HYBRID MALWARE SCANNER FOR APPLICATION LAYER SECURITY

BY
JOSHUA RUSSELL

22nd April 2024

Dr Anatolij Bezemskij

&

Prof Manos Panaousis

A dissertation submitted in partial fulfilment of the University of Greenwich Degree:

BSc (Hons) Cyber Security and Digital Forensics

Word Count:11,647

Table of Contents

MALWARE SCANNER FOR APPLICATION LAYER SECURITY

Abstract

Signature-based detection relies on predefined patterns or signatures of known threats to identify malicious activities. It compares incoming data or network traffic against a database of signatures, triggering an alert when a match is found. This method is effective against known threats but may miss new or previously unseen attacks.

Anomaly-based detection focuses on identifying deviations from normal behaviour within a system or network. It establishes a baseline of normal activities and raises alerts when it detects unusual or suspicious behaviour. While effective at detecting novel threats, it can also produce false positives if the baseline is not accurately defined.

The application layer is the top layer of the Open Systems Interconnection (OSI) model(a conceptual model which describes how computers communicate over a network), and is responsible for managing communication between software applications and the network. It encompasses protocols and services that directly support application-level functions, such as HTTP for web browsing, SMTP for email communication, and FTP for file transfer. The application layer ensures data integrity, security, and proper formatting for end-user applications.

The aim of my project was to design a security system which used signature-based and anomaly-based intrusion detection techniques to protect against threats against the application layer. There are issues with both forms of detection individually and combining the two techniques' methods into one hybrid program can improve the accuracy and range of the whole program, improving the security of the device running it. The program focuses specifically on attacks that impact the application layer due to its vulnerabilities.

Research has indicated that signature-based detection systems have certain issues relating to out-of-date datasets and they can struggle with detecting threats outside of the dataset. However, they are efficient in quickly detecting threats which are within its scope. Anomaly-based detection meanwhile has the issue of potentially generating false positives within a system. Despite this, it is exceptionally good at detecting never-before-seen attacks, known as zero-days. In addition, the application layer has the widest attack vector due to being the one which the user directly interacts with. This is why the program focuses on a specific group of attacks, as these target the most vulnerable part of the computer. This new system will combine the two detection methods to combat their respective issues and improve the overall accuracy of the threat detection process.

This report will consist of: an overview of the project, the literature review, details relating to the requirements, design, implementation, testing and finally, an evaluation of the program that I have created.

Keywords: Signature-Based Detection, Anomaly-Based Detection, Application Layer, Hybrid

Acknowledgements

I would like to acknowledge the assistance I have received from my tutors, Anatolij and Manos who have supported me in developing the project and improving on my original project concept. I would also like to thank my parents for their unwavering supporting throughout my life and specifically during this university period.

List of Tables used

List of Figures used

1) Introduction

1.1)Background

According to the Cyber Security Breaches Survey 2023, published by the UK's Department for Digital, Culture, Media & Sport in April 2023, across all UK businesses, there were approximately 2.39 million instances of cyber-crime and approximately 49,000 instances of fraud because of cybercrime in the last 12 months prior to the report release in April 2023. Across charities, there were approximately 785,000 cybercrimes over this period. A total of 11% of businesses and 8% of charities have experienced cybercrime in the last 12 months. (Department for Science, Innovation & Technology, 2023). And this is for official businesses that have their own security. The UK had 4783 cybercrime victims per million internet users in 2022, the world's highest (AAG IT Services, 2024). These stats, as well as those shown in Figure 1 indicate that despite the major advancements made in cybersecurity, there are still issues. Therefore, there needs to be further development in existing security methods. My project is based on one of these methods, an intrusion detection system. This is a device or software application that monitors networks and/or systems for malicious activity or policy violations.

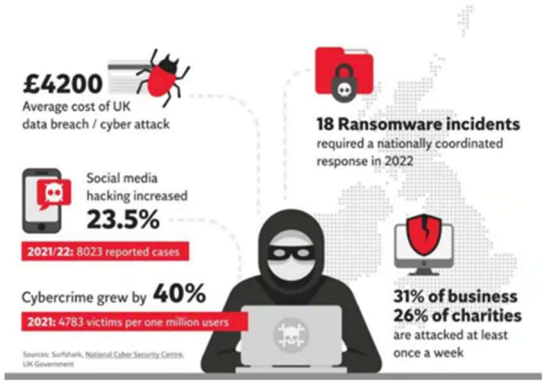


Figure 1- Infographic of Cyber Security Breaches

1.2)Intrusion Detection Systems and Malware Scanners

An intrusion detection system(IDS) is a device or software application that monitors network or host for malicious activity or policy violations. A host-based intrusion detection system (HIDS) can monitor the internals of a computer system as well as the network interfaces. A HIDS will focus on the internal operations of the host itself. It monitors activity such as file system changes, system calls, log entries, and application behaviour to detect any signs of unauthorized access, malware infections, or other suspicious activities (Sysdig, n.d.). A HIDS works by comparing observed activity against a database of known attack signatures or by using behavioural analysis to detect deviations from normal system behaviour. When it identifies potentially malicious activity, it can trigger alerts, log events, or even take automated actions to mitigate the threat, such as blocking network connections or quarantining suspicious files. Overall, HIDSs play a critical role in ensuring the security of individual computer systems by providing real-time monitoring and detection of threats, helping to prevent unauthorized access and protect sensitive data (Heimdal, 2023).

While what I've developed bears similarities to elements found in HIDSs, it doesn't strictly qualify as one for several reasons. Unlike traditional HIDSs, which actively monitor system

activities for suspicious behaviour and intrusions, my implementation primarily focuses on scanning files and processes for known malware signatures and behaviours. While it incorporates features like anomaly detection in running processes, network packet analysis for potential attacks such as Denial of Service (DoS), and keylogger detection, these functionalities are more aligned with malware scanning rather than real-time intrusion detection and prevention. Moreover, while HIDSs typically operate by analysing network traffic and system logs, my system is more akin to antivirus software in its approach, which scans files and processes for known malware patterns. Additionally, aspects of firewall functionality, such as packet sniffing and detection of potential network intrusions, are also integrated. While elements of HIDSs, antivirus software, and firewalls are present, the primary focus remains on malware scanning rather than comprehensive intrusion detection and prevention.

1.3)Intrusion Detection Methods

Signature-based detection is the most usual form of IDS. It involves identifying known patterns, or "signatures," within data or code to detect specific threats or malicious activity. These methods compare observed data or code against a database of predefined signatures, enabling the identification of known threats. It is effective against known and frequent attacks such as malware, phishing, and DoS attacks. It’s also easy to implement and maintain. However, it cannot detect new or unknown vulnerabilities (“zero-days”). Meanwhile, anomaly-based detection identifies deviations from expected behaviour within a system or dataset by comparing observed patterns to established norms. These methods leverage statistical analysis, machine learning algorithms, or heuristic approaches to flag unusual activities indicative of potential threats or abnormalities. It is effective against zero-days or variants on existing attacks and is quite adaptive and dynamic. The issue with this method is that there is a prominent level of false alarms and false positives (LinkedIn, 2023).

1.4)Weakness of Application Layer

The application layer is often considered the most vulnerable in the OSI model, a reference model made to describe network communications, because it directly interacts with end-users and is exposed to various security threats. Its complexity, diverse functionalities, and frequent updates increase the likelihood of coding errors and vulnerabilities. Additionally, user inputs at this layer create opportunities for injection attacks, cross-site scripting, and other forms of manipulation. As applications often rely on external databases and services, attackers may exploit vulnerabilities in these dependencies to compromise the entire system. Apps that heavily rely on web-based interfaces are particularly susceptible to cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks, both of which exploit weaknesses in the application layer.

There have been many examples of attacks on the application layer. In 2011, hackers infiltrated Sony's PlayStation Network (PSN), compromising the personal information,

including usernames, passwords, and credit card details, of over 77 million users. The breach occurred due to vulnerabilities in the PSN's web application infrastructure, allowing attackers to gain unauthorized access to sensitive data (Quinn & Arthur, 2011). In 2020, the University of California, San Francisco (UCSF) experienced a ransomware attack that specifically targeted its School of Medicine's IT systems. The attack resulted in the encryption of critical data, disrupting operations, and threatening the integrity of research and patient care activities. Despite efforts to mitigate the impact, UCSF chose to negotiate with the attackers and paid $1.14 million to regain access to the encrypted data. This incident underscored the vulnerability of academic institutions to cyber threats and highlighted the challenges in balancing data security with the imperative to maintain continuity in essential services and research activities (Tidy, 2020). These attacks demonstrate the vulnerability of the application layer and the need for something to protect it.

1.5)Project Rationale

It has been established that the Signature and Anomaly-Based IDSs currently available each have their own weaknesses. My project seeks to solve these issues by combining signature-based and anomaly-based detection into one program and running both functions simultaneously to detect all issues. The program targets specifically application layer malware such as ransomware and keyloggers due to the host device's vulnerability to this layer.

1.6)Methodology

The Agile methodology is an iterative approach to software development that prioritizes flexibility, collaboration, and customer feedback. It involves breaking down projects into smaller increments called "sprints," allowing for continuous improvement and adaptation to changing requirements throughout the development cycle. The full cycle is shown below in Figure 2.

Figure 2- Infographic of Agile Methodology steps

This was adopted for developing my project, primarily due to its iterative and adaptable nature, which suits the dynamic requirements of cybersecurity research and development. The methodology allows for continuous feedback loops between myself and my supervisors, ensuring alignment with project objectives and academic standards. Additionally, by breaking down the dissertation into smaller, manageable tasks, agile development facilitates a structured approach to research, enabling me to focus on incremental progress while maintaining

flexibility to accommodate any adjustments or new insights that may arise during the project. This approach enhances the efficiency and effectiveness of the research process, even in a solo project scenario.

The program is deliberately written in Python. This is beneficial for several reasons, including its readability (clean, understandable syntax that is easy to understand for people new to Python), its ability to run subfunctions within itself with multithreading and its portability which allows it to be run on multiple operating systems. It also contains an extensive library of modules for different tasks. The main ones used in my project have been listed in Chapter 5-Implementation.

1.7)Targeted Malware Types

There are several malware types that can be detected by my program. Below I have written a definition of each of them:

Ransomware:

- Ransomware encrypts files on a computer, demanding payment for decryption. It spreads via email, compromised sites, or exploit kits. Victims pay ransom in cryptocurrency to regain access.

Keyloggers:

- Keyloggers secretly record keystrokes, capturing sensitive data like passwords and credit card numbers. They can operate at different system levels and transmit logged keystrokes to attackers.

DoS (Denial of Service) Attacks:

- DoS attacks flood a system with illegitimate traffic, causing it to become slow or unresponsive. Techniques include UDP flood, SYN flood, and HTTP flood. The goal is to disrupt services and cause damage.

Reverse Shells:

- Reverse shells establish a connection from victim to attacker, granting remote access to the victim's system. Attackers can execute commands, exfiltrate data, or deploy additional malware.

Packet Sniffers:

- Packet sniffers intercept and monitor network traffic, capturing sensitive information like usernames, passwords, and emails. They operate in promiscuous or targeted mode for reconnaissance and data theft.

Data Theft:

- Data theft involves the unauthorized copying of sensitive information from a victim's system or network. Attackers use various techniques and malware types to steal personal, financial, or intellectual property data. Preventative measures include encryption, access controls, and employee awareness training.

1.8)Similar Products

1.8.1)Introduction

Research on GitHub (a web-based platform for sharing codes and scripts) has revealed a series of HIDS scripts, each serving as a tool for monitoring the integrity of files and system activities. The products discussed have similar features to my final implementation. See below for descriptions of the programs that defined themselves as IDSs.

1.8.2)Script Summaries

1.8.2.1)“Host based ids.py” by samirjamehdar (Samir Jamehdar)

This has the purpose of monitoring changes within a specified directory and its subdirectories. It begins by prompting the user to input the absolute path to a directory to be scanned. Upon receiving the input, the script recursively traverses the directory and its subdirectories to gather information about files, their paths, and their corresponding SHA-1 hash values, storing this information in a log file named "IDSlog.txt". Upon subsequent executions, the script compares the current directory structure and file hashes with those stored in the log file to detect any changes. If changes are detected, the script reports which files have been modified, added, or removed since the last scan. This script provides a straightforward and effective means of monitoring and identifying alterations within a directory, enabling users to detect potential unauthorized modifications or suspicious activity (Jamehdar, 2023).

1.8.2.2)“PyHIDS” by cedricbonhomme (Cédric Bonhomme)

This script focuses on hashing files and command outputs, with a modular structure for extensibility. Its primary function is to monitor the integrity of files and command outputs on a system. The script utilizes cryptographic hashing techniques to compute and compare hash values of files and command outputs against a pre-defined baseline of expected hash values stored in a database. If any discrepancies are found, indicating a potential unauthorized modification or tampering, the script generates alerts in various forms such as logging to a file, sending syslog messages, and notifying via IRC or email. Additionally, the script supports verifying the integrity of its own database using digital signatures. It employs multithreading to efficiently perform integrity checks on multiple files and commands simultaneously. Overall, the script provides a robust mechanism for detecting potential intrusions or unauthorized changes on a system, enhancing its security posture (Bonhomme, 2023).

1.8.2.3)“NIDS-Intrusion-Detection” by ggullgun (Gokberk Gulgun)

This script uses machine learning techniques. The IDS is designed to classify network traffic as either normal or malicious. The script first loads a dataset containing network traffic information, preprocesses it, and then applies feature reduction techniques using Principal Component Analysis (PCA). After feature reduction, the data is normalized to improve model

performance. The script then offers two classification methods: Support Vector Machine (SVM) using a third-party library and a custom implementation of k-Nearest Neighbours (k-NN) algorithm. Finally, it evaluates the performance of the SVM model and the custom k-NN model on the prepared dataset. Overall, the script provides a basic framework for building and testing an IDS using machine learning (Gulgun, 2017).

1.8.2.4)"TrueIDS-Desktop-Application" by vehanmr (Vehan Rathnayake)

This script works by continuously monitoring system logs, system resources (such as CPU usage, memory usage, and disk usage), and network traffic. It analyses this data to detect any potential intrusions or abnormal behaviours. When an intrusion is detected, it alerts the user through the graphical user interface, displaying the intrusion alert along with relevant data and results. The script provides functionalities such as starting and stopping the monitoring process, viewing the results of the last monitoring session, displaying all data collected by the system, exporting all data to an Excel file, and logging out or exiting the application (Rathnayake, 2023).

1.8.2.5)“intrusion-detection-system-host-base-with-Python-port-and-SSH-server-scanner” by ibrahimkhalilmasud (Muhammad Ibrahim Khalil)

This is a network scanning tool primarily focused on identifying open SSH (Secure Shell) ports on remote hosts. It accepts input in various formats, including individual IP addresses, CIDR notation, and dash-separated ranges. By utilizing Python's socket module, the script attempts to establish connections to each specified host on port 22, the default port for SSH. Additionally, it incorporates pexpect (a python module for spawning interactive applications) for automating SSH login attempts to determine not only if the SSH port is open but also if it's accessible with the provided credentials. The script randomizes the order of IP addresses within specified ranges to distribute scanning load and minimize predictability. Finally, it generates an output file containing information about hosts with open SSH ports, including successful login attempts and encountered errors (Khalil, 2020).

1.8.2.6)“Intrusion-Detection-Prevention-System” by hmzakhalid (Hamza Khalid)

This can detect anomaly CPU/RAM usage and file modifications and anomaly network connections. It tracks network connections by periodically querying psutil for active connections and logging any new connections to a specified file. The latter function monitors system processes, checking CPU and memory usage against predefined thresholds and logging processes exceeding these thresholds to a designated log file. It also implements methods to manage file system events like creation, deletion, movement, and modification. It includes functionality to log these events, ignore specific file patterns, and feed feature vectors to an anomaly detector (Khalid, 2023).

