



CALIFORNIA STATE UNIVERSITY, DOMINGUEZ HILLS

Fall 2023

Senior Project Report

Computer Science Department

California State University, **Dominguez Hills**



csc@csudh.edu
<http://csc.csudh.edu>

Computer Science Department
College of Natural and Behavioral Sciences
California State University, Dominguez Hills

NSM A-132
1000 East Victoria
Carson, CA 90747
Ph: (310) 243-3398
fax: (310) 243-3153

Project/Thesis Proposal

Select one: ITC 492 __ CTC492 x CSC 492 __ CSC 590 __ CSC599 __ CYB590 __

Semester: Fall **Year:** 2023

Title: System Monitor

Prepared by: achraf el khatib **Date:** 09/19/2023

Faculty advisor

Signature

Date

Committee member

Signature

Date

Committee member

Signature

Date

Graduate coordinator

Signature

Date

Proposal Description: (one to two pages)

Threat detection/System Monitoring software

The project aims to utilize CPU monitoring for threat detection against malicious processes and activities by continuously monitoring the CPU usage across the system, when anomalies are detected, such processes would get flagged for further investigation, either done manually or by the software. When done by the software it would compare the process in suspicion to a database such as virus total's database for IoC's which could include known malicious IP addresses, malware files, etc.

Another feature of the software would be analyzing the systems network traffic, by displaying current open ports within the system, as well as discovering devices on the same network with their ports and information gathered through the scans such as devices names. It would also include a map that would display the live feed traffic for the system based on IP Geolocation to investigate traffic location for potential man in the middle attacks.

The software would also utilize scripts written to investigate known issues and vulnerabilities within the system; such scripts would run in the background and give the user recommendations on how to harden the device.

In summary the overall aim for the software is to be a one stop dashboard application for an everyday joe to utilize and asses their own systems defense and safety.

System Monitoring Software

Achraf El Khatib

California State University Dominguez Hills

Course Number: CTC492

Mohsen Beheshti

10/10/2023

Abstract

The current world is dominated by technology and the interconnected digital landscape, and the same way as you'd protect anything sensitive to you by placing it in a safe or lock, protecting your digital assets has become equally imperative. In line this project aims to introduce user-friendly Threat Detection and System Monitoring software for your home windows computer. This software serves as a computer monitoring system that employs techniques such as CPU monitoring to detect anomalies indicative of malicious processes. It also analyzes network traffic, identifies open ports, and maps the live feed traffic for potential man-in-the-middle attacks. Furthermore, the software as well proactively engages in script-based investigation to address known vulnerabilities, providing users with tailored recommendations for system fortification. To summarize, this project aims to bring cybersecurity to every home, making it a user-friendly experience, to where it would be accessible for an everyday individual. Empowering individuals to take charge of their digital security and reinforcing the notion that protecting our digital assets is not a luxury but a necessity in our interconnected world.

Table Of Contents

1. Introduction	6
1.1 Introduction	6
1.2 Purpose	7
Democratizing Cybersecurity:.....	7
Comprehensive Threat Detection and Monitoring.....	7
Preventing Action and System Hardening	8
User-Friendly Interface	8
2. Background	8
2.1 Background	8
2.2 CPU Monitoring.....	9
2.3 RAM Monitoring	10
2.4 Network Monitoring	11
3. System Monitor Code	13
3.1 Architecture	13
System Monitor Design Diagram	13
PyQt5-Based User Interface (GUI)	16
Multi-Threading for Real-Time Updates	17
Data Logging and Persistence	17
User Interaction and User Experience	18
3.2 GUI	18

System Information Frame('InfoFrame')	18
User Interaction and Accessibility.....	19
Summary	20
3.3 APIs.....	20
AbuseIPDB API Key.....	20
IP Analysis function	20
Continuous IP Checking Thread (IPCheckThread).....	21
IPCheckThread	22
Signal Emission.....	22
Handling API Limit.....	23
3.4 Sample Performance.....	23
Real-Time System Performance.....	24
4. Conclusion.....	27
4.1 Conclusion.....	27
4.2 Future Plans	28
System Auditing Scripts	28
Automatic Firewall Updates.....	28
System Health Monitoring Scripts	28
Governance, Risk, and Compliance (GRC) Scoring.....	29
In conclusion,	29

References 30

1. Introduction

1.1 Introduction

In an era dominated by technology's pervasive influence, our reliance on digital systems has reached unprecedented levels, permeating every aspect of our lives. The escalating integration of digital platforms into routine tasks, professional endeavors, and entertainment exposes our digital assets to an ever-evolving array of threats. In response to this dynamic landscape, the imperative for robust and vigilant system monitoring becomes paramount to fortify against potential risks.

This software emerges as a loyal companion in the ongoing quest to safeguard digital assets. A user-friendly dashboard lies at its core, harmonizing simplicity with the robust features essential for proactive threat detection and system monitoring. Evolving beyond conventional solutions, this software embodies a dual functionality — adeptly detecting threats while providing a comprehensive overview of system health.

The foundation of our software rests on the vigilant monitoring of CPU usage, serving as a sentinel against anomalies and potential malicious activities. The evolution of cybersecurity demands a more nuanced approach, and this software rises to the challenge by extending its capabilities to real-time network traffic analysis. This expanded functionality not only identifies potential vulnerabilities but also unveils the subtle intricacies indicative of a man-in-the-middle attack.

Embracing a proactive stance, the software goes further by engaging in script-based investigations, empowering users with actionable recommendations to fortify their systems against known vulnerabilities. This strategic amalgamation of threat detection, system monitoring, and actionable insights positions the software as a trailblazer in the realm of cybersecurity.

The significant augmentation to our software lies in the integration of the AbuseIPDB API, providing real-time analysis of IP addresses. This enhances the application's capacity to discern potential security threats, leveraging AbuseIPDB's extensive database. Through continuous IP checking, the

software dynamically interacts with the API, offering users immediate insights into the security status of monitored IP addresses. This not only fortifies the system against potential risks but also exemplifies the software's commitment to staying ahead of emerging threats.

As we unravel the intricacies of this innovative solution, it becomes evident that our software heralds a new era in cybersecurity — one where protection is accessible, intuitive, and empowering for users across all digital domains. The following sections delve into the core functionalities, implementation details, and the impactful integration of the AbuseIPDB API, showcasing how this software stands as a sentinel in the ever-evolving landscape of digital security.

