universidade da Beira Interior** “.

Through a combination of theoretical analysis, practical experimentation, and case studies, this work evaluates the effectiveness, limitations, and best practices for employing keyloggers in defensive cybersecurity operations directed at insider and outsider threats alike. The findings aim to contribute to the development of innovative Blue Team strategies, reframing keyloggers as valuable assets in strengthening organizational resilience against

evolving cyber threats we face daily, thus lastly showing how we can turn a “weapon” into something *pro bono*.

- Can the program be installed as a stealth component? (in python)

- Anomalies like failed authentication attempts or rapid, repetitive inputs (indicative of brute-force attacks).
- Activities occurring during off-hours or from unauthorized geographical locations.

- Actions performed by users flagged for heightened monitoring (e.g., based on prior suspicious behavior). (??)

- Escalation path (1st level warning/2nd level trigger immediate escalation (SOC lv2 for example/3rd level quarantine the system itself) (?)

- Use Machine learning models (Typing speed/sequence of high risk commands etc)

This app could flag a user for using cmd.exe and net user for example at 00:00 pm (unusual schedule access to certain apps should be a trigger followed by important writing afterwards)

Safeguards

Comms (logs):

- AES-256 with Kyber Crystals (is it doable??) - Most likely too complicated with *Kyber*
    - ❖ Use CRYSTALS-Kyber to securely establish session keys for AES-256.
    - ❖ AES-256 with a Simulated Secure Key Exchange (more likely)
    - ❖ Hybrid Cryptography (using AES but use post-quantum key exchange algorithms)

    - ○ Encrypt keystroke data in real-time with AES-256 in an authenticated encryption mode (e.g., AES-GCM).

    - Ensure generated session keys are unique for each session and stored securely.

CRYSTALS-Kyber for Session Key Exchange:

- *Perform key exchange when the logger starts.*
- *Store the shared session key in memory only.*

Encrypt Data in Real-Time:

- Use AES-GCM with the session key to encrypt keystrokes.
- Append encrypted logs to a file or send them over a secure channel.

Secure Storage of Encrypted Logs:

- Use a long-term key (protected by HSM or environment variables) to re-encrypt session keys or logs if needed.

Secure key storage: (Should include key rotation?)

- Using Hardware Security Modules (HSMs)
- During runtime but with “Zeroing out” or Using secure memory
- Storing them in an encrypted way in memory
- Google Cloud Key Management Service (KMS) ?
- Zeroing out memory/overwriting memory

(Making this project as secure as it can be, for a nicer “polishing” and more realistic)

IDS (Part of the Keylogger)

- Measure typing speed (e.g., characters per second or keystrokes per minute).

- Flag sequences exceeding a threshold for human input rates (e.g., > 150 words per minute).

- Build a dictionary of high-risk commands (e.g., `whoami`, `net user`, `wmic`, `powershell.exe`, etc.).

- Track execution patterns involving common malware or attack behaviors (e.g., file path changes, registry edits)

- Correlate keystrokes with browser activity or application usage.

- Flag if passwords or sensitive information are typed into unexpected applications
- Use TOTP (?)

Analysis (Dashboard)

- Heatmaps of input freq by time of day (set schedule from 8:30 am to 6 pm deemed as acceptable)

- Charts for command execution trends

- Live graphs of flagged behaviors

What if a trusted user is flagged, how should it be averted?

Ask for a credential generated in that login session? aka (Session Key) -> needs to be overwritten later

Dynamic trust levels? (Instead of binary)
Periodic Key Rotation (__ time spent or __ flagged commands executed

#Problem_1:   "False Positives"

How to address them?

Being Context-Aware:

Time of Day: Is the activity happening during working hours?
User Role: Does the user's role justify the flagged behavior ——---*(i'm not sure how doable this is?)*-------
Location: Is the action occurring from a known/acceptable IP address or location?