1.8.3)Conclusion

By delving into these scripts on GitHub, I've gained valuable insights into the issues they aim to solve and the methodologies they employ to address them. Analysing existing products allows me to identify common pain points, such as the need for efficient file integrity monitoring and rapid intrusion detection, while also recognizing areas for improvement, such as scalability, extensibility, and usability. Moreover, studying these products helps me identify gaps in the sector and innovative approaches that have yet to be explored. By synthesizing the strengths and weaknesses of existing solutions, I can formulate new ideas and approaches to further enhance system security and address emerging threats effectively.

2) Literature Review

2.1)Purpose of the Literature Review

The purpose of a literature review within the context of cybersecurity research is multifaceted. Primarily, it serves to provide a comprehensive understanding of the current state of knowledge and research in a particular area, in this case, intrusion detection and the application layer. By synthesizing information from various sources such as academic articles, conference papers, and research papers, a literature review identifies key concepts, trends, advancements, and gaps in the field. It helps researchers and practitioners gain insights into existing methodologies, techniques, and technologies employed in cybersecurity to address evolving cyber threats. Additionally, a literature review aids in formulating research questions, hypotheses, and methodologies for further investigation. It serves as a foundation upon which new research can build, contributing to the advancement of knowledge and the development of more robust and effective cybersecurity solutions.

2.2)Introduction

In the rapidly evolving landscape of cybersecurity, safeguarding digital assets has become a paramount concern. The emergence of sophisticated cyber threats has underscored the critical need for robust and effective Intrusion Detection Systems. This literature review focuses specifically on detection methods, with a particular emphasis on the application layer. As cyber threats advance in complexity, this layer's vital role in facilitating communication within networked environments has garnered increased attention. This review explores the utilization of signature-based and malware-based detection methods and delves into the integration of artificial intelligence (AI) techniques, incident response mechanisms, and obfuscation detection to enhance the overall defence of computers against cyber threats. It has been compiled from several articles across ResearchGate, IEEE Xplore, ACM Digital Library and Google Scholar.

2.3)Core Components of the Project

2.3.1)Signature Detection

Firstly, one must look at the advances made in signature-based detection methods. These involve examining network traffic, comparing to known signatures, and generating an alert when a match is made. Figure 3 below is a diagram of signature based detection steps.. There are several papers which can provide a good understanding of how systems that use these techniques work. The article by Kamran Shaukat et al. discusses the importance of automation over the conventional security systems and advises that the datasets often used in IDS need updating (Shaukat, et al., 2020). Automation and updating datasets for Intrusion Detection Systems (IDS) could inform the development of dynamic malware scanners. A publication by Noe Marcelo Yungaicela-Naula et al. discusses using machine learning and deep learning to detect DoS attacks on the Application layer by implementing a flow

controller, which monitors and controls traffic flows, and a flow manager module which helps mitigate attacks by saving logs from the detection module (Yungaicela-Naula, et al., 2021). Implementing flow control and monitoring for DoS attacks at the application layer can provide insights for enhancing live process monitoring. This shows that IDS need to be able to maintain constant surveillance of a system.

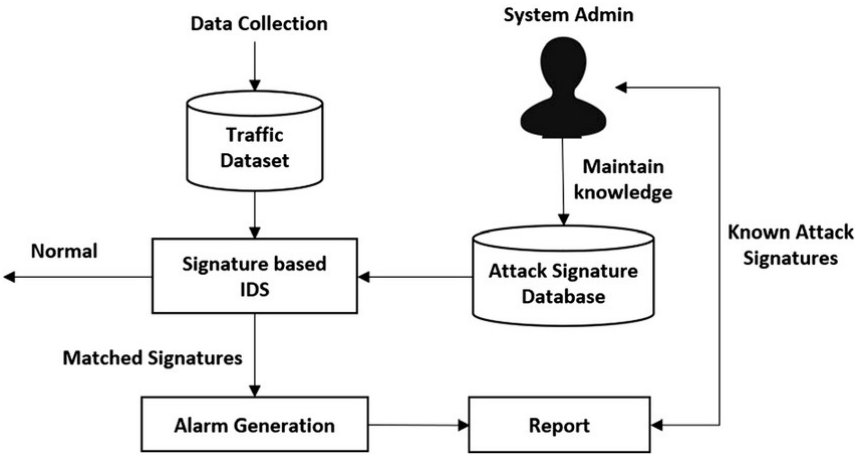


Figure 3- Diagram of Signature based detection steps

The publication by Zakiyabanu S. Malek et al. discusses a similar method of signature detection called Pattern-Based which detects when a user's system deviates from their normal processes and does other things which could potentially be due to an unknown threat (Malek, et al., 2020). This research provides strategies for identifying deviations in user system behaviour. By exploring signature-based detection methods and their integration with machine learning, their importance in developing a hybrid IDS for the application layer becomes clear. By automating detection and updating datasets, it is possible to enhance the system's ability to identify threats, strengthening its overall defences.

2.3.2)Anomaly Detection

Research has also been made into various articles related to anomaly-based detection systems. Figure 4 shows anomaly-based detection steps. These are mostly about flagging deviations from normal activity. A publication by Dominik Breitenbacher et al. talks about how IoT device makers priorities cost over security by prioritising the device doing whatever its main purpose is and ignoring potential security concerns. Their solution is HADES-IoT which monitors process spawning and stops any unauthorized processes before execution (Breitenbacher, et al., 2021). John H. Ring IV et al. has written a proposal of a host-based intrusion detection system which run based on the anomaly-based detection concept and uses an algorithm that finds anomalies at the application level (Ring, et al., 2021).

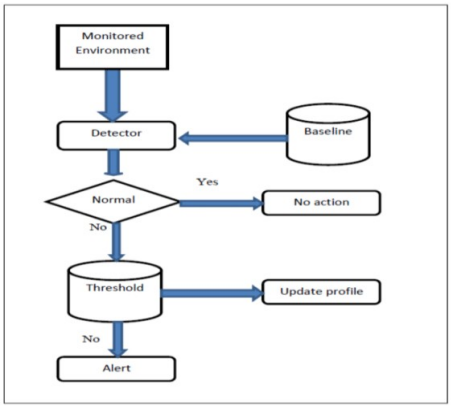


Figure 4- Diagram of Anomaly based detection steps

By proposing a host-based IDS that utilizes anomaly-based detection algorithms, this study contributes to the development of intrusion detection mechanisms tailored to the application layer. The article by Nour Moustafa et al. discusses Explainable Artificial Intelligence (XAI) enhanced cybersecurity measures and how XAI are used to enhance anomaly-based intrusion detection. It tracks the network traffic of IoT which are monitored by a pre-processor which sends data to decision to determine whether the traffic indicates a threat. If a threat is detected, the system deals with the threat and notifies the user(s) and the specialists of the event, the latter of which record for further analysis, allowing for them to later solve the issue causing the threat (Moustafa, et al., 2023). this research provides methodologies for integrating explainable AI into host-based IDSs. Suzan Hajj et al. produced an extensive review on anomaly-based intrusion detection systems discussing, detection techniques, attacks forms, attack features, most-used datasets, challenges and their potential solutions (Suzan Hajj, 2021).They talk about how Anomaly-based intrusion detection systems (AIDSs) identify the anomalies by comparing the received network traffic with normal traffic behaviour with anomaly being defined as behaviour that deviates significantly from normal. If any abnormal pattern is detected, the IDS warns the network security administrator of a potentially suspicious activity. Studying anomaly-based systems and their integration with XAI, it's easy to understand their role in complementing signature-based methods. The conference paper by Hao Wang et al. discusses a combined detection method based on matching attacks to their signature and the LSTM model which is used to record the normal behaviour of a network and use it as a baseline for detecting any issues (Wang, et al., 2020).This study presents a methodology for establishing baseline network behaviour, informing the development of a host-based IDS capable of detecting anomalies. By improving interpretability, it makes better real-time decisions in threat mitigation within the hybrid IDS.

2.3.3)Hybrid Detection

Articles relating to hybrid systems, which forms the base of my project, and using AI for security. In their publication, Stefanos Tsimenidis et al. discuss the pros and cons of anomaly-based and signature-based (referred to here as “knowledge-based) systems and highlights the importance of Deep learning over Machine learning. It also highlights some of the weaknesses of the application layer, namely that it is the key to accessing the computer for the user and links it to the physical world, which means it is the most vulnerable and the first line of defence (Tsimenidis, et al., 2022).

Another interesting article is by Mohammad Aljanabi et al. which explores different forms of intrusion detection such as misuse detection (aka signature detection) and anomaly detection as well as the challenges of it. It reviews distinctive features of Machine Learning. It also highlights the advantages of machine learning in improving the weaknesses of IDSs in different forms such as Artificial Neural Networks and Decision Trees. (Aljanabi, et al., 2021). Baicheng Zhong et al. released a publication which discusses a model for using AI to monitor computer network security through firewalls which maintains constant analysis of data and information passed through the network (Zhang, et al., 2022). Integrating anomaly-based and signature-based systems into a hybrid IDS is crucial for addressing the evolving threat landscape. Leveraging both approaches and AI creates a robust defence mechanism against sophisticated attacks.

2.3.4)Application Layer Defence

The project focuses on the weakness specifically the Application Layer in the OSI model(see Figure 5 for details on each layer), so research publications around this have been utilised. Giuseppe Nebbione and Maria Carla Calzarossa discusses the importance of the security of application layer protocols as well as their vulnerabilities and how these could be protected (Nebbione & Calzarossa, 2020). This research offers methodologies for continuous analysis of network data, facilitating the development of AI-driven host-based IDS solutions. Mahmoud Abbasi et al. released a research article in they talk about the IoT application layer, security requirements, threats, and countermeasures in this layer, and some of the open issues and future research lines (Abbasi, et al., 2022). Parvathy K discusses application layer protocols such as MQTT, AMQP, CoAP and how they are under threat from packet sniffers, data theft, reverse engineering and ransomware (K, et al., 2023) By identifying threats to application layer protocols, this study informs the development of host-based IDS by highlighting potential attack vectors and mitigation strategies, particularly relevant for defending against packet sniffers and data theft.

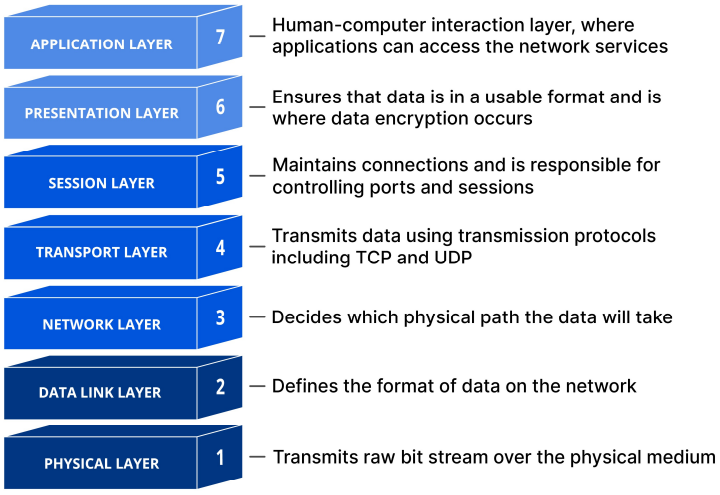


Figure 5- Infographic of OSI Model

Pankaj Kumar Gupta et al. discusses the potential attacks to the application layer which occur due to its “open-fished” nature such as access attacks, authentication attacks, backdoor attacks, and phishing attacks (Gupta, et al., 2020). This research is helpful in understanding attack vectors and mitigation measures, essential for protecting against data theft and other threats. Mayukha Selvaraj and R. Vadiel released an article articulating the threats to each layer of the OSI model as well as the possible mitigations for each. They describe the application layer as the one with the largest threat surface (the sum of the different points of attack) because of its functionality of user interaction. It goes into a more specific list of

attacks that includes data theft, HTTP floods, DoS, trojans, viruses and SQL injections. It lists the mitigations for this to include backup systems, Access control lists and firewalls (S & Vadivel, 2023). Preeti Sinha et al. lists some of the protocols supported by the application layer, namely HTTP, SMTP and FTP then lists some of the network security attacks they're vulnerable to. In this case, trojans, bounce attacks and email-spoofing-viruses, respectively. They indicate that firewalls and antiviruses are the best counter for such attacks (Sinha, et al., 2017). By identifying vulnerabilities in application layer protocols, this research informs the development of host-based IDS by highlighting attack vectors and recommending countermeasures from which tools can be taken and combined to develop the malware scanner. Understanding application layer vulnerabilities is vital for designing effective defence mechanisms in a hybrid IDS. By addressing protocol vulnerabilities and attack vectors, it is possible to protect critical applications and data.

2.4)Enhancing Program Capabilities

2.4.1)Preventing Obfuscation

To ensure that hidden scripts can still be detected, research into obfuscation detection was necessary. Jin-Young Kim and Sung-Bae Cho discuss using both global and local features to detect malware variants. The process is transforming malware into an image to efficiently represent global features with pre-defined latent space then extracting local features using the binary code sequences (Kim & Cho, 2022). Jorge Maestre Vidal wrote a report discussing 2 obfuscation methods- locality-based mimicry by action pruning(carefully controlling the timing of adversarial actions to stay within predefined thresholds, making the malicious behaviour appear more similar to legitimate usage patterns and thus harder to detect by intrusion detection systems) and locality-based mimicry by noise generation(manipulating metrics by introducing noise or padding actions to make the adversary's behaviour appear more similar to legitimate usage patterns) (Vidal & Monge, 2020). Mohammed M. Alani et al. proposes a method of detecting obfuscated malware based on features extracted from memory dumps. This is a 2-phase process. Phase 1 is development where the system processes the raw memory dumps through the feature extraction unit to extract a comprehensive set of features, resulting in the initial version of the dataset. Phase 2 is deployment which generates a prediction indicating whether the memory dump is infected with malware or not (Alani, et al., 2023). Detecting obfuscated malware is challenging but essential for application layer security. Researching obfuscation detection techniques helps to enhance the hybrid IDS's resilience to stealthy threats.

2.4.2)Incident Response

Once an attack has been detected, effort will then be needed to contain it. Therefore, this review includes research studies into what incident response mechanisms are available and how they could be implemented. Kennedy Torkura et al. discusses the challenges in security

of cloud storage, namely that it doesn't get designed with security features such as firewalls and IDSs. The authors suggest getting around this by regularly logging the events that occur within the storage space and analysing it regularly. There is then an active responder built into the system which notifies a human agent to act and can suspend a suspicious user if needed by revoking their API key (Torkura, et al., 2019). Addressing challenges in cloud storage security, this study proposes an incident response mechanism involving event logging and analysis which are needed for proper incident response as it necessary to understand what happened in the aftermath of an attack. Florian Kaiser et al. discussed how Cyber Threat Intelligence (a collection of data and information concerning cyber threats and threat actors which is used to defend against future attacks) is useful is useful in predicting threats. They reference repositories of information made by MITRE (a not-for-profit organisation that focuses on providing technical support to US government agencies) such as ATT&CK(see Figure 6) and D3FEND which have info on attacking and defensive techniques respectively that have been deployed. The article goes on to discusses how identification of malware patterns could be made easier using the basis data of the ATT&CK framework (Kaiser, et al., 2022).

Figure 6- Screenshot of MITRE ATT&CK Framework website

This links to the general concept of how a signature-based intrusion detection system may run where there is some form of repository or directory which contains some form of

malware pattern. Effective incident response is integral to our hybrid IDS's efficacy. Leveraging CTI and proactive strategies, one can minimize the impact of security incidents, maintaining application integrity.

2.5)Conclusion

In conclusion, this literature review seeks to provide a comprehensive exploration of general intrusion detection methods and some more detail on the application layer's vulnerability. The advancements in signature-based and anomaly-based detection methods, coupled with the integration of artificial intelligence (AI) techniques, reflect the dynamic strategies employed to fortify defence mechanisms against evolving cyber threats. The discussion on hybrid systems, obfuscation detection, and incident response mechanisms contributes to a holistic understanding of host based IDSs. Recognizing the challenges and needs in securing the application layer, this review highlights the ongoing efforts to enhance the robustness and efficacy of intrusion detection, showcasing a multifaceted approach essential for safeguarding digital assets in the contemporary cybersecurity landscape.