1.2 Purpose

The purpose of the Threat Detection and System Monitoring software is twofold: to provide individuals with a user-friendly experience without the technical expertise yet with a robust tool to fortify their digital assets, and to offer a proactive defense mechanism against cyber threats.

Democratizing Cybersecurity:

In a world where technology prevails over our daily lives and digital threats are present around every corner, democratizing cybersecurity for an everyday joe has become a necessity. The software aims to provide an accessible dashboard, where anyone would be able to utilize it. This democratization is a step towards ensuring that cybersecurity becomes a shared responsibility and empowering all users to take control of their digital assets' safety.

Comprehensive Threat Detection and Monitoring

The primary purpose of this software is to serve as a threat detection and system monitoring system. With the continuous monitoring of the CPU usage for anomalies, the software is able to detect anomalies that aren't visible to the naked eye. As well as it's network traffic analysis capabilities it provides the users with real-time insights into potential vulnerabilities, enhancing their ability to

proactively address and mitigate risks. The software functions as well as encompassing tools, giving users a holistic understanding and view of their system's security posture.

Preventing Action and System Hardening

Detecting threats is one of the functions of the software, but through script-based investigations, the software identifies known vulnerabilities through linked databases with APIs within the system and offers users actionable recommendations for system hardening. This preventive actions empowers users to fortify their system before potential threats can exploit weaknesses within the system. Contributing to a proactive rather than reactive cybersecurity approach.

User-Friendly Interface

By providing the user a friendly interface, the software bridges the gap between sophisticated cybersecurity tools and everyday users. Its simplicity ensures accessibility to the average joe and it's accessibility, while retaining the features needed to navigate the intricacies of cyber threats. The purpose is to make cybersecurity not only effective but also approachable for individuals who may not possess advanced technical skills.

2. Background

2.1 Background

The central processing unit (CPU) within a networked device or computer serves as the backbone, orchestrating a multitude of tasks essential for seamless operations. This electronic workhorse is responsible for executing diverse processes, from running applications to facilitating efficient data exchange. The continuous engagement of the CPU, however, introduces challenges that warrant vigilant monitoring and management.

The analogy of the CPU as a workhorse aptly captures its relentless commitment to executing tasks within a computing environment. Imagine this workhorse managing a diverse array of

responsibilities, from handling complex computations to ensuring the smooth execution of multiple programs concurrently. The constant churn of processes places a significant demand on the CPU, a demand that, if left unchecked, can lead to a cascade of issues detrimental to the health and performance of the entire system.

One critical concern associated with sustained CPU usage is the propensity for overheating. As the CPU tirelessly executes tasks, it generates heat, and excessive heat can compromise the integrity of the CPU and surrounding components. Overheating poses a real threat to the hardware, potentially causing permanent damage and diminishing the overall lifespan of the device (Solarwinds, n.d).

2.2 CPU Monitoring

The relentless engagement of the CPU by various processes can result in a strain that extends beyond thermal concerns. The cumulative effect of continuous CPU usage may lead to a degradation in system performance, manifesting as sluggish responsiveness, delays in task execution, and an overall reduction in operational efficiency.

To mitigate these challenges, effective CPU monitoring emerges as a crucial component of system maintenance and optimization. By closely tracking CPU usage, one can gain insights into the workload demands placed on the CPU, identify potential bottlenecks, and proactively address issues before they escalate. Such monitoring is pivotal not only for preventing hardware damage due to overheating but also for preserving the optimal functionality of the entire computing system.

In the context of our project, which aims to develop a Threat Detection and System Monitoring software, understanding the intricacies of CPU usage becomes foundational. The continuous monitoring of CPU metrics will empower users to detect anomalies, identify potential threats posed by malicious processes, and ensure the overall health and longevity of their systems. As we delve into the design and implementation of our software, the awareness of the CPU's pivotal role underscores the importance of real-time monitoring in enhancing the resilience and performance of networked devices.

2.3 RAM Monitoring

Similar to the CPU, the random-access-memory (RAM) plays an integral role in the functionality and overall health of a networked device or computer. In the intricate dance of system processes, RAM acts as a dynamic workspace, providing swift access to data and programs actively in use. Understanding the significance of RAM is paramount, considering its direct impact on system performance and the prevention of potential anomalies.

In the realm of system architecture, RAM serves as a vital intermediary between the CPU and long-term storage, ensuring rapid data retrieval and efficient execution of tasks. The memory-intensive nature of modern applications, coupled with the demand for concurrent processes, underscores the critical role RAM plays in maintaining a responsive and smoothly functioning system.

The constant utilization of RAM by various processes is not without its challenges. Memory anomalies, deviations from the expected patterns of usage, can signal underlying issues that, if left unaddressed, may compromise system stability and performance. To illustrate, the identification of a memory usage anomaly can be visualized as a red dot amid the broader landscape of memory usage data as showcased within figure 2.1. This anomaly may manifest as unexpected spikes or irregularities, indicating a deviation from the norm (Bikash Agrawal, 2017).

Accurate detection of memory anomalies involves deciphering the intricate patterns within memory usage data. The interplay of elements, represented by the black line (original data, X), orange line (random noise, E), and blue line (low-rank signal, L), forms a complex tapestry that necessitates sophisticated monitoring and analytical tools. By pinpointing anomalies within this intricate data landscape, users can preemptively address potential threats to system stability and security.

