

**ADVANCED KEYLOGGER DETECTION AND DEFENCE SYSTEM:
BEHAVIORAL SPYWARE ANALYSIS USING MACHINE LEARNING****Jyotirmay Karemore¹, Saurabh Udapure², Prof. Rahul Lilhare³**^{1,2}PG Scholar, ³Assistant Professor

Department of Computer Application

K.D.K.College of Engineering, Nagpur, Maharashtra, India

jyotirmayakaremore.mca24f@kdkce.edu.in, udapureskishorao.mca24f@kdkce.edu.in,rahul.lilhare@kdkce.edu.in**Abstract**

The rapid evolution of malware, particularly stealthy keyloggers, poses a significant threat to user privacy and data security in personal and enterprise environments. Traditional antivirus solutions rely heavily on signature-based detection, which often fails against custom-made "Zero-Day" spyware that lacks a known digital fingerprint. This paper presents the design and implementation of an Advanced Keylogger Detection and Defence System developed as a hybrid local application. The proposed system integrates kernel-level metric extraction, a Hybrid Logic detection engine, and the Isolation Forest Machine Learning algorithm to identify malicious behavior patterns. Uniquely, the system employs a dual-interface architecture: a native **CustomTkinter** desktop controller for real-time threat management and a rich **HTML/CSS** dashboard for detailed behavioral analytics. The application enables users to monitor CPU and network anomalies, receive immediate visual alerts, and terminate threats via a "Kill Switch" interface without relying on external cloud analysis. Experimental evaluation demonstrates high accuracy in detecting high-resource surveillance scripts while maintaining low system overhead.

Index Terms—Keylogger Detection, Behavioral Analysis, Isolation Forest, Machine Learning, Anomaly Detection, Cybersecurity, Real-time Monitoring, Python.

I. Introduction

Keyloggers have evolved from simple executable files to sophisticated, custom-written scripts that run silently in the background, recording keystrokes and exfiltrating sensitive data such as passwords and financial information. Students and professionals often experience difficulty in identifying these threats because they do not manifest as visible windows or distinct icons.

Traditional security methods, such as standard Task Managers, provide raw system data but do not offer real-time risk assessment or warning mechanisms. Although commercial antivirus software exists, most rely on "Signature Matching" databases, which require a virus to be previously known and cataloged. This approach creates a critical vulnerability against new, custom-made Python scripts or "Zero-Day" malware.

The advent of Machine Learning has promised to streamline threat detection, yet many solutions rely on heavy Deep Learning models that require significant computational power. Furthermore, privacy concerns arise when system logs are sent to remote cloud servers for analysis.

To overcome these limitations, this paper proposes an Advanced Keylogger Detection and Defence System designed to protect users from behavioral anomalies. The system provides real-time monitoring, AI-driven threat analysis using the Isolation Forest algorithm, and instant remediation capabilities while operating entirely locally. By eliminating dependency on virus signatures, the system ensures protection against unknown threats. This work addresses a critical gap in the current landscape of personal cybersecurity tools by delivering a lightweight solution that functions seamlessly alongside standard applications.

II. Literature Review and Motivation**A. Signature-Based vs. Behavioral Detection**

Various research studies have focused on malware detection methodologies. Signature-based systems have been shown to be effective against known threats but fail when attackers slightly modify the malware's code to alter its hash. Research demonstrates that behavioral analysis—monitoring what a program does rather than what it is—is essential for detecting modern spyware.

B. Machine Learning in Cybersecurity

Studies on Anomaly Detection indicate that Unsupervised Learning algorithms can effectively identify outliers in system performance. The Isolation Forest algorithm, specifically, isolates anomalies rather than profiling normal data points, making it computationally efficient for real-time applications. Empirical studies corroborate the effectiveness of this approach, demonstrating that it can detect high-resource usage spikes associated with data encryption and exfiltration.

C. Python for System Security

Research on lightweight security tools highlights the effectiveness of Python libraries such as psutil for kernel-level interaction. Python provides a robust ecosystem for integrating system monitoring with Machine Learning (scikit-learn) and web technologies (Flask), enabling the development of sophisticated, modular security applications.

III. Proposed System Architecture and Design

A. System Overview

The Advanced Keylogger Detection and Defence System is a host-based intrusion detection system designed to assist users in identifying unauthorized surveillance activities. The system is architected following a modular Client-Server design pattern running locally, ensuring separation of concerns and maintainability. The system operates entirely on the host machine without requiring external cloud analysis, making it inherently privacy-preserving.