3) Analysis and Requirements

The project's main requirements are to maintain a detailed dataset different malicious script keywords that traverse different malware types and different programming languages, most notably Python, C and Java. It also needs to be able to detect common signs of active malware attacks. Additionally, it must be able to detect anomalies within a computer's processes and identify malicious ones. These along with the signature-detected threats need to be identified and reported to the user with urgency at time of detection. These also must be logged for future analysis.

3.1)Functional Requirements

- File Scanning: The system should be able to scan files in the selected directory for potential malware using predefined signatures and patterns; It must distinguish between binary and text files and apply appropriate checks.
- Live Process Monitoring: The system should continuously monitor live processes on the computer for suspicious activities based on predefined indicators.
- Anomaly-Based Intrusion Detection: The system should analyse CPU and memory usage to detect abnormal behaviour indicative of a potential intrusion; It must trigger alerts when CPU or memory usage exceeds predefined thresholds.
- Graphical User Interface (GUI): The system should provide a user-friendly GUI for interacting with the application; The GUI should include buttons for selecting directories, starting, and stopping the scanning processes, and displaying scan results.
- Alerting Mechanism: The system must provide an alerting mechanism to notify the user of potential threats; Alerts can be displayed in the GUI and may include warning messages and popup notifications.

- Configuration Options: Users should have the ability to configure certain settings, such allowed applications and detection thresholds.
- Logging and Reporting: The system should log scanning activities, detected threats, and other relevant information to a log file; Users should have the option to view or export log files for analysis.

3.2)Non-Functional Requirements

- Performance: The system should perform file scanning and process monitoring with minimal impact on the computer's overall performance; Scanning should be efficient, and live monitoring should not cause significant system slowdown.
- Usability: The GUI should be intuitive and user-friendly; users with minimal technical knowledge should be able to navigate and use the system effectively.
- Reliability: The system must be dependable, with a low rate of false positives and false negatives in detecting potential threats.
- Security: The application should store signatures, logs, and other sensitive information securely; The intrusion detection system itself should not be vulnerable to exploitation by malicious entities.
- Scalability: The system should be able to manage a growing malware dataset and adapt to changes in the threat landscape.
- Compatibility: The application should be compatible with major operating systems commonly used on personal computers, such as Windows, macOS, and Linux.
- Maintainability: The codebase should be well-documented and maintainable, facilitating future updates and improvements; The system should support easy signature updates and database maintenance.
- Availability: The system should be available and responsive whenever users need to perform scans or check for alerts.
- Regulatory Compliance: The system should comply with relevant legal and regulatory requirements concerning data privacy and security.
- Training and Support: The application should provide user documentation and support resources to assist users in understanding and using the system effectively.

3.3)Legal, Social, Ethical and Professional Issues and Considerations

3.3.1)Legal Issues

Laws, regulations, and legal principles that govern a particular activity, profession, or situation. These are rules established by governmental bodies or other authorized institutions.

a. Computer Misuse Act 1990 which involves monitoring unauthorized access to computer system. The Computer Misuse Act 1990 makes the following actions chargeable offences:
   - Unauthorised access to computer material
   - Unauthorised access with intent to commit or facilitate commission of further offences.
   - Unauthorised acts with intent to impair, or with recklessness as to impairing the operation of a computer.

b. Data Protection Act 2018 which requires testers to protect any data impacted by the testing process. UK implementation of the GDPR.
   - EU's General Data Protection Regulation (GDPR) which requires that that data is, among other things, obtained legally and kept safe(see Figure 7 below).

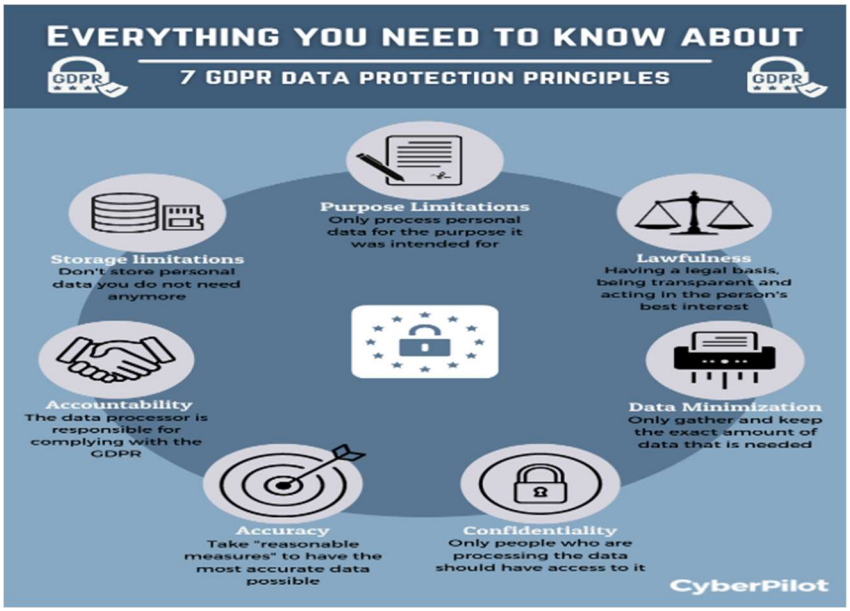


Figure 7- Infographic of the principles of GDPR

3.3.2)Social Issues:

Concerns related to interactions and dynamics within society, including cultural norms, values, beliefs, and societal expectations.

a. Trust of Employees: Constant monitoring without transparency can erode trust, leading to morale decline and increased stress. Employers should openly communicate reasons for monitoring, its scope, and reassure employees it's for security, not micromanagement.
b. Impact on Productivity: False positives and frequent notifications disrupt workflow and reduce productivity. Employers need to fine-tune software to minimize false alerts, prioritize based on severity, and establish clear protocols to address legitimate threats while minimizing interruptions.

3.3.3)Ethical Considerations:

Evaluating actions, decisions, and behaviours based on principles of right and wrong, fairness, honesty, and integrity. Ethics guide moral reasoning and conduct.

a. Informed Consent: Obtaining informed consent before monitoring respects employees' autonomy and privacy rights. Employers must clearly inform employees about monitoring's nature, reasons, and potential impacts, allowing them to make informed decisions.
b. Transparency: Transparent operation of the scanner builds trust and accountability. Employers should communicate clearly about data collection, analysis, and access, ensuring employees understand their rights and responsibilities regarding privacy and security.
c. Proportionality and Necessity: Monitoring should be proportional to security risks and necessary for legitimate objectives. Employers should conduct risk assessments to strike a balance between security and privacy, avoiding excessive or unjustified surveillance.

3.3.4)Professional Considerations:

Standards, norms, and expectations within a particular profession or field of expertise. These encompass professional conduct, responsibilities, and best practices.

a. Competence and Training: Personnel managing the program should have expertise in cybersecurity, system operation, and incident response. Ongoing training is crucial to stay updated on evolving threats and technologies.
b. Ethical Hacking: Ethical hacking identifies vulnerabilities responsibly, following guidelines and obtaining proper authorization to minimize disruption and prioritize data integrity.
c. Disclosure of Conflicts of Interest: Personnel involved in monitoring should disclose conflicts of interest to ensure impartiality and integrity in decision-making regarding security incidents.

4) Design

Here I discuss diagrams I made to create a visual image of how the program is supposed to be structured and run properly.

4.1)Diagrams

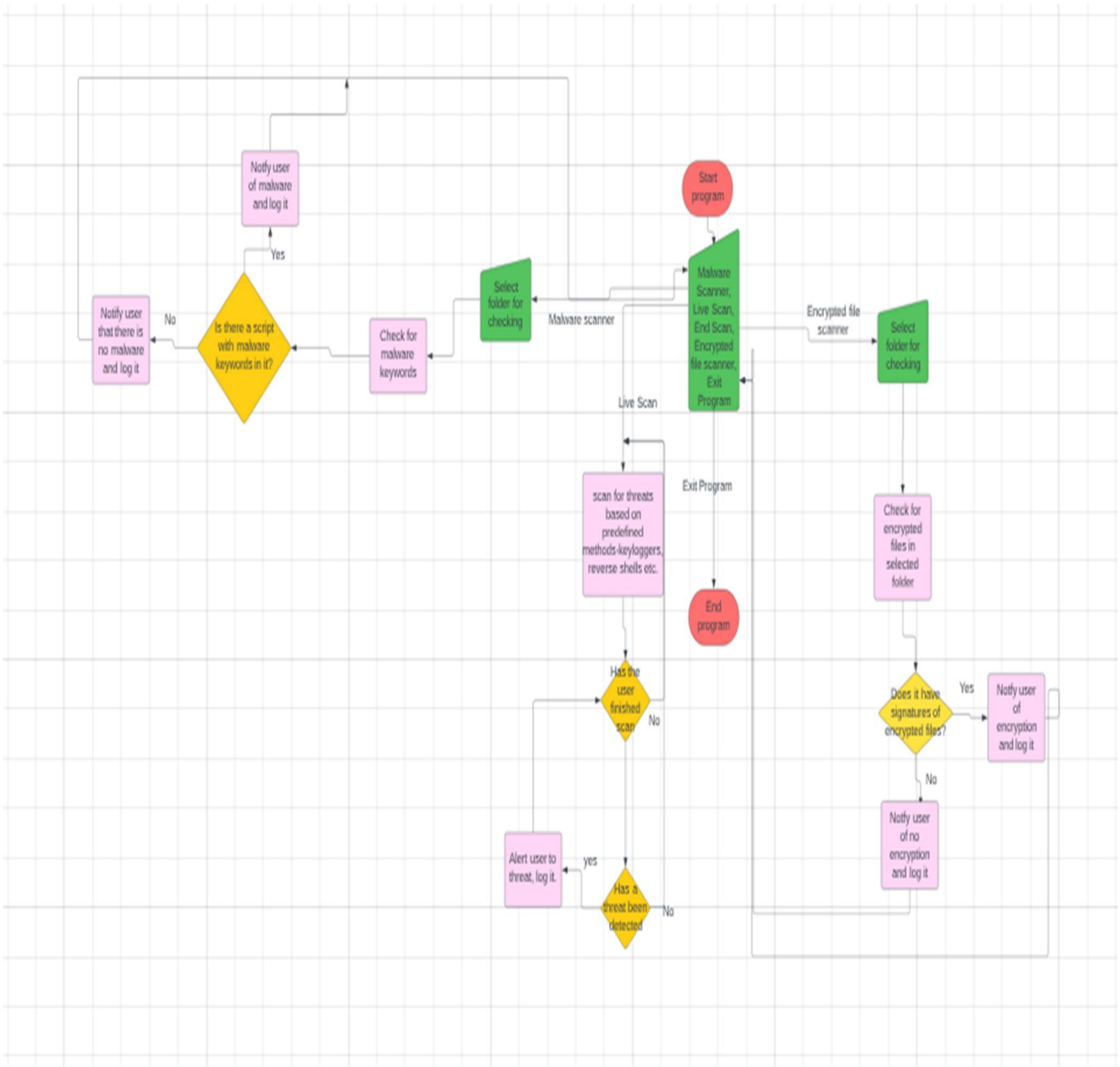


Figure 8- Flowchart of the Malware Scanner

Figure 8 is a flowchart of the program. The red stadiums are the indicators, the yellow diamonds are questions/decisions. The pink rectangles are processes conducted by the program and the green shapes are manual user inputs.

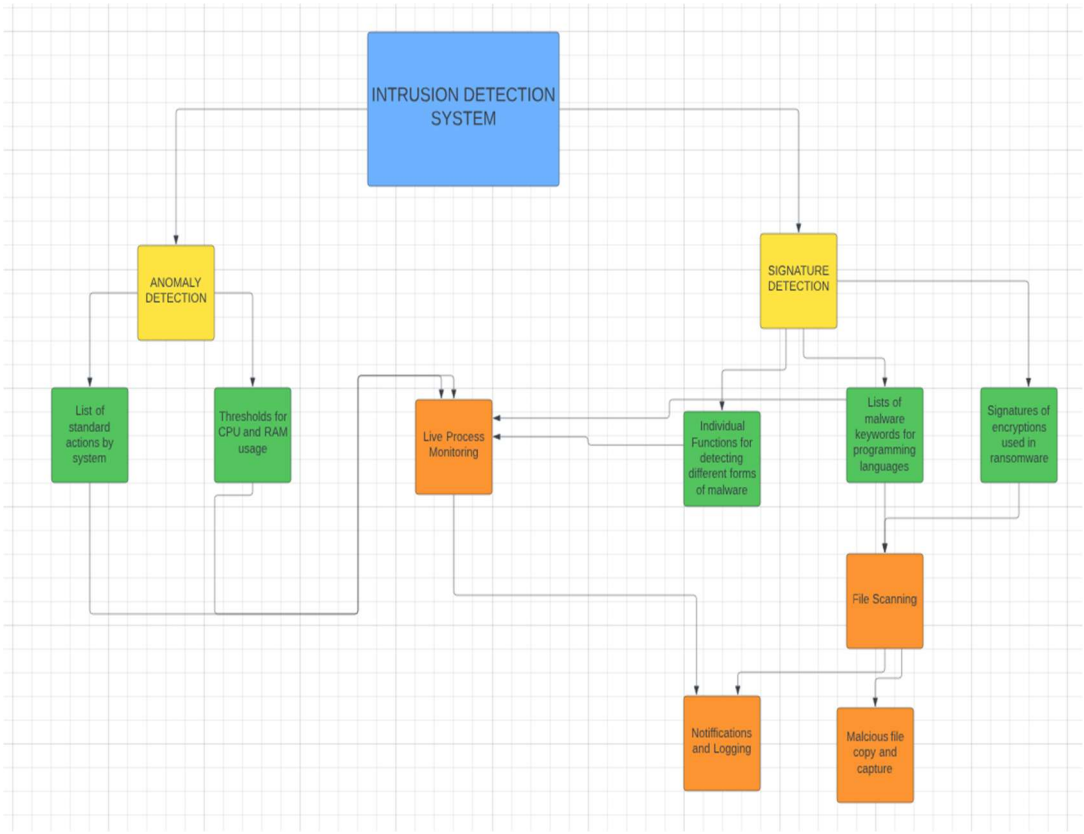


Figure 9- UML diagram of the Malware Scanner

Figure 9 shows a UML diagram of the program. The blue represents the program, the green represents the variables which the program stores and uses, the orange is the processes conducted using the variables to detect threats and the yellow represents the the signature and anomaly-based intrusion detection mechanisms.

5) Implementation

5.1)Source Code Breakdown

The program can be split into 3 main parts: a malicious script finder, an encrypted file finder and a live scan function. It uses several python libraries to run. The main ones are:

- **logging**-used for recording the results of scans.
- **scapy**-used for capturing packets, which is a key part of the keylogger and DoS detection,
- **os**-which is for monitoring the operating system for processes and resource usages.
- **psutil**-which checks the processes running on the computer, which is part of the detection of anomaly programs running, reverse shells and detecting packet sniffers.
- **tkinter**-Used for the designing the Graphical User Interface which combines all the elements of the program together and allows the user to use it.

The source code in its entirety is available to view in Appendix B. In the section below, I will outline the main functions within the program.