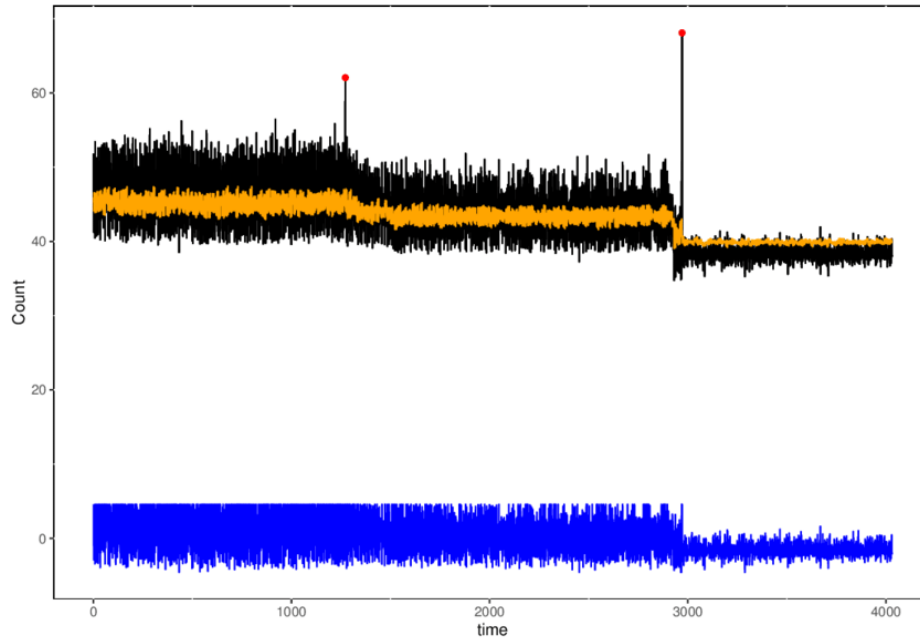


Figure 2.1: Memory Usage Graph

As we embark on the development of our Threat Detection and System Monitoring software, the nuanced understanding of RAM's pivotal role becomes foundational. Continuous monitoring of memory usage allows for the early detection of anomalies, enabling users to investigate and resolve potential issues before they escalate. This proactive approach not only ensures the robustness of individual devices but also contributes to the overall resilience of interconnected networks in the face of emerging cybersecurity threats.

The incorporation of memory anomaly detection within our software aligns with the broader goal of empowering users to comprehensively monitor and fortify their systems against potential vulnerabilities, preserving the integrity and optimal functionality of their digital assets.

2.4 Network Monitoring

In the intricate landscape of networked environments, monitoring and safeguarding the flow of data and communication is paramount. Network monitoring, a critical facet of system administration and cybersecurity, involves the continuous surveillance of data packets, traffic patterns, and the myriad

interactions within a network. Understanding the complexities of network monitoring is essential for identifying anomalies, preempting potential threats, and fortifying the digital perimeters against cybersecurity risks.

Networks, whether local or expansive, serve as conduits for the exchange of information and resources. The efficient functioning of networks is contingent on a delicate balance, wherein various devices communicate seamlessly, and data traverses the network with minimal latency. However, this dynamic environment is susceptible to anomalies, deviations from established patterns of network behavior that may signify potential security threats.

One powerful method employed in network monitoring is anomaly-based detection. This approach allows network administrators to classify behavior as either "good" or "normal" and, conversely, as "suspicious." By leveraging anomaly-based detection, alerts can be triggered when specific activities deviate from established norms within network traffic (ManageEngine, n.d).

Anomaly-based detection serves as a proactive defense mechanism against a spectrum of potential threats that might elude traditional signature-based detection methods. This approach is particularly adept at identifying anomalies that do not align with baseline behavior, such as unusual login patterns, the addition of rogue devices to the network, or a sudden surge of connection requests. Zero-day intrusions, characterized by novel attack vectors that lack predefined signatures, can be promptly detected and flagged, bolstering the security posture of the network (ManageEngine, n.d).

As we embark on the development of our Threat Detection and System Monitoring software, the inclusion of robust network monitoring capabilities becomes pivotal. The software's network traffic analysis component, coupled with anomaly-based detection, empowers users to scrutinize live feeds, identify potential threats, and receive timely alerts on activities deviating from established network behavior. By integrating anomaly-based detection, our software aligns with contemporary cybersecurity practices, ensuring the immediate detection and mitigation of emerging threats to network security.

In essence, network monitoring within our software encapsulates a holistic approach to cybersecurity, providing users with the tools to fortify their networks against both known and unforeseen threats. As we delve into the intricacies of network monitoring, we acknowledge the imperative role it plays in the broader mission of enhancing the resilience and security of interconnected digital ecosystems.

3. System Monitor Code

3.1 Architecture

System Monitor Design Diagram

1. Main Application (SystemMonitor):

- Responsible for initializing and managing the entire application.
- Creates the main window for the user interface.
- Manages two primary threads for real-time updates: **UpdateThread** and **TrafficThread**.

2. User Interface (PyQt5 GUI):

- Main Window (QMainWindow):
 - Provides the core structure for the graphical interface.
 - Contains the System Information Frame and Network Information Frame.
- System Information Frame (InfoFrame):
 - Positioned to the left of the main window.
 - Displays real-time system information using a **QTreeWidget**.
 - Utilizes the **set_info** method for dynamic updates of CPU and RAM usage.
- Network Information Frame (TrafficWidget):
 - Positioned to the right of the main window.
 - Displays real-time network traffic information using a **QTreeWidget**.

3. Multi-Threading for Real-Time Updates:

- **UpdateThread:**
 - Continuously collects and updates real-time system performance data.
 - Retrieves CPU usage, RAM usage, network data transmission rates, and other metrics.
 - Emits signals for updating the System Information Frame in real-time.
- **TrafficThread:**
 - Monitors network traffic continuously.
 - Collects data on network connections, including connection types and addresses.
 - Saves network traffic data to a CSV file and emits signals for updating the Network Information Frame.

4. **Data Logging and Persistence:**

- Network traffic data is logged and stored in a CSV file for historical reference.
- Unique IP addresses are stored in a separate file for reference and analysis.

5. **User Interaction and User Experience:**

- The graphical user interface (GUI) is designed to be user-friendly and visually engaging.
- System performance metrics are presented in a clear and accessible manner.
- Real-time updates and smooth transitions within the GUI enhance the user experience.

This textual design diagram provides an overview of the major components and their interactions within the System Monitor code. It illustrates how the application's architecture facilitates real-time monitoring of system performance and network activity while maintaining historical records for analysis.