B. System Modules and Functional Components

- **Data Acquisition Module:** This module is the sensory component, utilizing the psutil library to extract kernel-level metrics every second. It monitors CPU Usage (%), RAM Usage (%), and Network Upload/Download Speed (KB/s).
- **Intelligence Module (The Brain):** This module implements the core detection logic using a Hybrid approach. It utilizes a Rule-Based Filter to ignore low-resource processes and an AI Analysis engine (Isolation Forest) to mathematically isolate behavioral anomalies.
- **Presentation Module (Dual-Interface):**
 - **Desktop Control Hub (CustomTkinter):** The primary user interface is a native desktop application built with CustomTkinter. It provides a "Glassmorphism" inspired dark-mode widget that remains always-on-top, offering immediate status indicators (Red/Green) and the critical "Kill Switch" functionality for rapid threat termination.
 - **Analytics Dashboard (HTML/CSS):** For deep-dive analysis, the system renders a local web dashboard. This component utilizes HTML5 and CSS3 to structure and style detailed reports, while JavaScript (Chart.js) visualizes historical system usage trends.

C. System Architecture Layers

The system follows a three-layer architecture:

- **Data Layer:** Handles raw extraction of system metrics from the OS kernel.
- **Logic Layer:** Implements the Hybrid Detection Algorithm and the Isolation Forest model.
- **Presentation Layer:** A hybrid layer combining a **Native GUI** (CustomTkinter) for control and a **Web UI** (HTML/CSS) for visualization.

IV. Methodology and System Development

A. Development Methodology

The system was developed following an iterative prototyping approach. Initial prototypes focused on accurate data extraction using psutil, followed by the incremental addition of the Machine Learning model and the web-based dashboard.

B. Algorithm Implementation (Hybrid Logic)

To ensure high accuracy, we implemented a -step verification logic:

1. **Threshold Gate:** A pre-check filters out processes consuming less than % CPU and KB/s Network. This prevents the AI from analyzing benign, idle applications.
2. **Anomaly Detection:** Only processes passing the gate are fed into the Isolation Forest. This ensures that only significant resource spikes—indicative of active logging or data exfiltration—are flagged.

C. Data Persistence Strategy

The system includes a self-training mechanism. Upon startup, the model generates a synthetic dataset representing "Normal" computer usage (Low CPU, Low Network). It trains the Isolation Forest on this data to establish a baseline for normalcy.

V. Experimental Evaluation and Results

A. Evaluation Methodology

The proposed system was evaluated through functional testing using a custom "Dummy Malware" script (fake_malware.py) designed to mimic keylogger behavior (high CPU usage loop).

B. Results and Analysis

The experimental results indicated:

- **Detection Latency:** The system successfully detected the dummy threat within - seconds of execution.
- **Visual Alert Efficacy:** The dashboard correctly transitioned from a "Secure" (Green) state to a "High Alert" (Red) state immediately upon detection, providing clear visual feedback.
- **Termination Success:** The "Kill Switch" functionality successfully terminated the target process via the OS kernel, returning the system to a secure state within seconds.
- **False Positive Reduction:** The implementation of the Hybrid Logic filter successfully prevented false positives from standard background applications (e.g., web browsers), provided they did not exceed the "Normal" threshold.

VI. Comparative Analysis with Existing Solutions

Table I: Comparative Analysis

Dimension	Proposed System	Standard Antivirus	Task Manager
Interface Strategy	Hybrid (Native + Web)	Native UI	Native UI
Detection Basis	Behavioral (Actions)	Signature (Files)	Manual Monitoring
Zero-Day Protection	High (Anomaly based)	Low (Database based)	None
Real-Time Alert	Desktop Widget Pop-up	Popup (if detected)	None (Passive)
Privacy	Local Processing	Cloud Uploads	Local
Kill Mechanism	One-Click Button	Quarantine/Delete	End Task

Positioning:

The proposed system fills a distinct niche in the cybersecurity landscape: it prioritizes behavioral analysis and user autonomy over static database matching. This positioning makes it particularly suitable for developers and privacy-conscious users who require protection against custom scripts.