5.1.1)Malicious script detector

The malicious script detector(see Figures 10.1-10.6) is a tool designed to detect potentially harmful scripts within directories. It recursively scans files, excluding certain file types that are unlikely to contain malware. It utilizes lists of keywords commonly associated with malicious activities such as keyloggers, reverse shells, DoS attacks, ransomware, and packet sniffers. These keywords are categorized by programming language (Python, C#, C, Java) and are sourced from known malware repositories. The main functions of the program include:

- Scan_File: This function scans individual files for malicious content based on their file extension and content. It checks for keywords specific to each programming language and also searches for shellcode patterns in binary files(Figure 10.1-Figure 10.4).
- Capture_Malicious_File: If a suspicious file is detected, this function copies it to a destination folder named 'Captured_Scripts' for further analysis and containment(Figure 10.5).
- Scan_Directory: This function recursively scans directories and their subdirectories for files to be analyzed. It calls the Scan_File function for each file encountered (Figure 10.5).
- Start_Scanner: This function initiates the scanning process by allowing the user to select a folder for scanning. It then repeatedly scans the selected folder at regular intervals, providing real-time feedback on any detected threats(Figure 10.6).

The tool also reads through non-coding files such as Word, PowerPoint, and Excel documents, which are checked for malicious macros that can be embedded within them. Additionally, it utilizes OCR data extraction to extract text from JPEG and PNG images, searching for embedded malicious content.

When a suspicious script or embedded text with malicious keywords is detected, the user is alerted, and the file is moved to the 'Captured_Scripts' directory for further investigation. This program provides a proactive approach to identifying and containing potential security threats within a system.

```
exemalware_keywords = ["GetAsyncKeyState", "SetWindowsHookEx", "flood", "udp", "tcp", "encrypt", "decrypt", "ransom", "pcap", "sniff", "captu
pymalware_keywords = ['socket.socket', 'subprocess.Popen', 'os.popen', 'pty.spawn', 'pty.fork', 'socket.create_connection', 'socket.settimeou
csharpmalware_keywords = ["Keylogger|GetAsyncKeyState", "nativekeypressed", 'TcpClient', 'TcpListener', 'Process.Start', 'System.Net.Sockets'
cmalware_keywords = ['socket', 'winsock', 'CreateProcess', 'system', 'ShellExecute', 'getaddrinfo', 'connect', 'send', 'recv', 'Sleep', 'Crea
javamalware_keywords = ['java.net.Socket', 'java.net.ServerSocket', 'java.io.BufferedReader', 'java.io.InputStreamReader', 'java.lang.Runtime
allmalwarekeywords= exemalware_keywords+pymalware_keywords+csharpmalware_keywords+cmalware_keywords+javamalware_keywords
1 usage
def Scan_File(file_path,allmalwarekeywords):
    try:
        with open(file_path, 'rb') as file:
            content = file.read()
            file_extension = os.path.splitext(file_path)[1].lower()

            # Binary file extensions
            binary_extensions = ['.exe']
            if file_extension in binary_extensions:
                if any(keyword.encode('utf-8') in content for keyword in exemalware_keywords):
                    alert_message = f"Alert! Malicious keywords found in file: {file_path}"
                    return True
                exe_shellcode_pattern = re.compile(rb'\\x[0-9a-fA-F]{2}')  # checks for shellcode in an exe file.
                if exe_shellcode_pattern.search(content):
                    alert_message = f"Alert! Malicious shellcode pattern found in file: {file_path}"
                    return True
            else:
                try:
                    content_str = content.decode( encoding: 'utf-8', errors='ignore')

                    # Additional checks for text file extensions
                    file_extension = os.path.splitext(file_path)[
                        1].lower()
                    document_extensions = ['.doc', '.pdf', '.docx', '.xlsx', '.pptx']
                    if file_extension in document_extensions:
                        # Handle each document type separately
                        if file_extension == '.doc':  # For .doc files
                            doc_content = docx.Document(file_path).read_text()
                            keyword_count = sum(
                                keyword.lower() in doc_content.lower() for keyword in allmalwarekeywords)
                            if keyword_count >= 2:
                                return True
```

Figure 10.1- Malicious Script Detector

```
    elif file_extension == '.txt':
        # Check for keywords in TXT files
        keyword_count = sum(
            keyword.lower() in content_str.lower() for keyword in allmalwarekeywords)
        if keyword_count >= 2:
            return True

    elif file_extension == '.pdf':  # For .pdf files
        pdf_content = ""
        pdf_reader = PyPDF2.PdfReader(file_path)
        for page in pdf_reader.pages:
            pdf_content += page.extract_text()
        keyword_count = sum(
            keyword.lower() in pdf_content.lower() for keyword in allmalwarekeywords)
        if keyword_count >= 2:
            return True

    elif file_extension == '.docx':  # For .docx files
        docx_content = ""
        doc = docx.Document(file_path)
        for paragraph in doc.paragraphs:
            docx_content += paragraph.text
        keyword_count = sum(
            keyword.lower() in docx_content.lower() for keyword in allmalwarekeywords)
        if keyword_count >= 2:
            return True

    elif file_extension == '.xlsx':  # For .xlsx files
        wb = openpyxl.load_workbook(file_path)
        keyword_count = 0
        for sheet in wb.worksheets:
            for row in sheet.iter_rows():
                for cell in row:
                    if cell.value and any(keyword.lower() in str(cell.value).lower() for keyword in
                                          allmalwarekeywords):
                        keyword_count += 1
                        if keyword_count >= 2:
                            return True
```

Figure 10.2- Malicious Script Detector

```
    elif file_extension == '.pptx':  # For .pptx files
        prs = Presentation(file_path)
        pptx_content = ""
        for slide in prs.slides:
            for shape in slide.shapes:
                if hasattr(shape, "text"):
                    pptx_content += shape.text
        keyword_count = sum(
            keyword.lower() in pptx_content.lower() for keyword in allmalwarekeywords)
        if keyword_count >= 2:
            return True

image_extensions=['.jpeg', '.jpg', '.png']
if file_extension in image_extensions:
    # Perform OCR to extract text from the image
    image = Image.open(file_path)
    extracted_text = pytesseract.image_to_string(image)
    # Check the extracted text for malicious keywords
    if any(keyword.lower() in extracted_text.lower() for keyword in allmalwarekeywords):
        alert_message = f"Alert! Malicious keywords found in image: {file_path}"
        return True
```

Figure 10.3- Malicious Script Detector

```
            elif file_extension == '.py':#Python
                keyword_count = 0
                for keyword in pymalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:#check for at least 2 keywords
                        return True
            elif file_extension == '.cs':#C Sharp
                keyword_count = 0
                for keyword in csharpmalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:
                        return True
            elif file_extension == '.c':#C
                keyword_count = 0
                for keyword in cmalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:
                        return True
            elif file_extension == '.java':#Java
                keyword_count = 0
                for keyword in javamalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:
                        return True
            exe_shellcode_pattern = re.compile(b"\\\\x[0-9a-fA-F]{2}")  # checks for shellcode in the above file types
            if exe_shellcode_pattern.search(content):
                return True
            else:
                pass  # skips any file that isn't python, C#, visual basic, c or java

        except (UnicodeDecodeError, PermissionError, FileNotFoundError) as e:
            logging.error( msg: f"Error reading {file_path}: {e}", exc_info=True)

except (PermissionError, FileNotFoundError) as e:
    logging.error( msg: f"Error reading {file_path}: {e}", exc_info=True)
return False
```

Figure 10.4- Malicious Script Detector

```
def Capture_Malicious_File(file_path, destination_folder):#Copies any malicious file and sends them to the destination folder(in this instance "Captured_Scripts")
    try:
        shutil.copy(file_path, destination_folder)
        logging.info(f"Malicious file captured and contained: {file_path}")
    except Exception as e:
        logging.error( msg: f"Error capturing malicious file: {e}", exc_info=True)

2 usages
def Scan_Directory(directory,allmalwarekeywords,output_text):
    malware_found = False

    for root, dirs, files in os.walk(directory):
        dirs = dirs or []

        for file in files:
            file_path = os.path.join(root, file)
            file_extension = os.path.splitext(file_path)[1].lower()

            if file_extension not in ['.py', '.cs', '.c', '.java','.doc', '.pdf', '.docx', '.xlsx', '.pptx', '.jpeg', '.jpg', '.png']:
                continue  # Skip scanning this file

            if Scan_File(file_path,allmalwarekeywords):
                malware_found = True
                alert_message = f"Alert! Signature matched in file: {file_path}"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)
                Capture_Malicious_File(file_path, destination_folder: "Captured_Scripts")

        for subdir in dirs:  # Checks subfolders within chosen folder.
            subdir_path = os.path.join(root, subdir)
            Scan_Directory(subdir_path,allmalwarekeywords, output_text)

    if not malware_found:
        output_text.insert(tk.END, "All (other) files here are safe.\n")
        logging.info("No malware found")
```

Figure 10.5- Malicious Script Detector

```
def Start_Scanner(allmalwarekeywords, output_text):  # starts scan of the file
    global stop_folder_scanner_flag
    global scan_folder
    scan_folder = filedialog.askdirectory()
    output_text.insert(tk.END, f"Scan folder selected: {scan_folder}\n")
    while not stop_folder_scanner_flag:
        Scan_Directory(scan_folder, allmalwarekeywords, output_text)
        time.sleep(60)  # Sleep for 60 seconds before the next scan
```

Figure 10.6- Malicious Script Detector

5.1.2)Encrypted File detector

The program also has a function for detecting encrypted files within a directory(see Figures 11.1-11.3). The encrypted files could be a sign of ransomware. The function "Is_File_Encrypted"(Figures 11.1 and 11.2) is responsible for determining if a file is encrypted. It utilizes various methods to detect encryption, including checking file extensions commonly associated with encrypted files and analysing file signatures. Here's an explanation of its functionality:

- File Extension Check: The function first checks if the file has a known encrypted file extension. It compares the file's extension with a list of common encrypted file extensions like .locky, .wnry, etc. If a match is found, it considers the file encrypted.
- MIME Type Check: Next, it determines the MIME type of the file and compares it with the file's extension. If there's a mismatch between the extension and MIME type, it issues a warning, as this could indicate tampering or mislabelling.
- Signature Analysis: The function reads the first 1024 bytes of the file and compares them against known encryption signatures. These signatures are sequences of bytes that commonly appear at the beginning of encrypted files. If a signature match is found, it concludes that the file is encrypted.
- Error Handling: The function gracefully handles exceptions such as file not found, permission errors, or decoding errors. If any error occurs during the analysis, it logs the error and considers the file encrypted to avoid potential security risks.

The function "Scan_For_Locked_Files"(Figure 11.3) utilizes Is_File_Encrypted to scan a selected folder and its subdirectories for encrypted files. It iterates through each file, calling Is_File_Encrypted to check if it's encrypted. If an encrypted file is found, it alerts the user and logs a warning message. Finally, it provides feedback on the completion of the scan, indicating whether any locked files were detected. This functionality enables users to identify potentially compromised files within their system.

```python
def Is_File_Encrypted(file_path, output_text):
    encryption_signatures = [
        b"gAAAA",  # Fernet
        b"Salted__",  # OpenSSL
        b"BEGIN RSA PRIVATE KEY",  # RSA private key
        b"BEGIN DSA PRIVATE KEY",  # DSA private key
        b"BEGIN EC PRIVATE KEY",  # EC private key
        b"-----BEGIN PGP PRIVATE KEY BLOCK-----",  # PGP private key block
        b"-----BEGIN ENCRYPTED PRIVATE KEY-----",  # Encrypted private key
        b"-----BEGIN CERTIFICATE-----",  # Certificate
        b"-----BEGIN ENCRYPTED MESSAGE-----",  # Encrypted message
        b"-----BEGIN PKCS7-----",  # PKCS7 (Cryptographic Message Syntax Standard)
        b"-----BEGIN PGP MESSAGE-----",  # PGP message
        b"-----BEGIN GPG MESSAGE-----",  # GPG message
    ]
    try:
        # Check file extension
        _, file_extension = os.path.splitext(file_path)
        file_extension = file_extension.lower()

        # Check if the file extension matches common encrypted file extensions
        encrypted_extensions = ['.wcry', '.wnry','.wncry','.locky', '.cerber', '.zepto', '.crysis', '.lockyv2']
        if file_extension in encrypted_extensions:
            output_text.insert(tk.END, f"File {file_path} has a known encrypted file extension.\n")
            return True

        # Get the MIME type of the file
        mime_type, _ = mimetypes.guess_type(file_path)
        if mime_type is not None:
            # Extract the extension from the MIME type
            _, file_extension = os.path.splitext(file_path)
            file_extension = file_extension.lower()
            mime_extension = mimetypes.guess_extension(mime_type)

            # Compare the file extension to the MIME type's extension
            if file_extension != mime_extension:
                output_text.insert(tk.END,f"Warning: File extension ({file_extension}) does not match MIME type ({mime_extension}).\n")
```

Figure 11.1- Encrypted File Detector

```
    # Open the file in binary mode for reading
    with open(file_path, 'rb') as file:
        # Read a portion of the file for signature analysis
        data = file.read(1024)  # Read the first 1024 bytes of the file
        # Check for known encryption signatures
        for signature in encryption_signatures:
            if data.startswith(signature):
                output_text.insert(tk.END, f"File {file_path} appears to be encrypted.\n")
                return True
except (FileNotFoundError, PermissionError, UnicodeDecodeError) as e:
    output_text.insert(tk.END, f"Error checking file {file_path}: {e}\n")
    return True
except Exception as e:
    # Handle other exceptions gracefully
    output_text.insert(tk.END, f"Error checking file {file_path}: {e}\n")
    return True
return False
```

Figure 11.2-Encrypted File Detector

```
def Scan_For_Locked_Files(output_text):
    global selected_folder
    selected_folder = filedialog.askdirectory()
    output_text.insert(tk.END, f"Scanning for locked files...\n")
    locked_files_found = False

    for root, dirs, files in os.walk(selected_folder):
        for file in files:
            file_path = os.path.join(root, file)

            if Is_File_Encrypted(file_path, output_text):
                alert_message = f"Alert! Locked file found"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)
                locked_files_found = True

    if not locked_files_found:
        output_text.insert(tk.END, f"No locked files found.\n")

    output_text.insert(tk.END, f"Scan for locked files complete.\n")
```

Figure 11.3- Encrypted File Detector

5.1.3)Live Malware detector

The live scan function(see Figure 12) is used to detect all the aforementioned malware when it is attacking the system excluding ransomware. It calls upon a series of individual functions that run concurrently to detect a wide range of threats at once.

```python
def Live_Scan(output_text, base_directory, target_ports):
    output_text.insert(tk.END, "Live scan started\n")
    global stop_live_scanner_flag
    stop_live_scanner_flag = False

    # Start file monitoring in a separate thread
    file_monitor_thread = threading.Thread(target=start_file_monitoring, args=(base_directory, output_text), daemon=True)
    file_monitor_thread.start()

    while not stop_live_scanner_flag:
        if Anomaly_Action_Checking(output_text):
            alert_message = "Alert!-Anomaly unlisted process detected!"
            output_text.insert(tk.END, alert_message + "\n")
            logging.warning(alert_message)

        if Anomaly_Resource_Usage(output_text):
            alert_message = "Alert!-Anomaly CPU/RAM usage detected!"
            output_text.insert(tk.END, alert_message + "\n")
            logging.warning(alert_message)

        Packet_Sniffer_Detection(output_text)

        Keylogger_Detector(output_text)

        reverse_shell_thread = threading.Thread(target=Check_For_Reverse_Shell, args=(output_text,), daemon=True)
        reverse_shell_thread.start()

        sniff(prn=lambda packet: DoS_Detection(output_text), store=False, timeout=60)

1 usage
def Stop_Live_Scan(output_text):
    global stop_live_scanner_flag
    stop_live_scanner_flag = True
    output_text.insert(tk.END, "Live scan stopped.\n")
```

Figure 12- Live Scan function

The DoS detection function(see Figure 13) monitors network traffic to identify potential denial-of-service (DoS) attacks targeting specific ports. It operates as follows:

- DoS_Detection: This function detects potential DoS attacks by analyzing network packets. It retrieves the local machine's IP address and defines a packet callback function, Packet_Callback, which is invoked for each captured packet. The function checks if the packet is destined for the local machine and if its destination port matches any commonly targeted ports for DoS attacks.
- If a match is found, the function tracks the number of packets received from each source IP address. If the number of packets from a single source IP exceeds a predefined threshold, it raises a potential DoS attack alert, providing details such as the source IP address and the targeted port.
- The function uses the sniff function from the Scapy library to capture network packets, running for a specified timeout period.
- Error handling is implemented to log any exceptions encountered during execution.