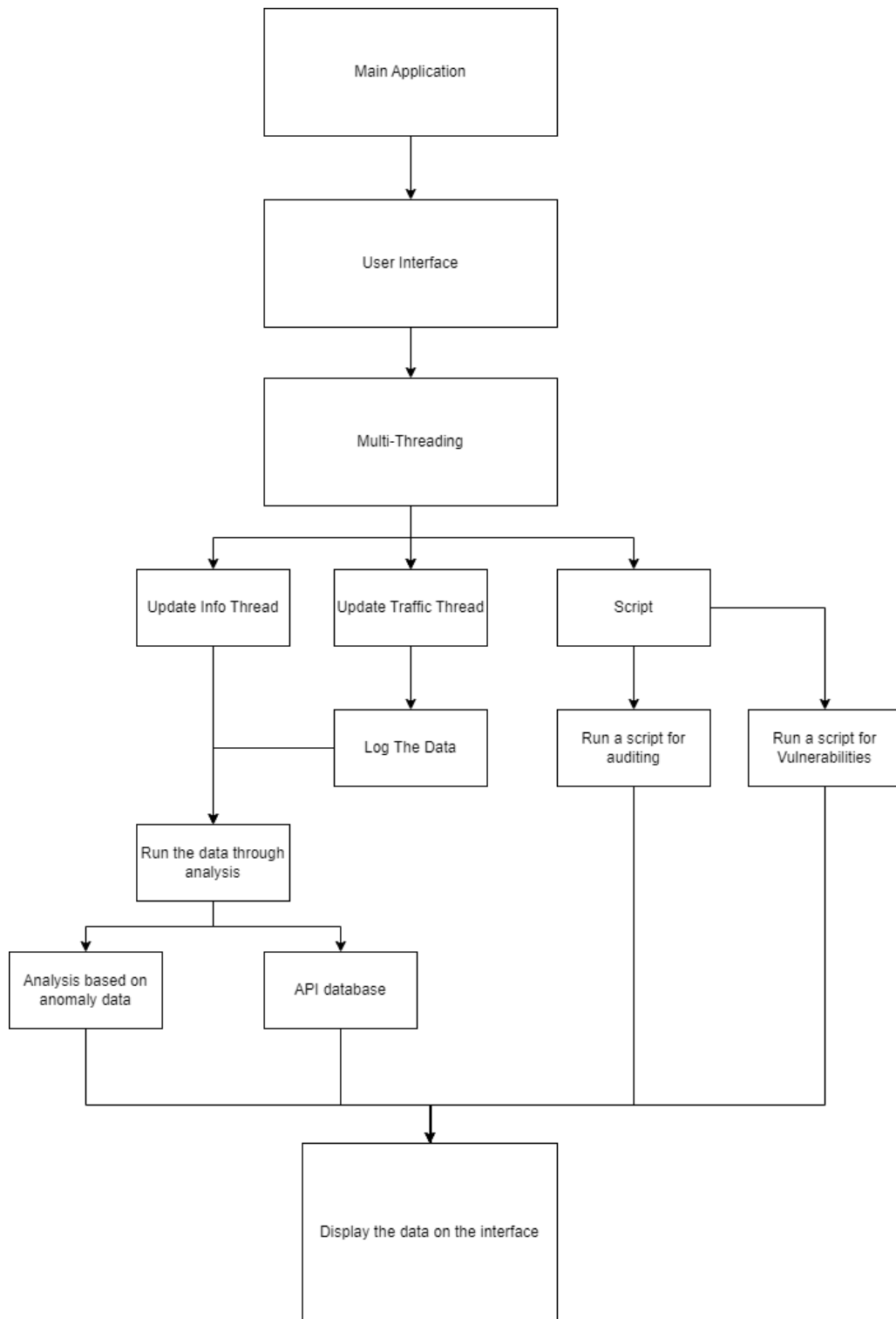


Figure 3.1a: System Monitor Diagram

PyQt5-Based User Interface (GUI)

At the forefront of the System Monitor is a user-friendly graphical user interface (GUI) developed using the PyQt5 library. The GUI serves as the primary interaction point for users and provides real-time insights into system performance. Key elements of the GUI architecture include:

- **Main Window:** The application's main window, created using **QMainWindow**, houses the central components of the GUI.

```
class SystemMonitor(QMainWindow):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("System Monitor")
        self.setGeometry(100, 100, 1920, 1080)

        system_info_group = QGroupBox("System Information")
        system_info_layout = QVBoxLayout()

        self.system_info = InfoFrame()
        system_info_layout.addWidget(self.system_info)

        system_info_group.setLayout(system_info_layout)
        system_info_group.setFixedWidth(400)

        network_info_group = QGroupBox("Network Information")
        network_info_layout = QVBoxLayout()

        self.traffic_widget = TrafficWidget()
        network_info_layout.addWidget(self.traffic_widget)

        network_info_group.setLayout(network_info_layout)

        self.setCentralWidget(QWidget())
        layout = QVBoxLayout(self.centralWidget())
        layout.addWidget(system_info_group)
        layout.addWidget(network_info_group)

        # Specify the path for the CSV file
        csv_file = "NetworkTrafficLog.csv"
        unique_ip_file = "uniqueIpAddress.csv"

        self.update_thread = UpdateThread()
        self.update_thread.updated.connect(self.system_info.set_info)

        self.update_thread.start()

        self.traffic_thread = TrafficThread(csv_file, unique_ip_file)
        self.traffic_thread.updated.connect(self.traffic_widget.set_traffic_info)
        self.traffic_thread.start()
```

Figure 3.1b: QMainWindow code

- **System Information Frame (InfoFrame):** Positioned at the top-left corner of the application window, the '**InfoFrame**' is a labeled container presenting crucial system information. It utilizes a **QTreeWidget** to organize and display data in a tabular format. Real-time updates of CPU and RAM information are achieved through the **set_info** method, ensuring that users have immediate access to system performance metrics.
- **Traffic Widget:** The **TrafficWidget**, a crucial part of the updated GUI, offers real-time insights into network traffic. It utilizes a **QTreeWidget** to present network connection details, including connection type, local address, and remote address.

Multi-Threading for Real-Time Updates

Efficient real-time data collection and updates are achieved using multi-threading. Two vital threads, **UpdateThread** and **TrafficThread**, handle the collection and presentation of data. The multi-threading architecture comprises the following components:

- **UpdateThread:** This thread continuously collects and updates real-time system performance data. It retrieves CPU usage, network data transmission rates, and other crucial metrics using the **psutil** library. The collected data is emitted as signals and displayed in the GUI in real-time.
- **TrafficThread:** Responsible for monitoring network traffic, the **TrafficThread** continuously collects information about network connections. It records details such as connection types, local and remote addresses, and distinguishes between TCP and UDP connections. The collected network data is presented in the GUI, enabling users to monitor active network connections.