VII. Technical Stack and Implementation Details

The application is built using a robust Python stack that leverages both native and web-based rendering for optimal user experience. The technology stack comprises:

- **Python 3.x:** Core programming language chosen for robust system access.
- **CustomTkinter:** Selected as the primary GUI framework. It provides a modern, high-DPI aware interface that manages the application lifecycle and "Kill Switch" mechanisms.
- **Flask:** Lightweight web framework used to serve the detailed analytics dashboard locally.
- **HTML/CSS:** Semantic markup and responsive "Hacker Theme" design used to render complex data visualizations within the dashboard view.
- **Scikit-Learn:** Library used for implementing the Isolation Forest algorithm.
- **Psutil:** Library for cross-platform hardware monitoring.
-

VIII. Limitations and Considerations

A. System Limitations

- **Passive Loggers:** If a keylogger is strictly saving keystrokes to a local text file without using significant CPU or Internet resources, it might evade the high-usage threshold.
- **System Overhead:** The detector itself uses a nominal amount of CPU to monitor other applications, which is an inherent trade-off for real-time security.

B. Future Enhancements

- **Deep Learning Integration:** Future iterations will incorporate Long Short-Term Memory

(LSTM) networks to predict malicious behavior based on sequences of actions rather than instantaneous spikes.

- Forensic Logging: We plan to implement file system monitoring to detect rapid log file creation.
- System Service: Converting the application into a background service (Daemon) to prevent it from being easily closed by malware.

IX. Future Enhancements and Extensions

A. Deep Learning Integration While the current Isolation Forest model is effective for anomaly detection, future iterations will incorporate **Long Short-Term Memory (LSTM)** networks. Unlike the current snapshot-based analysis, LSTMs can analyze the *sequence* of system calls over time, allowing the system to predict malicious behavior patterns before they fully execute.

B. Remote Administration Dashboard Leveraging the existing HTML/CSS analytics module, future work will focus on securely exposing the local dashboard via an encrypted tunnel (e.g., VPN or SSH). This would allow system administrators to remotely monitor the security status of multiple machines from a centralized web interface without compromising user privacy.

C. Enhanced Forensic Logging To provide a complete security audit, the system will be expanded to include **File Integrity Monitoring (FIM)**. By integrating Python's watchdog library, the application will track rapid file creation or modification events—often associated with keyloggers saving stolen data to log files—and correlate them with CPU spikes.

D. Background Service Deployment Currently, the application runs as a user-space program. Future development aims to convert the core detection engine into a **System Service (Daemon)** on Windows and Linux. This ensures the protection mechanism starts automatically at boot and cannot be easily terminated by standard malware attempting to close the application window.

E. Cross-Platform UI Adaptation While the current **CustomTkinter** interface is optimized for Windows, future versions will utilize cross-platform packaging tools to ensure a consistent "Hacker Theme" experience on macOS and Linux distributions, making the tool universally accessible.

X. Conclusion

This paper presented an Advanced Keylogger Detection and Defence System designed to improve personal cybersecurity through behavioral analysis. The system effectively supports real-time monitoring, anomaly detection using Isolation Forest, and immediate threat termination. The modular architecture and Python-based implementation make it suitable for deployment on personal computers. Experimental evaluation demonstrated the system's ability to detect high-resource unauthorized scripts and empower users to neutralize threats instantly. By eliminating dependency on virus signatures, the application democratizes access to advanced security tools for users facing custom or zero-day threats.

References

1. F. T. Liu, K. M. Ting and Z. -H. Zhou, "Isolation Forest," in Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008.
2. G. Rodola, "psutil: process and system utilities," [Online]. Available: <https://github.com/giampaolo/psutil>.
3. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.
 - A. Grinberg, "Flask Web Development: Developing Web Applications with Python," O'Reilly Media, 2018.
4. A. Grinberg, "Flask Web Development: Developing Web Applications with Python," O'Reilly Media, 2018.
5. S. Gupta, "Malware Detection Techniques: A Review," International Journal of Computer Applications, vol. 10, no. 1, 2012.
6. M. E. Kabay, "A Brief History of Computer Viruses and Keyloggers," Journal of Information Security, 2008.
7. IEEE, "IEEE Editorial Style Manual," IEEE Publishing, 2020.
8. T. Schimansky, "CustomTkinter: A modern and customizable python UI-library based on Tkinter," [Online]. Available: <https://github.com/TomSchimansky/CustomTkinter>.