This DoS detection mechanism enhances system security by providing real-time alerts for potential DoS attacks on critical network ports.

```
def DoS_Detection(output_text):
    try:
        # Define a list of commonly targeted ports for DoS attacks
        dos_ports = target_ports  # Example ports commonly targeted for HTTP and HTTPS DoS attacks

        # Get the IP address of the local machine
        local_ip = socket.gethostbyname(socket.gethostname())

        # Initialize a dictionary to store counts of packets per source IP address
        packet_counts = {}

        # Packet callback function
        def Packet_Callback(packet):
            if packet.haslayer(IP) and packet[IP].dst == local_ip:
                src_ip = packet[IP].src
                dst_port = packet[IP].dport

                # Check if the packet's destination port matches any of the DoS target ports
                if dst_port in dos_ports:
                    if src_ip in packet_counts:
                        packet_counts[src_ip] += 1
                    else:
                        packet_counts[src_ip] = 1

                    # Check if the number of packets from a single source IP exceeds a threshold
                    dos_threshold = 100  # Adjust as needed based on expected traffic
                    if packet_counts[src_ip] >= dos_threshold:
                        dos_message = f"Potential DoS attack detected from {src_ip} on port {dst_port}"
                        output_text.insert(tk.END, str(dos_message) + "\n")
                        logging.warning(dos_message)

        sniff(prn=Packet_Callback, store=False, timeout=60)

    except Exception as e:
        logging.error( msg: f"Error in DoS detection: {e}", exc_info=True)
```

Figure 13- DoS Detection

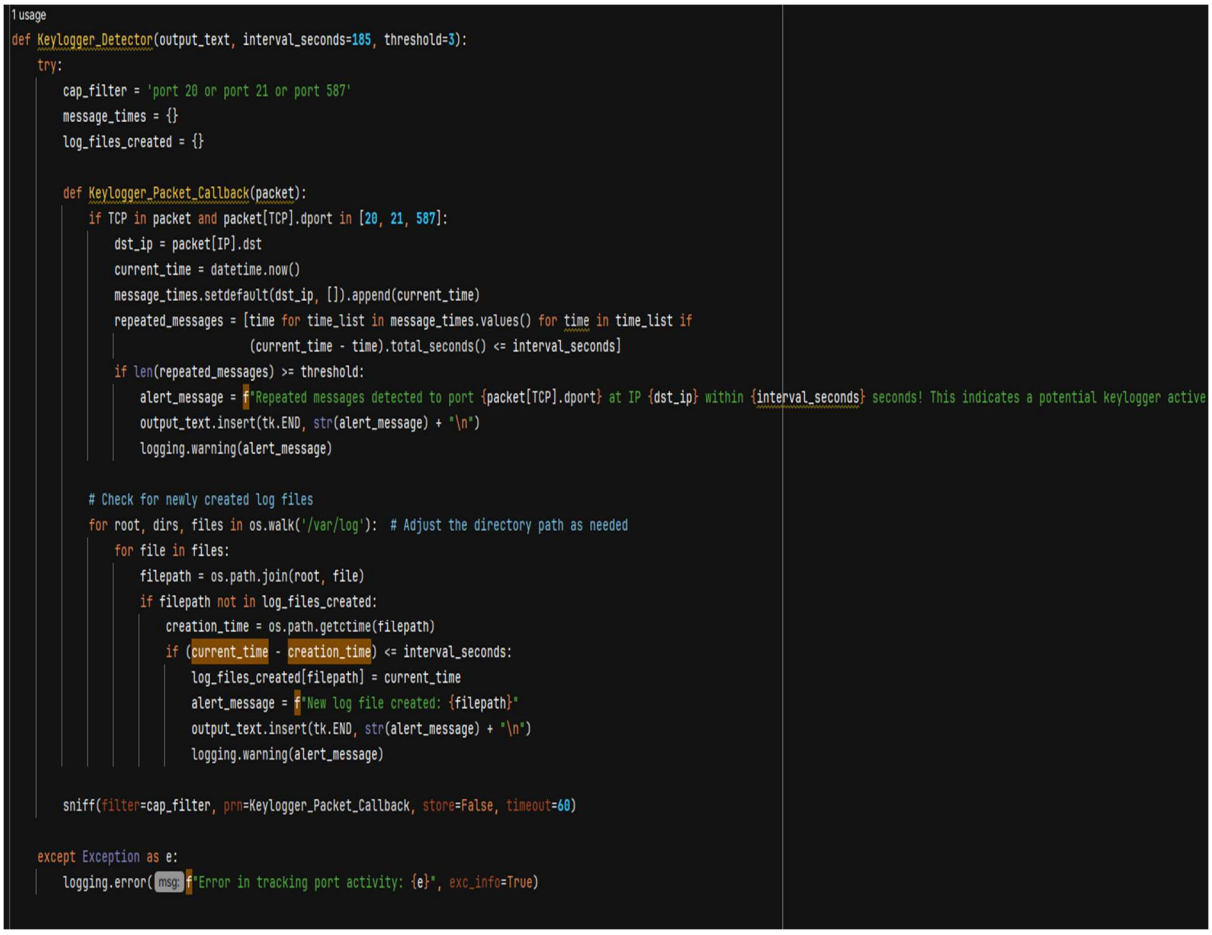
The Keylogger detection function(see Figure 14) monitors for repeated interactions with a specific IP address from the email and file transfer ports with a set time period. The functions of the process are:

- Keylogger_Detector: This function is the entry point of the program. It sets up the necessary configurations and starts monitoring for keylogger activity. The function accepts parameters such as the output text widget for displaying alerts, the interval for monitoring network traffic, and the threshold for identifying repeated messages.
- Keylogger_Packet_Callback: This nested function is called for each network packet captured by the packet sniffer. It checks for TCP packets with destination ports commonly associated with FTP (port 20, 21) and email (port 587) protocols, as these ports are often targeted by keyloggers. It records the timestamps of messages sent to these ports and alerts the user if a threshold number of messages are sent within a specified interval, indicating potential keylogging activity. Additionally, it monitors the creation of new log files in the /var/log directory, as keyloggers often write

captured keystrokes to log files. If a new log file is created within the specified interval, it alerts the user.

- sniff: This function from the scapy library is used to capture network packets based on a given filter (cap_filter). It calls the Keylogger_Packet_Callback function for each captured packet and sets a timeout for packet capturing.

The program continuously monitors network traffic and file system changes to detect potential keylogging activity in real-time. When suspicious activity is detected, it alerts the user through the output text widget and logging mechanism.



```
1 usage
def Keylogger_Detector(output_text, interval_seconds=185, threshold=3):
    try:
        cap_filter = 'port 20 or port 21 or port 587'
        message_times = {}
        log_files_created = {}

        def Keylogger_Packet_Callback(packet):
            if TCP in packet and packet[TCP].dport in [20, 21, 587]:
                dst_ip = packet[IP].dst
                current_time = datetime.now()
                message_times.setdefault(dst_ip, []).append(current_time)
                repeated_messages = [time for time_list in message_times.values() for time in time_list if
                                     (current_time - time).total_seconds() <= interval_seconds]
                if len(repeated_messages) >= threshold:
                    alert_message = f"Repeated messages detected to port {packet[TCP].dport} at IP {dst_ip} within {interval_seconds} seconds! This indicates a potential keylogger active
                    output_text.insert(tk.END, str(alert_message) + "\n")
                    logging.warning(alert_message)

            # Check for newly created log files
            for root, dirs, files in os.walk('/var/log'):  # Adjust the directory path as needed
                for file in files:
                    filepath = os.path.join(root, file)
                    if filepath not in log_files_created:
                        creation_time = os.path.getctime(filepath)
                        if (current_time - creation_time) <= interval_seconds:
                            log_files_created[filepath] = current_time
                            alert_message = f"New log file created: {filepath}"
                            output_text.insert(tk.END, str(alert_message) + "\n")
                            logging.warning(alert_message)

        sniff(filter=cap_filter, prn=Keylogger_Packet_Callback, store=False, timeout=60)

    except Exception as e:
        logging.error(msg=f"Error in tracking port activity: {e}", exc_info=True)
```

Figure 14- Keylogger Detection

The packet sniffer detecting function(see Figure 15), by identifying potential signs of sniffing on the system. It employs several techniques to detect such activities and alerts the user accordingly. The main elements are of this function are as follows:

- Packet_Sniffer_Detection: This function serves as the main entry point for packet sniffing detection. It checks for popular packet sniffing processes and raw socket creations, and it also looks for signs of promiscuous mode on network interfaces.
- sniffing_processes: This list contains the names of popular packet sniffing processes such as Wireshark, tcpdump, and WinPcap. The function iterates over the running processes on the system and compares their names against this list to identify potential packet sniffers.

- raw_sockets: This list stores information about processes that have created raw sockets, which can be indicative of packet sniffing activities. It checks for processes creating raw sockets on both Linux and Windows systems.
- network_interfaces: This variable holds information about network interfaces on the system. The function examines each interface for signs of promiscuous mode, which is often used by packet sniffers to capture network traffic.
- Also detects processes with raw sockets and elevated network privileges

If any suspicious activity is detected during the process, the function logs a warning message and alerts the user through the output text widget.

```
def Packet_Sniffer_Detection(output_text):
    try:
        # List of popular packet sniffing processes/applications to check for
        sniffing_processes = ["Wireshark.exe", "tcpdump.exe", "WinPcap.exe"]
        for proc in psutil.process_iter(["name"]):
            if proc.info["name"] in sniffing_processes:
                alert_message = f"Potential packet sniffing process detected: {proc.info['name']}"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)
        raw_sockets = []        # Check for raw socket creation on both Linux and Windows
        for proc in psutil.process_iter(['pid', 'name']):
            try:
                connections = proc.connections(kind='all')
            except (psutil.NoSuchProcess):
                continue
            for conn in connections:
                if conn.family in (psutil.AF_INET, psutil.AF_INET6) and conn.type == socket.SOCK_RAW:
                    raw_sockets.append((proc.pid, conn.fd))
                    break
        if raw_sockets:
            alert_message = "Raw socket(s) created by program(s):"
            for pid, fd in raw_sockets:
                alert_message += f"\nPID: {pid}, FD: {fd}"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
        # Check for promiscuous mode
        network_interfaces = psutil.net_if_addrs()
        for interface, addresses in network_interfaces.items():
            for address in addresses:
                if address.family == psutil.AF_LINK and address.address == "00:00:00:00:00:00":
                    alert_message = f"Promiscuous mode detected on interface: {interface}"
                    output_text.insert(tk.END, str(alert_message) + "\n")
                    logging.warning(alert_message)
        for proc in psutil.process_iter(): # Check for suspicious network privileges
            try:
                if proc.connections(kind='inet') and proc.username() != 'SYSTEM':
                    alert_message = f"Suspicious process {proc.name()} with raw sockets or elevated network privileges!"
                    output_text.insert(tk.END, str(alert_message) + "\n")
                    logging.warning(alert_message)
            except psutil.AccessDenied:
                continue
```

Figure 15- Packet Sniffer Detection

The reverse shell detection function(see Figure 16) continuously monitors network connections and alerts the user if suspicious outbound connections are detected. The key functions of this program are:

- Is_IPAdress_Private: This function checks whether an IP address is private or not. It utilizes the ipaddress module to validate the IP address and determine if it belongs to a private IP address range.
- Check_For_Reverse_Shell: This function continuously monitors network connections using the psutil module. It iterates through all network connections and checks for outbound connections that are not to common service ports (e.g., port 80 for HTTP, port 443 for HTTPS) and not originating from localhost (127.0.0.1). If such a connection is found with a non-private remote IP address, it raises an alert indicating a potential reverse shell.

The program runs in a loop, periodically checking for new network connections every 3 seconds.

```
1 usage
def Is_IPAdress_Private(ip):
    try:
        return ipaddress.ip_address(ip).is_private
    except ValueError:
        return False

1 usage
def get_my_ip():
    return socket.gethostbyname(socket.gethostname())

1 usage
def Check_For_Reverse_Shell(output_text):
    my_ip=get_my_ip()
    try:
        while not stop_live_scanner_flag:
            # Get all network connections
            connections = psutil.net_connections(kind='inet')

            # Check for outbound connections
            for conn in connections:
                if conn.status == psutil.CONN_ESTABLISHED:
                    local_ip, local_port = conn.laddr
                    remote_ip, remote_port = conn.raddr

                    # Check if the remote IP address is private-will ca
                    if Is_IPAdress_Private(remote_ip) and remote_ip != my_ip:
                    # Check if the connection is outbound and not to a common service port-127.0.0.1 is localhost. port 80 is HTTP and 443 is HTTPS
                        if local_ip != '127.0.0.1' and remote_ip != '127.0.0.1' and remote_port not in [80, 443]:
                            alert_message = f"Potential reverse shell detected: {local_ip}:{local_port} -> {remote_ip}:{remote_port}"
                            output_text.insert(tk.END, str(alert_message) + "\n")
                            logging.warning(alert_message)

            time.sleep(3)
    except Exception as e:
        logging.error( msg: f"Error in reverse shell detection: {e}", exc_info=True)
```

Figure 16- Reverse Shell Detection

The last part of the signature detection incorporated into the live process monitoring function is for data theft which checks for signs of data/files being moved, deleted, created, or modified(see Figures 17.1 and 17.2). It can also detect files that have been copied. These are focused on the directory path "C:\Users\ (current user) \OneDrive\Documents". This was chosen due to this being the main folder path for a user's files so the most likely to be targeted. It utilizes the FileSystemEventHandler class from the watchdog library to handle file system events such as file creation, deletion, modification, and movement. The main functions of the program include:

- FileTransferHandler(Figure 17.1): This class inherits from FileSystemEventHandler and overrides its methods to define custom behavior for each type of file system event. It includes methods such as on_created, on_deleted, on_modified, and on_moved. When an event is triggered, an alert message is generated and displayed in the output text widget. Additionally, each alert message is logged for record-keeping purposes.
- start_file_monitoring(Figure 17.2): This function initiates the file monitoring process by creating an instance of FileTransferHandler and setting up an observer object from the watchdog library. The observer is configured to monitor the specified base directory and its subdirectories recursively. Once started, the observer continuously watches for file system events within the monitored directory.

```
class FileTransferHandler(FileSystemEventHandler):
    def __init__(self, output_text):
        super().__init__()
        self.output_text = output_text

    def on_created(self, event):
        if isinstance(event, FileCreatedEvent):
            alert_message = f"Alert! File created: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_deleted(self, event):
        if isinstance(event, FileDeletedEvent):
            alert_message = f"Alert! File deleted: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_modified(self, event):
        if isinstance(event, FileModifiedEvent):
            alert_message = f"Alert! File modified: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_moved(self, event):
        if isinstance(event, FileMovedEvent):
            alert_message = f"Alert! File moved: {event.src_path} -> {event.dest_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
```

Figure 17.1- File Modification Detection

```
def start_file_monitoring(base_directory, output_text):
    current_user = os.getlogin()
    base_directory = os.path.join("C:\\Users", current_user, "OneDrive", "Documents")
    event_handler = FileTransferHandler(output_text)
    observer = Observer()
    observer.schedule(event_handler, base_directory, recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

Figure 17.2- File Modification Detection

There is also some anomaly detection built into the live monitoring process(see Figures 18.1-18.3). It will alert the user when an unrecognized program is being run on the system and when the CPU/RAM usage is higher than normal. Both functions can be customized. Normal programs to be ignored can be added to the list and the normal CPU/RAM threshold can be changed. The functions involved in this are:

- Read_Standard_Actions(Figure 18.1): This function reads a text file named "standard_actions.txt" to obtain a list of acceptable actions regularly accessed by the PC. It returns a list of these standard actions.
- Anomaly_Action_Checking(Figure 18.1): This function checks for any actions occurring on the system that are not considered normal based on the standard actions listed in the "standard_actions.txt" file. It retrieves a list of running processes and compares them against the standard actions. If any anomalies are detected, an alert message is generated.
- Read_Thresholds_From_File(Figure 18.2): This function reads CPU and RAM thresholds from a configuration file named "resource_usage.txt." It extracts the thresholds from the file and returns them for further use.
- Anomaly_Resource_Usage(Figure 18.3): This function monitors resource usage on the system, including CPU and RAM. It compares the current resource usage with predefined thresholds obtained from the configuration file. If the usage exceeds the thresholds, it generates an alert message indicating abnormal usage.
- Customization: This function(see Figure 19) provides a mechanism for customization by the user. It prompts the user to enter a password for authentication. Upon successful authentication, the user can add new standard actions or adjust CPU and RAM thresholds. New standard actions are appended to the "standard_actions.txt" file, and the new thresholds are updated in the "resource_usage.txt" file.