Data Logging and Persistence

The System Monitor incorporates data logging and persistence features to maintain historical records of network connections and unique IP addresses. The network traffic data is saved to a CSV file,

allowing users to track historical network connections. Additionally, unique IP addresses are stored in a separate file for reference. This data persistence ensures that historical network traffic information is accessible for analysis and reference.

User Interaction and User Experience

The architecture of the System Monitor places a strong emphasis on user-friendliness and a seamless user experience. The GUI design strikes a balance between simplicity and functionality, making it accessible to users with varying levels of technical expertise. The intuitive visualization of system performance metrics, such as CPU and RAM usage, enhances user engagement and provides a visually enriching experience.

In summary, the architecture of the System Monitor code combines an intuitive PyQt5-based GUI, multi-threading for real-time updates, data logging and persistence, and a user-centric design. This holistic approach offers users a powerful and accessible tool for monitoring and understanding their system's performance and network activity in real-time.

3.2 GUI

The graphical interface of the System Monitor has been designed to offer both functionality and user-friendly aesthetics. Developed using the PyQt5 library, the main window of the application is structured to provide an intuitive experience for the users of varying technical backgrounds. The following components constitute the core elements of the graphical interface:

System Information Frame('InfoFrame')

- Positioned at the top-left corner of the application window, the 'InfoFrame' serves as a labeled container for presenting essential system information.
- The 'InfoFrame' incorporates a TreeWidget, facilitating the organized display of data in a tabular format.

- Dynamic updates to CPU and RAM information are achieved through the `set_info` method, ensuring real-time representation of system performance.
- The `add_info` method seamlessly incorporates additional data into the tabular display, enhancing the comprehensiveness of the presented information.

```
class InfoFrame(QTreeWidget):
    def __init__(self):
        super().__init__()

        self.setHeaderLabels(["Info", "Value"])
        self.setColumnWidth(0, 200)
        self.setColumnWidth(1, 100)

    def set_info(self, values):
        self.clear_info()

        for key, value in values.items():
            parent = QTreeWidgetItem(self)
            parent.setText(0, key)
            parent.setText(1, str(value))

    def clear_info(self):
        for i in reversed(range(self.topLevelItemCount())):
            self.takeTopLevelItem(i)
```

Figure 3.1c: InfoFrame Code

User Interaction and Accessibility

- The overall GUI design prioritizes user-friendliness, presenting intricate system metrics in an accessible manner.
- The interface strikes a delicate balance between simplicity and functionality, catering to a diverse user base with varying levels of technical expertise.
- The seamless updates and smooth transitions within the gauges enhance user engagement, providing a visually enriching experience.

Summary

In summary, the graphical interface of the System Monitor not only facilitates the comprehensive monitoring of system metrics but also prioritizes user accessibility through thoughtful design choices and dynamic visual representations. This synthesis of functionality and user-centric design aligns with the overarching goal of democratizing cybersecurity, making robust system monitoring accessible to users across different proficiency levels.

3.3 APIs

The System Monitor software incorporates AbuseIPDB's API to analyze the IP addresses collected and stored within the CSV files and identify potential threats. The integration enhances the system's ability to assess network security and identify abusive IP addresses and behavior.

AbuseIPDB API Key

The application utilizes an API Key that provides access to the AbuseIPDB services in a secure matter. The key is stored in a configuration file called '**config.ini**' to ensure easy management and security, the API key would have to be provided by the user.

```
config = configparser.ConfigParser()  
config.read('config.ini')  
  
api_key = config['API']['abuseipdb_key']
```

Figure 3.3a: API configuration

IP Analysis function

The '**check_abuseipdb**' function sends requests to the AbuseIPDB API to retrieve information about a given IP address. It provides details such as whether the IP is whitelisted, the abuse confidence score, and the country associated with the IP.

```

def check_abuseipdb(ip_address, max_age_in_days=30, verbose=False):
    base_url = 'https://api.abuseipdb.com/api/v2/check'
    query_params = {
        'ipAddress': ip_address,
        'maxAgeInDays': max_age_in_days,
        'verbose': verbose,
    }
    headers = {
        'Accept': 'application/json',
        'Key': api_key,
    }

    try:
        response = requests.get(base_url, params=query_params, headers=headers)
        response.raise_for_status()

        decoded_response = response.json()
        data = decoded_response.get('data', {})
        is_whitelisted = data.get('isWhitelisted', None)
        abuse_confidence_score = data.get('abuseConfidenceScore', None)
        country_name = data.get('countryName', {})
        return is_whitelisted, abuse_confidence_score, country_name

    except requests.exceptions.RequestException as e:
        print(f"Request Exception: {e}")
        return None, None, None
    except json.JSONDecodeError as e:
        print(f"JSON Decode Error: {e}")
        return None, None, None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None, None, None

```

Figure 3.3b: check_abuseipdb function

Continuous IP Checking Thread (IPCheckThread)

The program features a dedicated thread, IPCheckThread, which continuously monitors unique IP addresses collected during network activities. For each identified IP address, the thread interfaces with AbuseIPDB to retrieve real-time abuse-related information, emitting signals for immediate integration into the application.

```

def check_ip(self, ip_address):
    # Check if the IP has already been checked
    if not self.is_ip_checked(ip_address):
        print(f"IP {ip_address} not checked yet. Checking AbuseIPDB.")

        # Check if the API limit is reached
        if not self.is_api_limit_reached():
            is_whitelisted, abuse_confidence_score, country_name = check_abuseipdb(ip_address, self.max_age_in_days)
            self.update_checked_ips(ip_address, is_whitelisted, abuse_confidence_score, country_name)
            self.ip_checked.emit(ip_address, is_whitelisted if is_whitelisted is not None else False,
                                abuse_confidence_score, str(country_name))
        else:
            print("API limit reached. Skipping IP check.")