Command Whitelisting:

Allow common commands (e.g., ls, cd) without triggering alerts. which can be considered benign for the system

Feedback Loop:

Once the anomaly is deemed trustworthy, implementing it in the code so that next time the system doesn't flag it as possible danger (could be exploited)

for example whitelist a typing speed of 200< (function)

Possible Gamification of Risk:

Instead of binary decisions (alert/no alert), assign risk scores to activities —-*(possibly more demanding but maybe more specific)*------

Suspicious command: +7 points.
High typing speed: +3 points.
Unknown application: +5 points.

if total_score >= 10:

trigger alert

example of implementation

- [ ] Using Machine learning to teach the system how to tell apart from a given use case

Online Behavior

Automate Updates with Threat Intelligence Feeds

Use threat intelligence feeds to dynamically update your list of flagged IPs. Sources include:

- **AbuseIPDB:** Provides information on malicious IPs.

- **Emerging Threats:** Offers public blocklists.

- **VirusTotal API:** Allows querying for IP reputation.

- Use CIDR blocks to check for specific ranges of IP's
- Use for example Slack API for sending real time alerts (?)

Combine with captured Keystroke Data

To link IP monitoring with keystroke logging:

- Capture active window or process associated with keystrokes.
- Correlate the process's network activity with flagged IPs.

Good py libraries for threat monitoring:

maltego-trx

AlienVault OTX (api key)

Screenshots:

- Using *mss, pyautogui*

Goals:

- Trigger screenshot once a suspicious ip has been detected
- Trigger screenshot once certain words have been written

(Potential) Weaknesses:

- ☐ **Grover's algorithm**
- ☑ **~~Shor's algorithm~~**
- ☐ **Side-Channel Attacks**
- ☐ **Improper Padding**
- ☐ **Cache timing attacks**
- ☐ **Branch Prediction Attacks**
- ☐ **Uninstalling the keylogger**
- ☐ **Lack of Code Signing**

Possibility (purple teaming?)

(use js to install python in a stealth way upon clicking on a link)

```
const downloadPython = () => {
const url =
"https://www.python.org/ftp/python/3.11.5/python-3.11.5-amd64.exe";
const fileName = "python-installer.exe";

const anchor = document.createElement("a");
anchor.href = url;
anchor.download = fileName;
anchor.click();

alert("Python installer has been downloaded. Run it to complete the
installation.");
};

document.getElementById("installButton").addEventListener("click",
downloadPython);
```

Besides the good-conduct UBI document, follow as well the NIST CSF framework widely known

Sneaky: NIST Cybersecurity Framework (CSF) Alignment

1. IDENTIFY (Asset Management, Risk Assessment)

Goal: Establish a baseline of user behavior and define security policies.

🔹 Key Features & Implementation

✅ User Role & Context Awareness

- Maintain a **user-role mapping** (e.g., Admins vs. Standard Users).
- Store baseline behaviors per user (e.g., normal login times, typing habits).

✅ Threat Intelligence Integration

- Query **AbuseIPDB, VirusTotal API, AlienVault OTX** for known malicious IPs.
- Automatically update blocklists and compare with **geolocation lookups**.

✅ Baseline User Behavior (Behavioral Profiling)

- Use **machine learning** to analyze keystroke dynamics:
    - **Typing cadence** (speed, pauses, frequent errors).
    - **Command sequences** (e.g., normal software usage vs. hacking commands).
- Store behavior profiles in a **local SQLite or cloud-based database**.

✅ Tracking Privileged Access & Anomalies

- Flag unauthorized access attempts to sensitive directories.
- Monitor privilege escalation attempts (`net user /add`, `whoami /priv`).

---

2. PROTECT (Access Control, Data Security, Safeguards)

Goal: Ensure the security and integrity of collected data while protecting user privacy.

🔹 Key Features & Implementation

✅ Encryption & Secure Storage

- **AES-256 encryption with CRYSTALS-Kyber key exchange:**
    - Generate session keys at runtime.
    - Encrypt logs **in real-time** using **AES-GCM** (Authenticated Encryption).
- Store logs in an **append-only encrypted database** (MongoDB or SQLite).

✅ Stealth & Integrity Protection

- Implement a **hidden install** process (e.g., stealth service on Windows).
- Use **anti-tampering mechanisms**:
    - Periodic **hash verification** of the executable.
    - **Self-repair** if modified or deleted (e.g., using a watchdog process).