```python
def Read_Standard_Actions():#reads standard_actions.txt to get a list of acceptable actions by the pc(applications regularly accessed by PC.
    standard_actions = []
    file_path = "standard_actions.txt"

    if os.path.exists(file_path):
        with open(file_path, 'r') as file:
            standard_actions = [line.strip() for line in file.readlines()]

    return standard_actions

standard_actions = Read_Standard_Actions()

1 usage
def Anomaly_Action_Checking(output_text):#Checks for actions occuring that aren't normal to PC based on standard_actions.txt
    try:
        running_processes = [proc.name().lower() for proc in psutil.process_iter(attrs=['name'])]

        # Load standard actions from the file
        with open('standard_actions.txt', 'r') as file:
            standard_actions = set(file.read().splitlines())

        # Check if any process is not in the list of standard actions
        anomalies = [process for process in running_processes if process not in standard_actions]

        if anomalies:
            alert_message = f"Alert! Anomaly detected. Unexpected processes running:{', '.join(anomalies)}"#edited out bit which lists anomalies
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

    except Exception as e:
        logging.error( msg: f"Error in anomaly-based process running detection: {e}", exc_info=True)
    return False
```

Figure 18.1- Anomaly Detection

```python
def Read_Thresholds_From_File():
    # Get the current directory
    current_directory = os.getcwd()

    # Construct the absolute path to the file
    file_path = os.path.join(current_directory, "resource_usage.txt")

    try:
        with open(file_path, "r") as file:
            lines = file.readlines()
            # Extract CPU threshold from the first line and RAM threshold from the second line
            cpu_threshold = float(lines[0].split(':')[1].strip())
            ram_threshold = float(lines[1].split(':')[1].strip())
            return cpu_threshold, ram_threshold
    except FileNotFoundError:
        # Handle the case when the configuration file is not found
        print("Configuration file not found.")
        return None, None
    except Exception as e:
        # Handle other exceptions (e.g., invalid format)
        print(f"Error reading configuration file: {e}")
        return None, None
```

Figure 18.2- Anomaly Detection

```python
def Anomaly_Resource_Usage(output_text):
    try:
        # Read CPU and RAM thresholds from the configuration file
        cpu_threshold, ram_threshold = Read_Thresholds_From_File()

        if cpu_threshold is None or ram_threshold is None:
            return False  # Return False if thresholds couldn't be read

        # Get CPU usage percentage
        cpu_usage = psutil.cpu_percent(interval=1)
        # Get memory usage percentage
        memory_usage = psutil.virtual_memory().percent

        # Check if CPU usage exceeds the threshold
        if cpu_usage > cpu_threshold:
            alert_message = f"Alert! Abnormal CPU usage detected: {cpu_usage}%"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

        # Check if memory usage exceeds the threshold
        elif memory_usage > ram_threshold:
            alert_message = f"Alert! Abnormal memory usage detected:  {memory_usage}%"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

    except Exception as e:
        logging.error( msg: f"Error in anomaly-based resource usage detection: {e}", exc_info=True)

    return False
```

Figure 18.3- Anomaly Detection

```
def Customization(output_text):
    try:
        # Add standard actions
        password = tk_simpledialog.askstring(title: "Password", prompt: "Enter password:", show='*')
        if password == "admin":
            new_processes = tk_simpledialog.askstring(title: "Add Standard Actions", prompt: "Enter process names separated by commas:")
            if new_processes:
                # Split the input by commas and add each process individually
                new_processes_list = [process.strip() for process in new_processes.split(',')]
                for new_process_name in new_processes_list:
                    standard_actions.append(new_process_name.lower())
                    output_text.insert(tk.END, f"Added '{new_process_name}' to standard actions.\n")
                    with open("standard_actions.txt", 'a') as file:
                        file.write(f"{new_process_name}\n")

        # Change CPU/RAM thresholds
        cpu_threshold = tk_simpledialog.askfloat(title: "Change CPU Threshold", prompt: "Enter CPU threshold (0-100):", minvalue=0, maxvalue=100)
        ram_threshold = tk_simpledialog.askfloat(title: "Change RAM Threshold", prompt: "Enter RAM threshold (0-100):", minvalue=0, maxvalue=100)

        # Write the new thresholds to the configuration file
        with open('resource_usage.txt', 'w') as file:
            file.write(f"CPU Threshold: {cpu_threshold}\n")
            file.write(f"RAM Threshold: {ram_threshold}\n")

        output_text.insert(tk.END, f"Customization successful. CPU Threshold changed to {cpu_threshold}% and RAM Threshold changed to {ram_threshold}%.\n")