```

Figure 3.3c: check_ip definition

IPCheckThread

A dedicated thread '**IPCheckThread**' continuously monitors unique IP addresses collected during network activity. For each IP address, it checks AbuseIPDB for abuse-related information and emits signals for further processing.

```

class IPCheckThread(QThread):
    ip_checked = pyqtSignal(str, bool, int, str)

    def __init__(self, max_age_in_days=30, api_call_limit=100, api_call_reset_duration=timedelta(minutes=5), parent=None):
        super(IPCheckThread, self).__init__(parent)
        self.max_age_in_days = max_age_in_days
        self.api_call_limit = api_call_limit
        self.api_call_reset_duration = api_call_reset_duration
        self.last_api_call_time = None
        self._is_running = True

```

Figure 3.3d: IPCheckThread

Signal Emission

The results of the IP analysis, including whether the IP is whitelisted, the abuse confidence score, and the country, are emitted through signals. These signals enable real-time updates in the application interface.


```

def check_ip(self, ip_address):
    # Check if the IP has already been checked
    if not self.is_ip_checked(ip_address):
        print(f"IP {ip_address} not checked yet. Checking AbuseIPDB.")

        # Check if the API limit is reached
        if not self.is_api_limit_reached():
            is_whitelisted, abuse_confidence_score, country_name = check_abuseipdb(ip_address, self.max_age_in_days)
            self.update_checked_ips(ip_address, is_whitelisted, abuse_confidence_score, country_name)
            self.ip_checked.emit(ip_address, is_whitelisted if is_whitelisted is not None else False,
                                abuse_confidence_score, str(country_name))
        else:
            print("API limit reached. Skipping IP check.")

```

Figure 3.3e: check_ip definition

Handling API Limit

The application intelligently handles AbuseIPDB API limits to avoid exceeding the allowed number of requests. It checks the API call limit and waits for the reset duration before making additional requests.

```

def is_api_limit_reached(self):
    # Check if the API call limit is reached
    base_url = 'https://api.abuseipdb.com/api/v2/check'
    query_params = {
        'ipAddress': '8.8.8.8',
        'maxAgeInDays': self.max_age_in_days,
        'verbose': False,
    }
    headers = {
        'Accept': 'application/json',
        'Key': api_key,
    }

    try:
        response = requests.get(base_url, params=query_params, headers=headers)
        return response.status_code == 429
    except requests.exceptions.RequestException as e:
        print(f"Request Exception: {e}")
        return True

```

Figure 3.3f: API limit check

3.4 Sample Performance

The effectiveness of the System Monitor software is underscored by its robust performance in real-world scenarios.

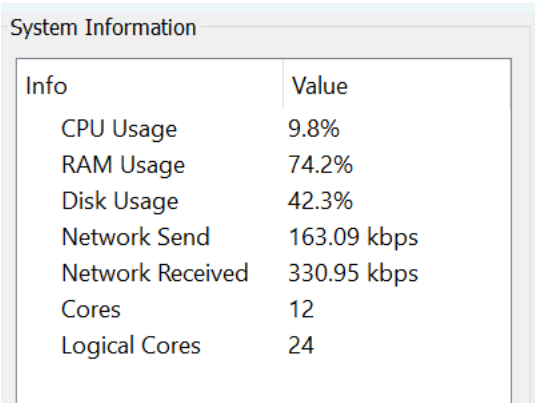
Real-Time System Performance

CPU Usage Monitoring

The System Monitor excels in real-time CPU usage monitoring, providing users with immediate visibility into processor utilization. Users can accurately gauge performance fluctuations and identify resource-intensive processes promptly.

RAM Utilization

Effortlessly tracking RAM utilization in real-time, the software ensures that users stay informed about memory usage patterns. This feature aids in proactive system management, preventing memory-related bottlenecks.

A screenshot of a software window titled "System Information". Inside the window is a table with two columns: "Info" and "Value". The table lists several system metrics and their current values.

Info	Value
CPU Usage	9.8%
RAM Usage	74.2%
Disk Usage	42.3%
Network Send	163.09 kbps
Network Received	330.95 kbps
Cores	12
Logical Cores	24

Figure 3.4a: System Information

Network Traffic Insights

The software's ability to capture and present real-time network traffic information is a key performance highlight. Users can visualize network activities, including data transmission rates, connection types, and addresses, enabling them to detect anomalies and potential security threats swiftly.

Vulnerability Identification

Leveraging script-based investigations, the System Monitor identifies known vulnerabilities within the system. By referencing linked databases with APIs, the software offers users actionable recommendations for system hardening, mitigating potential risks before they materialize.