✅ Preventative Monitoring for Unauthorized Activities

- Monitor and block:
    - Execution of **cmd.exe, PowerShell, net user** (unless whitelisted).
    - **Registry edits** (modifying startup programs).
- Alert when **USB mass storage** is inserted (data exfiltration risk).

✅ Compliance & Privacy Controls

- Encrypt logs **at rest and in transit**.

- Implement **role-based access control (RBAC)** for log access.
- Ensure compliance with **Código de Integridade da Universidade da Beira Interior**.

---

3. DETECT (Threat Detection, Anomaly Detection, Behavioral Analytics)

Goal: Identify insider threats, brute-force attempts, and unusual system behavior.

🔹 Key Features & Implementation

✅ Keystroke Behavior Analysis

- Detect **excessive typing speed** (>150 WPM → potential automation).
- Flag **rapid, repetitive failed login attempts** (brute force attack).

✅ Command Execution Monitoring

- Track high-risk command sequences (e.g., `whoami`, `wmic process`, `net user`).
- Use regex to flag suspicious PowerShell scripts (`base64 encoded commands`).

✅ Anomaly-Based Contextual Alerts

- **Off-hours access detection** (e.g., user logs in at 2 AM).
- **Location-based flagging** (e.g., user logs in from a foreign country).

✅ Threat Scoring System

- Assign risk scores dynamically:
  - **Suspicious command:** +7 points
  - **High typing speed:** +3 points
  - **Unknown application execution:** +5 points
  - **Total ≥ 10? → Trigger an alert!**

✅ Live Monitoring Dashboard (Real-Time Alerts & Reports)

- Display **heatmaps** of typing activity by time of day.
- Show **charts of flagged commands**.
- Enable **real-time Slack API notifications** for SOC teams.

---

4. RESPOND (Incident Response, Alerting & Automated Actions)

Goal: Implement automated threat mitigation and escalation workflows.

🔹 Key Features & Implementation

✅ Automated Escalation Path

- **1st Level (Warning):** Log event, notify the user.
- **2nd Level (SOC Alert):** Send alert to security team if risk score **≥10**.
- **3rd Level (Quarantine):** Automatically lock user session if behavior is extreme.

✅ Dynamic Trust Levels & Adaptive Authentication

- Ask for an additional **session-generated credential** if flagged.
- Implement **TOTP (Time-based One-Time Password) verification** for flagged users.

✅ Threat Intelligence Updates & Automated Countermeasures

- **Block user's IP** dynamically if linked to known threats.
- **Disable account access** if **multiple high-risk behaviors occur in sequence**.

✅ Automated Threat Response via API Hooks

- Trigger **MSS screenshots** when a suspicious command is executed.
- Capture **active window metadata** for forensic analysis.
- If a flagged IP is detected: **Take a screenshot & notify SOC**.

---

5. RECOVER (Post-Incident Analysis, Learning & Refinement)

Goal: Improve security resilience and refine threat detection over time.

🔹 Key Features & Implementation

✅ Log Review & Forensics

- Store **encrypted keystroke logs** for **post-incident investigation**.
- Generate **detailed security reports** for SOC review.

✅ Adaptive Whitelisting & False Positive Reduction

- If flagged behavior is verified as safe, add it to the whitelist.
- Implement **context-aware whitelisting**:
  - Exempt security admins from alerts for running `net user`.
  - Ignore `ping google.com` but flag `ping <company_internal_IP>`.

✅ Machine Learning for Continuous Improvement

- Implement **anomaly detection algorithms**:
  - Compare **current user behavior to past trends**.
  - Retrain model periodically with **new behavior patterns**.

✅ Purple Teaming & Penetration Testing

- **Simulated attack scenarios** to test effectiveness.
- Perform **SOC drills** using red team tactics against "Sneaky".

✅ Incident Response Drills & Playbooks

- Create **incident playbooks** for responding to flagged behavior.
- Train **security teams** on identifying & mitigating threats using "Sneaky".