    except Exception as e:
        output_text.insert(tk.END, f"Error during customization: {e}\n")
```

Figure 19- Customization function

This is all tied together through a GUI(shown in Appendix A) which has buttons for selecting directories to check for malicious scripts and encrypted files. There are also buttons for starting and stopping the live scan function. There is also a “Customization” button which opens a tab for the CPU/RAM usage modifying and normal-process-adding functions and an button to clear the display section.

5.2)Prototypes/Previous Iterations

I explored several options of how to run specifically the signature detection aspect of the code. One of them involved having a directory of malware, acquired from Exploit Database and the GitHub repository “theZoo” which contains a collection of malware samples from preexisting malware (including viruses, worms, trojans and ransomware) that has previously been used by threat actors, for research and educational purposes (ytisf, 2014). This directory was called “MalwareBox”. The original plan was that the program would read from MalwareBox and be able to extract the signature from the scripts within MalwareBox and build an idea from that of how to detect malware. There was also an attempt to use a Virus Total API, but this didn’t add much to the original idea as it relied on the malware being the same ones that it had previously seen and had been publicly used (i.e. Locky Ransomware-ransomware malware released in 2016. It is delivered by email (that is allegedly an invoice

requiring payment) with an attached Microsoft Word document that contains malicious macros. On February 18, 2016, the Hollywood Presbyterian Medical Center paid a $17,000 ransom in the form of bitcoins for the decryption key for patient data. The hospital was infected by the delivery of an email attachment disguised as a Microsoft Word invoice.)

In relation to the live malware detected, I also attempted to put functions that conducted malicious actions into a text file and have the main program read from. This concept didn't work as I thought it would in reality, as it required more than just a function to read scripts and automatically understand what they did and instead needed a whole machine learning system built into the program to read the scripts, extract exactly what it was that made them do whatever malicious thing they did(packet sniffing, keylogging etc) and then learn to recognise it so that it could detect others.

6) Testing

6.1)Introduction

To assess that the program worked, I ran a series of tests on the program. One was to make malware scripts to determine the detection accuracy of the malicious script detector. It was able to locate these scripts from within directories and distinguish from regular scripts. I also assessed each feature with python script which replicated malware, shell commands and packet sniffing applications and then recorded the outcome. These malware python scripts were used to compared to existing Python intrusion detection scripts found on the internet. The scripts were mostly acquired from the books "Black Hat Python" by Justin Seitz and Tim Arnold (Seitz & Arnold, 2021) and "Ethical Hacking with Python" by The Python Code (The Python Code, 2023).I also used real-world programs taken from places such as theZoo.

6.2)Scripts and Software used for Testing

To determine that the script could detect malware, I created a series of scripts in Python, Java and C which could be detected by the program. I also copied some of these into word documents for testing that they could be detected.

6.2.1)Ransomware

For the ransomware test, I used a Python script from The Python Code book that enables file encryption and decryption using symmetric key cryptography. Upon execution, it prompts the user to enter a password and the name of the file to be processed. The user is then asked whether they want to encrypt the file. If encryption is chosen, a key is derived from the password and optionally saved to a file. The file is then encrypted using the derived key and saved. The script utilizes the Fernet symmetric encryption scheme from the cryptography library, and it employs Scrypt for key derivation to enhance security. I also used WannaCry(downloaded from the Zoo), which is a real example of ransomware that encrypts

files on a computer and demands payment for decryption; it was released in May 2017 and targeted various organizations worldwide, including healthcare and government sectors.

6.2.2)Keylogger

For the keylogger test, I used a script from the Python Code book that captures keystrokes and either emails them or saves them to a local file, depending on the chosen reporting method. The Keylogger class initializes with an interval for reporting keystrokes, along with a specified reporting method. It sets up a callback function to capture keyboard events using the keyboard module. Special keys are translated into readable formats, and keystrokes are appended to a log. At each reporting interval, the log is sent via email or saved to a file. I also used “Best Free Keylogger” which tracks the user’s keystrokes and reports them (Best X Software, 2024).

6.2.3)Packet Sniffer

For the packet sniffer test, I used a script acquired from Black Hat Python which was a network utility designed for packet sniffing and raw packet transmission using raw sockets. It sets up a raw socket and captures incoming packets. It then parses the IP header of each captured packet and prints out information about the protocol and the source and destination IP addresses. The main execution section allows the script to be run directly from the command line, specifying a host IP address to sniff on, defaulting to my IP address if not provided. The script demonstrates fundamental functionalities for network monitoring and packet manipulation. I also used Wireshark, a software tool used for network analysis and troubleshooting. It allows users to capture and inspect data packets traveling across a computer network.

6.2.4)DoS

I used 2 scripts from GitHub. The first initiates an infinite loop within a threading environment, continuously creating socket connections to a previously specified target IP and port. With each connection, it sends a crafted HTTP request consisting of a GET request and a host header containing the fake IP address. This process repeats in multiple threads concurrently, simulating a flood of traffic directed at the target (depascaldc, 2020). I also used one which begins by obtaining the target host's IP address and user-defined parameters such as the port to target, the number of hits per run, and the thread count. Then, within a loop, it establishes a TCP connection to the target server and sends HTTP GET requests with a randomly generated payload. This process continues indefinitely until interrupted by the user or an exception occurs (z9fr, 2021).

6.2.5)Anomaly RAM usage

I utilized the turtle module to open a tab to draw an extensive geometric pattern. This process on its own, doesn’t do anything malicious, but the program is designed to do this 1000 times simultaneously which puts a strain on the RAM similarly to how a virus like spam or adware can cause multiple tabs to open which slows down the PC.

6.2.6)Reverse Shell

For the reverse shell detection, I used a client-server architecture from the Python Code book for remote command execution. The server listens for incoming connections on a specified IP address and port. Once a connection is established, it receives the current working directory from the client. The server then continuously prompts the user for commands, sends them to the client, and receives the command output along with the updated current directory from the client. The client, upon connecting to the server, sends its current working directory to the server and then enters a loop where it receives commands from the server, executes them (with special handling for "cd" commands to change directories), and sends back the output along with the updated current directory to the server. The communication between the client and server is facilitated using sockets and a defined separator string. The server and client are meant to be run separately, with the server script taking an optional command line argument for the server host. I also used “reverse_tcp.md”. This is a software payload that enables the execution of reverse shells. It can be downloaded through Metasploit, a computer security project used to assess the security of computer systems by simulating cyber-attacks.

6.2.7)File modifications

To assess the file modifications detection, I simply moved/copied/deleted files using control characters commands as well as a script which transfers files from one directory to another. The script defines a function, transfer_files, which transfers files and directories from a source directory to a destination directory. It creates the destination directory if it does not exist and then iterates through each item in the source directory, copying directories using shutil.copytree and files using shutil.copy2 to the destination directory. Finally, it prints a message indicating the completion of the file transfer process.

The scripts made for reverse shell, ransomware, DoS packet sniffers and keyloggers were also used, along with similar scripts, as part of the malicious script finder function test.

6.3)Devices used

1. ASUS ZenBook 14 with 8GB RAM and an Intel(R) Core(TM) i5-1135G7 x64 processor
2. Acer Swift 1 with 4GB RAM and an Intel(R) Celeron(R) N4000 x64 processor.

The Acer laptop was used to run the Intrusion Detection System for the DoS and Reverse Shell attacks which required 2 devices. The attacks were initiated from the ZenBook in that instance. Then in other instances, both devices would run the program and have the other malware programs running in the background to be detected. This was to simulate a live attack. This was run on both machines to confirm that the program worked on more than one computer.

7) Product Evaluation

7.1)Comparison to existing applications

After completing the testing of my own source code. I conducted the same tests against some of the programs shown in the "Similar Products" as well as some found on GitHub and recorded the outcome in a table. Here is a brief description of each of the new scripts:

- "dos_ddos_detector" by umarjatt420(Umar Jatt)-Continuously sniffs network packets, extracting the source IP address from each packet, and maintains a count of packets received from each unique source IP. If the count exceeds a predefined threshold, it triggers an alert indicating a Dos/DDoS attack originating from that source IP (Jatt, 2024).

- "E-Guard Keylogger Detector" by aelder202(Taylor Elder)-Detects potential keylogging activity by monitoring network connections and associated processes. It first checks for TCP connections on specific ports commonly associated with email communication (such as 587, 465, and 2525) using the 'netstat' command. Once a connection is detected, it extracts the Process ID (PID) of the associated process. Then, it retrieves the name of the process using the 'tasklist' command, specifically targeting the process with the obtained PID. If the process is not whitelisted, it prompts the user to decide whether to whitelist the application, terminate it, or add it to the blacklist based on perceived threat (Elder, 2022).

- "Reverse-Shell-Detector" by Mehmetali Bellur(bellurm)-Sets up a socket server to listen for incoming connections on port 8080. Upon connection, it continuously receives data from the client and checks for predefined strings associated with common reverse shell commands. If any of these strings are detected in the incoming data, it alerts the user that a reverse shell may be in progress. Finally, it responds to the client with a message indicating whether a reverse shell was detected or not before closing the connection and terminating the server (Bellur, 2023).

- "enc_det" by Nazarii Plesak(crslv0)-Detects potentially encrypted or compressed files within a specified directory. It calculates the entropy of each file's content to determine the level of randomness, which is then converted into a confidence percentage. Users can set a confidence threshold, and the script will output files exceeding this threshold, optionally sorted in descending or ascending order by confidence level (Plesak, 2021).

- "Python Anti-Keylogging Tool" by Adam Chemiron (Nada18-sec)-Utilizes PySimpleGUI for the user interface, PyFiglet for ASCII art text formatting, and the

tqdm library for progress bar functionality during scanning. The script first prompts the user to initiate a system scan, then iterates through the list of running processes, comparing them against a set of predefined indicators of compromise (IOC) stored in a JSON file, and terminates any processes identified as potential keyloggers (Chemiron, 2022).

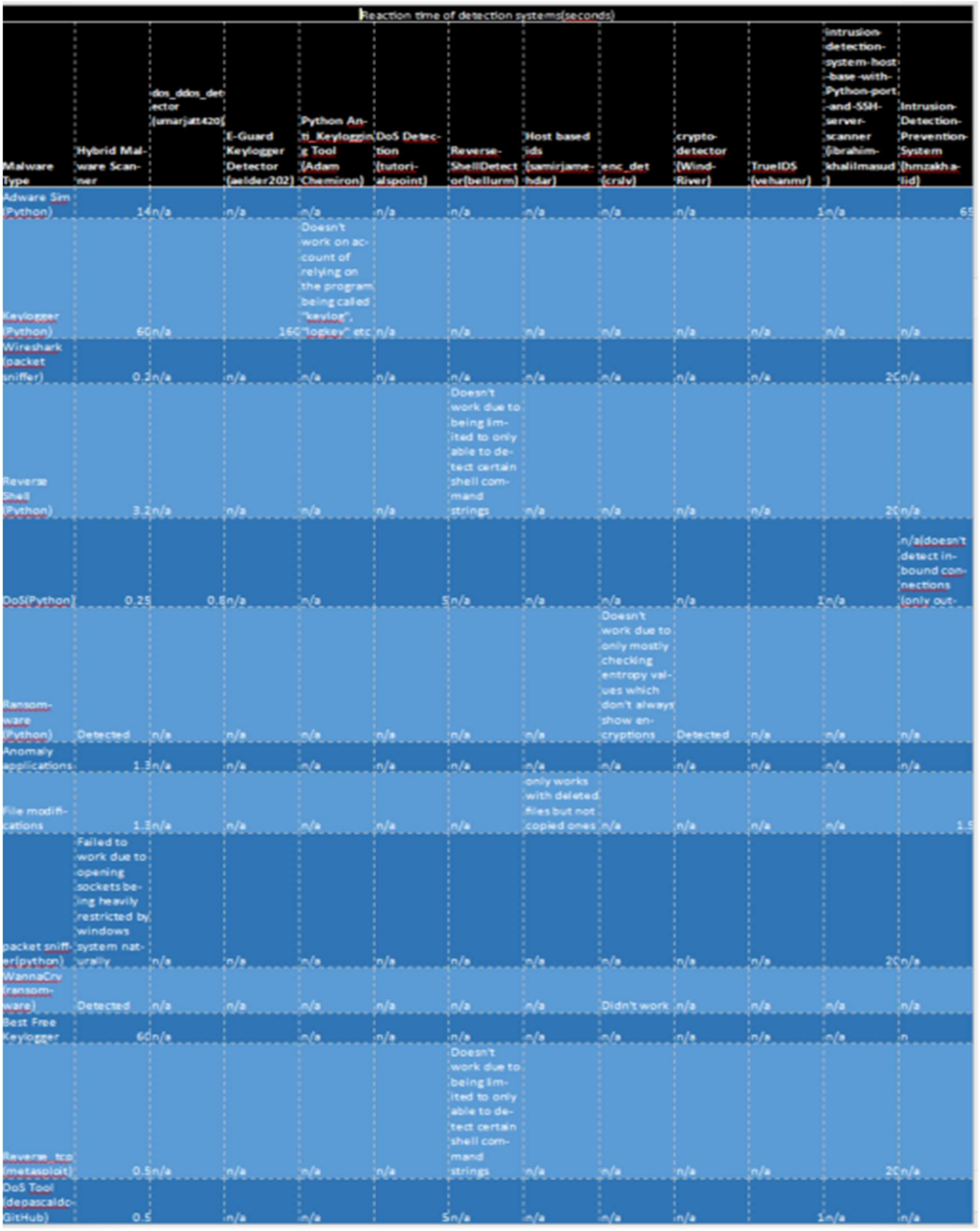
- "crypto-detector" by Joe Slater(Wind-River)-It utilizes various methods to scan files within specified packages for cryptographic patterns or evidence including keywords related to encryption found in the file (Slater, 2020)

- There was also a DoS detector I found on a computing educational website, "Tutorialspoint" which detects DDoS attacks. It utilizes the Scapy library to continuously monitors network traffic using a packet callback function, extracting source IP addresses from incoming packets. If the number of packets originating from a particular IP exceeds a predefined threshold, indicative of a potential DDoS attack, the script logs the IP address as a suspected attacker in a text file along with a timestamp of detection (Tutorialspoint, n.d.).

VirusTotal is a free online service that analyses files and URLs to detect malware and other types of malicious content. It aggregates results from various antivirus engines and other malware detection tools to provide users with comprehensive information about the safety of a file or website provided by users. I uploaded all my testing scripts to VirusTotal, but it failed to identify any of them due them not being made using known attack signatures/patterns and not being included in the databases of the antivirus engines (VirusTotal, 2012).

7.2)Testing Results and Product Effectiveness

Based on the testing done, I think that my code is very efficient. All elements work smoothly within the system to detect threats. The results of the test, demonstrated by the table(Table 1), which offers explanations for why certain programs didn't work, and the graph(Figure 20), which creates a visual image of how different programs compared at detecting threats, displayed below, indicate that my program is clearly superior in relation to speed of detection compared to some of the existing systems and can detect a wide range of threats.



Reaction time of detection systems(seconds)

Malware Type	Hybrid Malware Scanner	ddos_ddos_detector (umarjatt420)	E-Guard Keylogger Detector (aelder202)	Python Anti_Keylogging Tool (Adam Chemiron)	DoS Detection (tutorialspoint)	Reverse-ShellDetector(bellurm)	Host based ids (samirjamedhdar)	enc_det (crslv)	crypto-detector (WindRiver)	TrueIDS (vehanmr)	intrusion-detection-system-host-base-with-Python-port-and-SSH-server-scanner (ibrahimkhalilmasud)	Intrusion-Detection-Prevention-System (hmzakhaid)
Adware Sim (Python)	14	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	1	n/a	6
Keylogger (Python)	60	n/a	160	Doesn't work on account of relying on the program being called "keylog", "logkey" etc	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Wireshark (packet sniffer)	0.2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	2	n/a
Reverse shell (Python)	3.2	n/a	n/a	n/a	n/a	Doesn't work due to being limited to only able to detect certain shell command strings	n/a	n/a	n/a	n/a	2	n/a
DoS(Python)	0.25	0.8	n/a	n/a	5	n/a	n/a	n/a	n/a	1	n/a	n/aldoesn't detect inbound connections (only out-
Ransomware (Python)	Detected	n/a	n/a	n/a	n/a	n/a	n/a	Doesn't work due to only mostly checking entropy values which don't always show encryptions	Detected	n/a	n/a	n/a
Anomaly applications	1.3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
File modifications	1.3	n/a	n/a	n/a	n/a	n/a	only works with deleted files but not copied ones	n/a	n/a	n/a	n/a	1.5
packet sniffer(python)	Failed to work due to opening sockets being heavily restricted by windows system naturally	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	2	n/a
WannaCry (ransomware)	Detected	n/a	n/a	n/a	n/a	n/a	n/a	Didn't work	n/a	n/a	n/a	n/a
Best Free Keylogger	60	n/a		n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n
Reverse_tcp (metasploit)	0.5	n/a	n/a	n/a	n/a	Doesn't work due to being limited to only able to detect certain shell command strings	n/a	n/a	n/a	n/a	2	n/a
DoS Tool (deopascaldc-GitHub)	0.5		n/a	n/a	5	n/a	n/a	n/a	n/a	1	n/a	n/a

Table 1- Table comparing detection programs and their reaction times

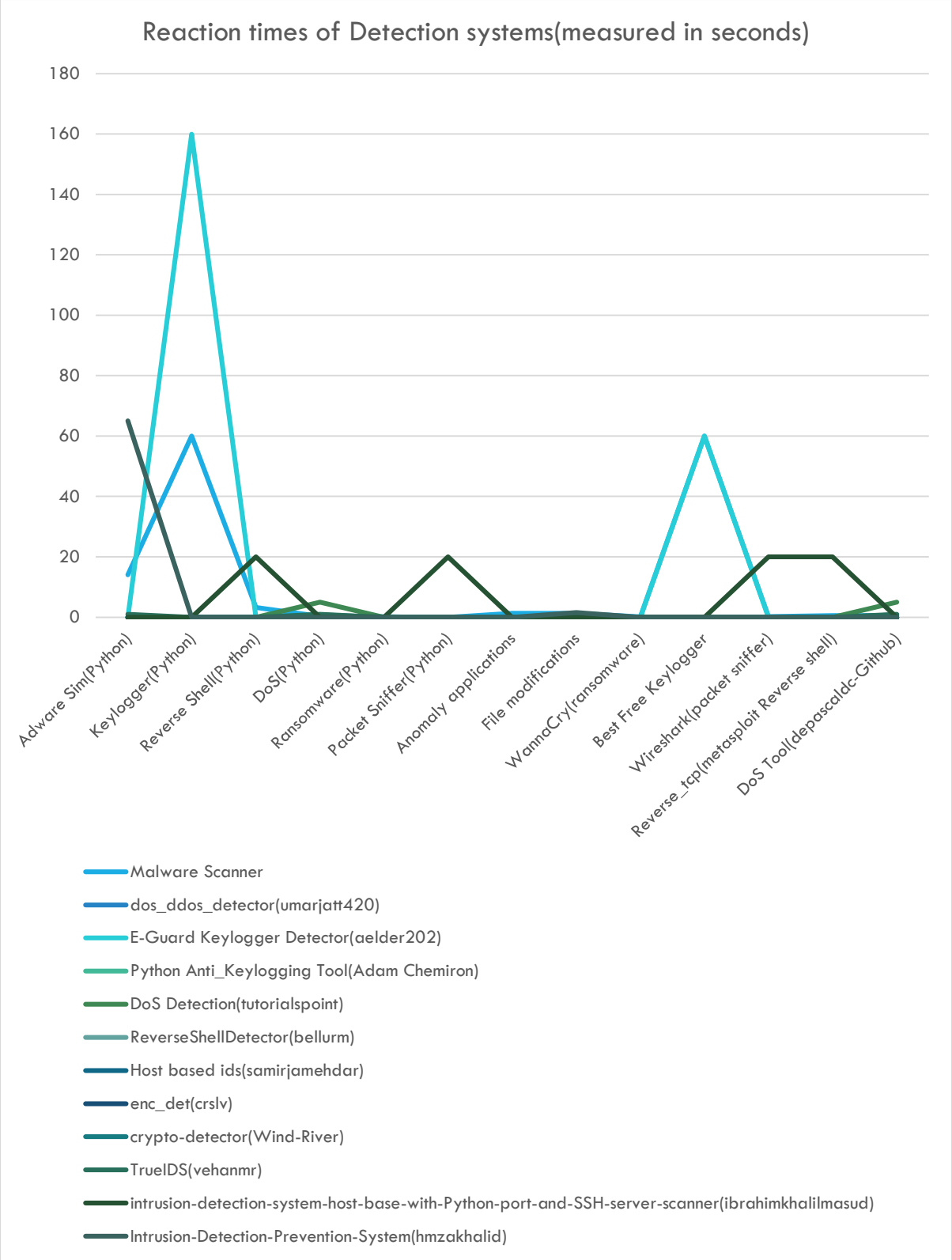


Figure 20- Graph comparing detection programs and their reaction times

7.3)Weaknesses

There are issues with certain functions of the program. For example, in detecting reverse shells. The program checks for any connections made between the host device and a private IP address. However, a standard device such as a printer could set this off so there would need to be checked to determine that this indeed a threat. To refine the function, you could implement additional checks to exclude known legitimate private IP addresses, such as those assigned to local network devices like printers, routers, and IoT devices. Also, as highlighted by the graph, there is a relatively long time period needed to detect a keylogger. This could be fixed by having a more detailed system for detecting keyloggers which could check for slow browser speeds and a lag in keystrokes and mouse movements as potential indicators. Additionally, consider incorporating a whitelist mechanism to allow specific private IP addresses that are deemed safe. There is also the possibility with the malware script finder where a script could be made without any of the keywords, also this is unlikely based on how important and niche to making malware the words are. For example, making a packet sniffer without “sniff”, “scapy” or “pcapy” would not be possible as these are functions that are key for the network monitoring/packet capturing that packet sniffers do.

There is also analysis of the host device needed prior to setting up the program to determine what would be considered standard RAM/CPU usage to properly ensure it is detecting threats and not detecting the standard processes of the user that can trigger high resource usage such as playing computer games and streaming videos. This can be tracked using system monitoring applications such as Windows Task Manager, Pandora FMS and MyASUS.

8) Final Conclusion and Reflections

8.1)Concluding thoughts

To begin with, my aim was to make a program which could solve security issues within the application layer and be an improvement of existing systems. The testing indicates that this was a success. In this project, I created a malware scanner that had the ability to scan directories for malware and monitor live processes for malicious attacks. After extensive research into what each of the malware attacks I targeted did, and what signatures they might have to be detected, I developed both functions using Python, a programming language which I feel confident and adept at using. I used Windows operating system for the testing as this is the most popular operating system for personal computers and can easily run Python IDEs.

After developing the scanner, I did various tests on it to determine its effectiveness. I sent DoS attacks from a second PC and ran packet sniffers, keyloggers and on the target to see if they would be detected. I also initiated a reverse shell connection. I did all these things with 2 different methods to demonstrate the program’s efficacy. The testing was highly successful as the program was able to detect all my attack attempts and, in some cases, with a greater speed than that of similar programs I found in my research.

Overall, I quite enjoyed the project. To produce a suitable project was a challenging, complicated process which required extensive research but upon putting together a suitable plan and outline for a project, the next stages were quite simple. Reading over similar projects and articles about detecting malware was quite a lengthy process and satisfying when an appropriate method that I could implement was discovered. The testing part was also quite interesting and gave me an understanding of the process undertaken by software developers which was quite fascinating.

At the start of the development process, I had a series of functional requirements which my program needed to fulfil. All of these were met as follows:

- File Scanning: The program can detect a variety of malware across file types and programming languages.
- Live Process Monitoring: The scan can run indefinitely and detect all forms of malware in one run of the scanner.
- Anomaly-Based Intrusion Detection: The program can quickly detect anomalies in the resource usage and immediate log any potentially dangerous processes running.
- Graphical User Interface (GUI): The GUI is user-friendly and self-explanatory.
- Alerting Mechanism: The alerting mechanism is efficient. It informs user of any users with simple language.
- Configuration Options: Users can customize the program to change the CPU/RAM threshold and add “safe” applications to be ignored.
- Logging and Reporting: All scans and their results are logged and timestamped in a text file in the same folder as the application.

8.2)Future Improvements

Like much of the software that is developed every day, there were certain elements of the program which could be improved. I outline below how the program could be improved in the future.

8.2.1)Increased range of attacks detected

The program is currently limited to detecting keyloggers, DoS attacks, ransomware, data theft, reverse shells, and packet sniffers. This program could be improved by adding other frequently used application-layer attacks to be detected. These include cross-site scripting (attack in which an attacker injects malicious executable scripts into the code of a trusted application or website), SQL injections (Attackers inject malicious SQL queries into input fields of web applications, exploiting vulnerabilities in the database layer. This can lead to unauthorized access, data leakage, or data manipulation.), buffer overflows (exploit vulnerabilities in poorly coded applications to write data beyond the allocated buffer) and man-in the middle attacks (Attackers intercept communication between two parties and may modify or eavesdrop on the data exchanged).

8.2.2)Better detection of obfuscation

Obfuscation is the practice of intentionally making source or machine code difficult to comprehend for humans or computers. Currently, the malware-locating part of the program relies on the scripts that are being detected having a set of keywords common amongst malicious scripts. If the program doesn't have those keywords or somehow the source code has been covered up by the program, it will be undetected. Detecting such obfuscated code involves a complex and extensive set of algorithms, surpassing the scope of my current capabilities within the given timeframe. Additional research could enhance the malware scanning functionality of the program by enabling better recognition and analysis of obfuscated code patterns.

8.2.3)Better user interface

The user interface could be designed to be more professional looking and resemble a full-on app. This could be something that resembled the Avast One Antivirus GUI. The full GUI could be improved by the following:

- Sound alerts and notifications
- Log reports could be sent to user via email or text
- Reduce the number of times the DoS alert appears
- Change button shapes/positions

Certain elements could be done better by implementing a different GUI framework such as Pyramid, which was used to make Reddit and Dropbox, or Kivy which was used for MyPaint. I chose Tkinter as this was a format that I was already skilled in using.

8.3)Learning Outcomes

From carrying out this project, I have increased my skills in coding, gaining more understanding of certain elements of python, such as newly discovered libraries including "scapy" and "psutil" which could prove useful in future developments of network/process monitoring systems. I've also obtained more knowledge around being to detect malware and run a basic form of penetration testing by testing my system using existing malware and my own scripts. This too will prove to be invaluable as I continue to progress towards a career in information security.

References

AAG IT Services, 2024. *AAG IT.COM | The Latest 2024 Cyber Crime Statistics (updated February 2024).* [Online]
Available at: https://aag-it.com/the-latest-cyber-crime-statistics/
[Accessed 1 March 2024].

Abbasi, M., Plaza-Hernández, M., Prieto, J. & Corchado, J. M., 2022. Security in the Internet of Things Application Layer: Requirements, Threats, and Solutions. *IEEE Access,* Volume 10, pp. 97197 - 97216.

Alani, M. M., Mashatan, A. & Miri, A., 2023. XMal: A lightweight memory-based explainable obfuscated-malware detector. *Computers and Security,* 133(C).

Al-Hawamleh, A. M., Alorfi, A., Al-Gasawneh, J. & Al-Rawashdeh, G. H., 2020. Cyber Security and Ethical Hacking: The Importance of Protecting User Data. *Solid State Technology,* 63(5), pp. 7894-7899.

Aljanabi, M., Ismail, M. A. & Ali, A. H., 2021. Intrusion Detection Systems, Issues, Challenges, and Needs. *International Journal of Computational Intelligence Systems,* pp. 1-11.

Bellur, M., 2023. *GitHub: Reverse-Shell-Detecter.* [Online]
Available at: https://github.com/bellurm/Reverse-Shell-Detecter
[Accessed 10 Jan 2024].

Best X Software, 2024. *Free Download Best Free Keylogger - 2024.* [Online]
Available at: https://bestxsoftware.com/
[Accessed 6 February 2024].

Bonhomme, C., 2023. *GitHub: PyHIDS.* [Online]
Available at:
https://github.com/cedricbonhomme/pyHIDS/commits/master/pyhids/pyHIDS.py
[Accessed 10 January 2024].

Breitenbacher, D. et al., 2021. HADES-IoT: A Practical and Effective Host-Based Anomaly Detection System for IoT Devices (Extended Version). *IEEE Internet of Things Journal,* 9(12), pp. 9640 - 9658.

Chemiron, A., 2022. *GitHub: Anti-keylogger.* [Online]
Available at: https://github.com/Nada18-sec/Anti-keylogger
[Accessed 10 January 2024].

Department for Science, Innovation & Technology, 2023. *Cyber security breaches survey 2023 GOV.UK.* [Online]
Available at: https://www.gov.uk/government/statistics/cyber-security-breaches-survey-

2023/cyber-security-breaches-survey-2023
[Accessed 1 April 2024].

depascaldc, 2020. *GitHub-depascaldc/DoS-Tool.* [Online]
Available at: https://github.com/depascaldc/DoS-Tool
[Accessed 6 February 2024].

Elder, T., 2022. *GitHub: E-Guard.* [Online]
Available at: https://github.com/aelder202/E-Guard
[Accessed 10 January 2024].

Guarda, T. et al., 2016. *Penetration Testing on Virtual Environments.* Kuala Lumpur, s.n.

Gulgun, G., 2017. *GitHub: NIDS-Intrusion-Detection.* [Online]
Available at: https://github.com/ggulgun/NIDS-Intrusion-Detection
[Accessed 10 January 2024].

Gupta, P. K., Mittal, S., Consul, P. & Jindal, J. K., 2020. A Review of Different Vulnerabilities of Security in a Layered Network. *Advances and Applications in Mathematical Sciences,* 20(2), pp. 227-236.

Heimdal, 2023. *Host Intrusion Detection System(HIDS). What it is and how it works.* [Online]
Available at: https://heimdalsecurity.com/blog/host-intrusion-detection-system-hids/
[Accessed 25 March 2024].

Jamehdar, S., 2023. *Intrusion-Detection-Prevention-System.* [Online]
Available at: https://github.com/hmzakhalid/Intrusion-Detection-Prevention-System
[Accessed 10 January 2024].

Jatt, U., 2024. *GitHub: dos_ddos_detector.* [Online]
Available at: https://github.com/umarjatt420/dos_ddos_detector
[Accessed 10 January 2024].

Kaiser, F. K. et al., 2022. *Cyber threat intelligence enabled automated attack incident response.* Flic-en-Flac, s.n.

Khalid, H., 2023. *GitHub: Intrusion-Detection-Prevention-System.* [Online]
Available at: https://github.com/hmzakhalid/Intrusion-Detection-Prevention-System
[Accessed 21 January 2024].

Khalil, M. I., 2020. *GitHub: intrusion-detection-system-host-base-with-Python-port-and-SSH-server-scanner.* [Online]
Available at: https://github.com/ibrahimkhalilmasud/-intrusion-detection-system-host-base-with-Python-port-and-SSH-server-scanner
[Accessed 10 January 2024].

Kim, J.-Y. & Cho, S.-B., 2022. Obfuscated Malware Detection Using Deep Generative Model based on Global/Local Features. *Computers and Security,* 112(C).

K, P., Nataraj, B. & Duraisamy, P., 2023. *An Investigation on Attacks in Application Layer Protocols and Ransomeware Threats in Internet of Things.* Coimbatore, s.n.

Lachkov, P. & Tawalbeh, L., 2022. Vulnerability Assessment for Applications Security Through Penetration Simulation and Testing. *Journal of Web Engineerin,* 21(7), p. 2187–2208.

LinkedIn, 2023. *https://www.linkedin.com/advice/0/what-pros-cons-signature-based-vs-anomaly-based.* [Online]
Available at: https://www.linkedin.com/advice/0/what-pros-cons-signature-based-vs-anomaly-based
[Accessed 25 March 2024].

Malek, Z. S., Trivedi, B. & Shah, A., 2020. *User behavior Pattern -Signature based Intrusion Detection.* London, s.n.

Moustafa, N. et al., 2023. Explainable Intrusion Detection for Cyber Defences in the Internet of Things: Opportunities and Solutions. *IEEE Communications Surveys & Tutorials,* 25(3), pp. 1775 - 1807.

Nebbione, G. & Calzarossa, M. C., 2020. Security of IoT Application Layer Protocols: Challenges and Findings. *Future Internet.*

Plesak, N., 2021. *GitHub: Threat Hunting with File Entropy.* [Online]
Available at: https://github.com/crslv/entropy
[Accessed 10 January 2024].

Quinn, B. & Arthur, C., 2011. PlayStation Network hackers access data of 77 million users. *The Guardian*, 26 April.

Rathnayake, V., 2023. *GitHub: TrueIDS-Desktop-Application.* [Online]
Available at: https://github.com/vehanmr/TrueIDS-Desktop-Application
[Accessed 10 Janaury 2024].

Ring, J. H. et al., 2021. Methods for Host-based Intrusion Detection with Deep Learning. *Digital Threats: Research and Practice,* 2(4), p. 1–29.

Seitz, J. & Arnold, T., 2021. *Black Hat Python.* 2nd ed. San Francisco: no starch press.

Shaukat, K., Luo, S., Chen, S. & Liu, D., 2020. *Cyber Threat Detection Using Machine Learning Techniques: A Performance Evaluation Perspective.* Norfolk, IEEE.

Sinha, P., Jha, V. K., Rai, A. K. & Bhushan, B., 2017. *Security vulnerabilities, attacks and countermeasures in wireless sensor networks at various layers of OSI reference model: A survey.* Coimbatore, s.n.

Slater, J., 2020. *GitHub: crypto-detector.* [Online]
Available at: https://github.com/Wind-River/crypto-detector
[Accessed 10 January 2024].

S, M. & Vadivel, R., 2023. Various Possible Attacks and Mitigations of the OSI Model Layers Through Pentesting – An Overview. *New Frontiers in Communication and Intelligent Systems.*

Suzan Hajj, R. E. S. J. B. A. J. D. A. M. C. G., 2021. Anomaly-based intrusion detection systems: The requirements, methods, measurements, and datasets. *Emerging Telecommunications Technologies,* 32(4), p. 36.

Sysdig, n.d. *What is HIDS (Host-Based Intrusion Detection System)?-Sysdig.* [Online]
Available at: https://sysdig.com/learn-cloud-native/detection-and-response/what-is-hids/#:~:text=Host%2DBased%20Intrusion%20Detection%20works,(which%20record%20login%20events).
[Accessed 25 March 2024].

The Python Code, 2023. *Ethical Hacking with Python Ebook - The Python Code.* [Online]
Available at: https://thepythoncode.com/ethical-hacking-with-python-ebook
[Accessed 30 August 2023].

Tidy, J., 2020. How hackers extorted $1.14m from University of California, San Francisco. *BBC News*, 29 June.

Torkura, K. A., Sukmana, M. I., Cheng, F. & Meinel, C., 2019. *SlingShot - Automated Threat Detection and Incident Response in Multi Cloud Storage Systems.* Cambridge, s.n.

Tsimenidis, S., Lagkas, T. & Rantos, K., 2022. Deep Learning in IoT Intrusion Detection. *Journal of Network and Systems Management,* 30(1).

Tutorialspoint, n.d. *DoS and DDoS attack.* [Online]
Available at:
https://www.tutorialspoint.com/python_penetration_testing/python_penetration_testing_dos_and_ddos_attack.htm
[Accessed 10 January 2024].

Vidal, J. M. & Monge, M. A. S., 2020. Obfuscation of Malicious Behaviors for Thwarting Masquerade Detection Systems Based on Locality Features. *Sensors,* pp. 1-14.

VirusTotal, 2012. *VirusTotal-Home.* [Online]
Available at:

https://www.google.com/search?q=virustotal&oq=virustotal&gs_lcrp=EgZjaHJvbWUqDggAEEUYJxg7GIAEGIoFMg4IABBFGCcYOxiABBiKBTIGCAEQRRg8MgYIAhAjGCcyBwgDEAAYgAQyBwgEEAAYgAQyBggFEEUYPDIGCAYQRRg8MgYIBxAFGECoAgCwAgA&sourceid=chrome&ie=UTF-8
[Accessed 20 February 2024].

Vishnuram, G., Tripathi, K. & Tyagi, A. K., 2022. *Ethical Hacking: Importance, Controversies and Scope in the Future.* Coimbatore, s.n.

Wang, H., Yang, J. & Lu, Y., 2020. *A Logical Combination Based Application Layer Intrusion Detection Model.* Guangzhou, s.n.

ytisf, 2014. *GitHub - ytisf /theZoo.* [Online]
Available at: https://github.com/ytisf/theZoo
[Accessed 14 March 2024].

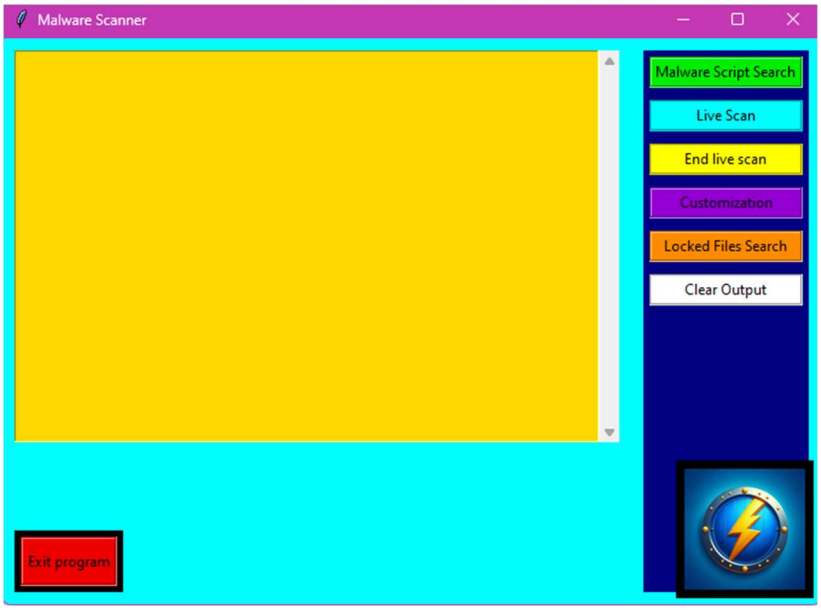
Yungaicela-Naula, N. M., Vargas-Rosales, C. & Perez-Diaz, J. A., 2021. SDN-Based Architecture for Transport and Application Layer DDoS Attack Detection by Using Machine and Deep Learning. *IEEE Access,* Volume 9, pp. 108495 - 108512.

z9fr, 2021. *GitHub-z9fr/DOS-Attack-Script.* [Online]
Available at: https://github.com/z9fr/DOS-Attack-Script
[Accessed 6 February 2024].

Zhang, Z. et al., 2022. Explainable Artificial Intelligence Applications in Cyber Security: State-of-the-Art in Research. *IEEE Access,* Volume 10, pp. 93104 - 93139.

Appendix

Appendix A-Graphical User Interface & Splash Screen



Appendix B-Source Code("Malware Scanner.py")