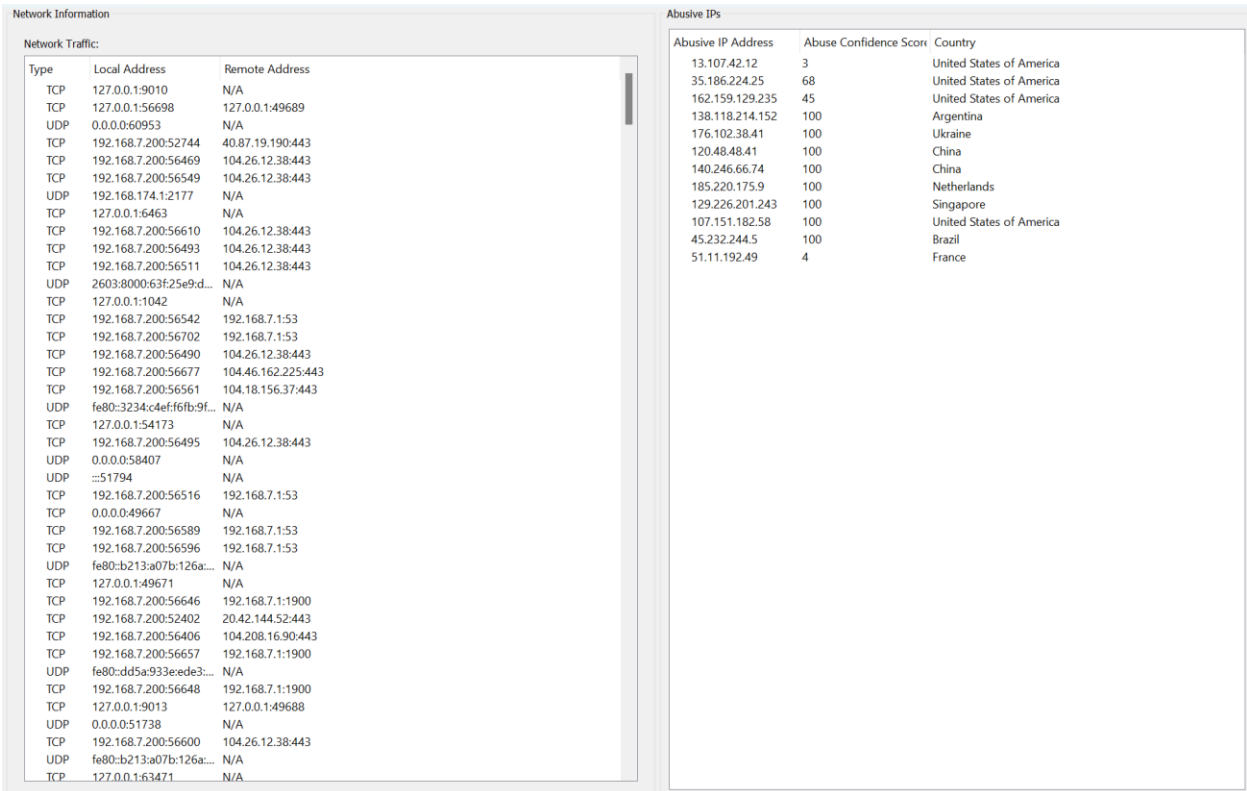


Figure 3.4b: Network Information

Access and Navigation

The graphical user interface (GUI) stands out for its accessibility and ease of navigation. Users, regardless of their technical expertise, can effortlessly access and interpret system performance metrics. The intuitive design ensures a positive user experience.

Real-Time Updates

System performance metrics are presented in real-time, allowing users to stay informed about the dynamic state of their system. The seamless integration of real-time updates enhances user engagement and responsiveness.

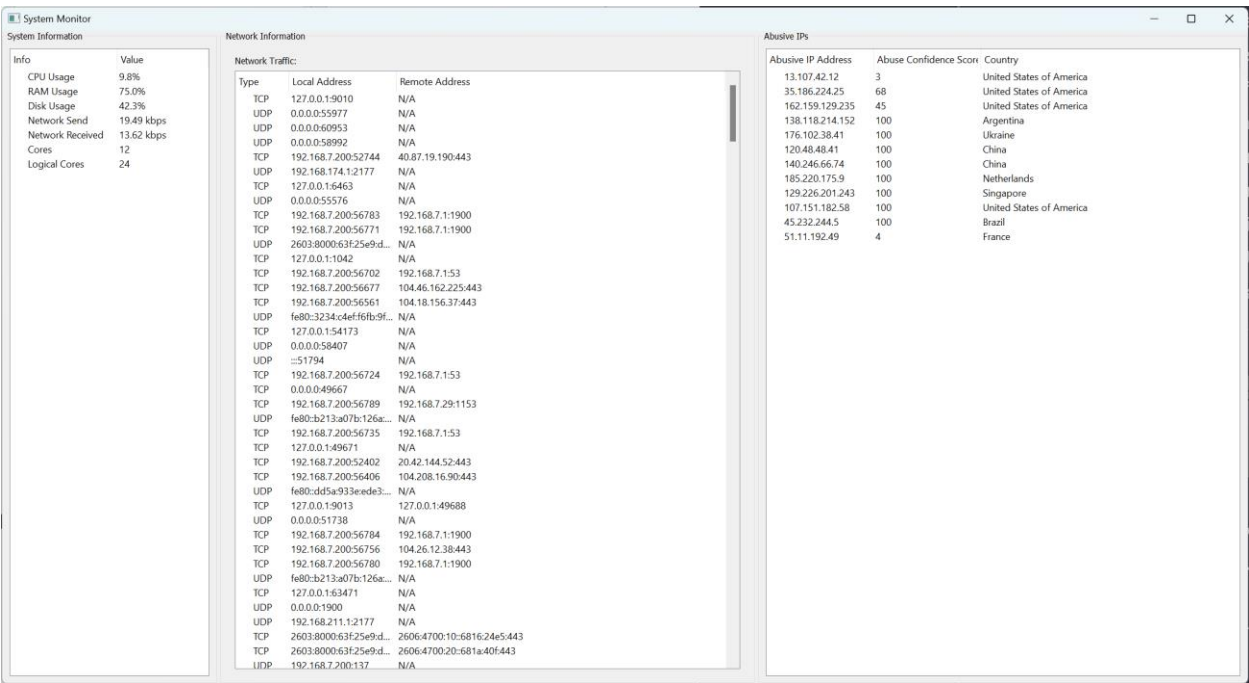


Figure 3.4c: GUI software

Network Traffic Data Logging

The System Monitor excels in logging network traffic data, providing a historical record for analysis and reference. This feature enables users to track changes in network connections over time, facilitating a comprehensive understanding of system behavior.

Unique IP Address Storage

Unique IP addresses are stored separately, contributing to historical reference and analysis. This data persistence ensures that users have access to valuable information for evaluating patterns and trends.





-  abusiveIPs
-  checkedIPs
-  NetworkTrafficLog
-  uniqueIPAddress

Figure 3.4d: CSV databases

4. Conclusion

4.1 Conclusion

In a digital landscape where threats evolve incessantly, the System Monitor stands as a beacon of proactive defense and user empowerment. Our journey through the intricacies of this software reveals not just a tool for monitoring but a holistic solution that caters to users across the technological spectrum.

The software's dual functionality, seamlessly integrating threat detection with real-time system monitoring, mirrors the complexity of contemporary cyber threats. By continuously monitoring CPU usage, tracking RAM utilization, and dissecting network traffic, it places users at the forefront of their digital defense. The real-time insights provided empower users to preemptively address vulnerabilities, thwart potential threats, and fortify their digital assets.

The democratization of cybersecurity, a core philosophy embedded in the software, brings a paradigm shift. Breaking down the barriers of technical complexity, the System Monitor transforms cybersecurity into a shared responsibility. Its user-friendly interface bridges the gap, making intricate cybersecurity tools accessible to every user, irrespective of their technical proficiency.

The performance analysis presented underscores the software's efficacy. From real-time system performance metrics to script-based investigations for vulnerability identification, the System Monitor proves itself as a versatile and robust ally in the realm of cybersecurity. Its commitment to data logging and historical reference ensures that users not only confront current threats but also learn from the past, evolving their defense strategies.

As we conclude this exploration, the System Monitor emerges not just as software but as a testament to the evolving nature of cybersecurity. It heralds an era where users, armed with knowledge and intuitive tools, actively participate in safeguarding their digital realms. In the face of ever-changing

threats, the System Monitor stands as a sentinel, a guardian that empowers users to navigate the complexities of the digital landscape with resilience and confidence.

4.2 Future Plans

As we conclude this exploration, the System Monitor emerges not just as software but as a testament to the evolving nature of cybersecurity. It heralds an era where users, armed with knowledge and intuitive tools, actively participate in safeguarding their digital realms. In the face of ever-changing threats, the System Monitor stands as a sentinel, a guardian that empowers users to navigate the complexities of the digital landscape with resilience and confidence.

System Auditing Scripts

Future iterations will introduce a robust framework of system auditing scripts. These scripts will delve into the intricacies of the system, meticulously examining configurations, permissions, and potential vulnerabilities. Through systematic audits, users will gain granular insights into the security posture of their systems. The auditing scripts will go beyond conventional threat detection, offering a proactive stance against emerging risks.

Automatic Firewall Updates

Automation takes center stage with the implementation of automatic firewall updates. The System Monitor will leverage real-time threat intelligence to dynamically adjust firewall rules. This adaptive approach ensures that the firewall evolves in tandem with the ever-changing threat landscape. By automating this process, users are relieved of the burden of manual firewall management, enhancing the overall resilience of their digital infrastructure.

System Health Monitoring Scripts

Expanding beyond performance metrics, the System Monitor will incorporate specialized scripts for monitoring system health. These scripts will encompass a holistic view of system components, including hardware health, disk integrity, and critical system services. Users will receive comprehensive

reports on the overall health of their systems, enabling them to identify and address potential issues before they escalate.

Governance, Risk, and Compliance (GRC) Scoring

The introduction of GRC scoring brings a strategic dimension to cybersecurity. The System Monitor will evaluate the system's adherence to governance policies, assess potential risks, and provide a compliance score. Users can leverage GRC scoring to align their digital practices with industry standards and regulatory requirements, ensuring a proactive and preemptive approach to cybersecurity governance.

In conclusion,

The future of the System Monitor unfolds as a dynamic journey, where each expansion is a response to the evolving nature of digital threats. By weaving together system auditing, automated firewall updates, system health monitoring scripts, and GRC scoring, the System Monitor aspires to be not just a tool but a strategic partner in the ongoing battle for digital security. Users can anticipate a continually evolving digital defense ally that adapts, augments, and empowers in the face of an ever-changing cybersecurity landscape.

References

What is CPU usage? - it glossary. SolarWinds. (n.d.). <https://www.solarwinds.com/resources/it-glossary/what-is-cpu>

Agrawal, B., Wiktorski, T., & Rong, C. (2017, August). Adaptive real-time ANOMALY DETECTION in cloud infrastructures. https://www.researchgate.net/publication/318912328_Adaptive_real-time_anomaly_detection_in_cloud_infrastructures

Network Anomaly Traffic Detection. Network Traffic Anomaly Detection | ManageEngine NetFlow Analyzer. (n.d.). <https://www.manageengine.com/products/netflow/network-traffic-anomaly-detection.html>

Sys - system-specific parameters and functions. Python documentation. (n.d.). <https://docs.python.org/3/library/sys.html>

Psutil. PyPI. (n.d.). <https://pypi.org/project/psutil/>

Socket - low-level networking interface. Python documentation. (n.d.-a). <https://docs.python.org/3/library/socket.html>

CSV - csv file reading and writing. Python documentation. (n.d.-a). <https://docs.python.org/3/library/csv.html>

OS - miscellaneous operating system interfaces. Python documentation. (n.d.-b). <https://docs.python.org/3/library/os.html>

Requests. PyPI. (n.d.-b). <https://pypi.org/project/requests/>

JSON - JSON encoder and decoder. Python documentation. (n.d.-b).

<https://docs.python.org/3/library/json.html>

Configparser - configuration file parser. Python documentation. (n.d.-a).

<https://docs.python.org/3/library/configparser.html>

Datetime - basic date and time types. Python documentation. (n.d.-c).

<https://docs.python.org/3/library/datetime.html>

Qt for python#. Qt for Python. (n.d.). <https://doc.qt.io/qtforpython-6/index.html>

AbuseIPDB. (n.d.). *Introduction.* AbuseIPDB APIv2 Documentation.

<https://docs.abuseipdb.com/#introduction>

Appendix



Overview

- Introduction
- Goal
- Motivation
- Contribution
 - Implementation
 - Logging
 - AbuseIPDB API
 - Error Handling
- Background
- Design
- Screenshots of the working software
- Budget/Cost
- Timetable
- Conclusion

Introduction

- In an era where technology permeates every facet of our daily lives — from routine chores to professional environments and entertainment — our reliance on digital systems is more pronounced than ever. As our digital footprint expands, the security of our digital assets takes center stage. In this dynamic landscape, where new threats continually emerge, staying vigilant and consistently monitoring our systems become imperative for safeguarding against potential risks.

Goal

- Making security monitoring for windows devices easier to understand for everyone regardless of the their knowledge in cybersecurity

Motivation

- Cybersecurity is a growing concern
- Everyone needs to have some sort of knowledge on how to protect themselves in the digital world
- The main place you should be monitoring for threats is your main system as it's the system you spend the most amount of time on.

Contribution

- Development of System Monitor software.
- Real-time monitoring and data analysis.
- Enhancements to user experience.
- Documentation and presentation support.
- Implementing API calls with AbuseIPDB

Implementation

- Utilization of PyQt5, psutil, and socket libraries.
- Concurrent data collection using multi-threading.
- User-friendly GUI design for data presentation.
- Real-time anomaly detection and network data storage.
- Extensive testing and documentation for project sustainability.

Logging

- Comprehensive recording of network traffic data.
- Efficient storage and easy data accessibility through CSV files.
- Enhanced anomaly detection and security threat identification.
- Maintained records of unique IP addresses for network analysis.
- Established a data retention policy for efficient data management.