```
import os,time,logging #os-used for interacting with the OS;time-used for adding delays within code;logging-used for writing log messages
import tkinter as tk #used for making GUIs(graphical user interfaces)
from tkinter import scrolledtext#allow for text widget
from tkinter import filedialog#creating file/directory selection windows.
import tkinter.simpledialog as tk_simpledialog#Gets input from tkinter
import threading
from threading import Thread #Allows you to implement threading(running functions concurrently)
import psutil#Used for retrieving info on processes and system utilization
import re #Supports regular expressions
import shutil #used for file operations(copying,moving,deleting,comparing etc)
from scapy.all import * #used for packet capturing-for DoS and packet snifffer detection
from datetime import datetime, timedelta #Used for handling date and time operations.
from scapy.all import sniff# Used for packet capturing
from scapy.layers.inet import ICMP,TCP,IP
import socket # Used for detecting network communication and connections
from scapy.layers.l2 import Ether
from scapy.layers.http import HTTPRequest
import subprocess# Used for running system commands or subprocesses
```

```
#from scapy.sendrecv import sniff
import pyshark# Provides a Python wrapper for tshark, useful for packet analysis
import ipaddress # Provides classes representing IP addresses and networks
import sys # Provides access to some variables used or maintained by the Python interpreter
import fnmatch # Supports Unix-style filename pattern matching.
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler # Allows monitoring file system
events.
from watchdog.events import FileCreatedEvent, FileDeletedEvent, FileMovedEvent,
FileModifiedEvent
import math # Provides mathematical functions.
import time #handlles time operations
from PIL import Image, ImageTk#opening, manipulating, and saving many different image
file formats.
import mimetypes #Aquire data about a file's MIME type
import docx  # For handling .docx files
import PyPDF2  # For handling PDF files
import openpyxl  # For handling .xlsx files
from pptx import Presentation  # For handling .pptx files
import pytesseract #For using OCR data extraction processes to extract malicious embedded
files from within JPEG and PNG images.
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'


selected_folder = ""
detected_processes = []

scan_folder = ""  # Will be set when the user selects a folder
stop_live_scanner_flag = False  # Flag to indicate whether to stop the scanner
stop_folder_scanner_flag = False

# Configure logging
logging.basicConfig(level=logging.INFO, filename='scan_log.txt', filemode='a',
                    format='%(asctime)s - %(levelname)s - %(message)s')

target_ports = [20, 21, 22, 23, 25, 53, 80, 110, 143, 443, 445, 587, 3389]
# 20/21-FTP,22-SSH,23-Telnet,25-SMTP,53-DNS,80-HTTP,110-POP3,143-IMAP,443-
HTTPS,445-SMB,587-encrypted SMTP,3389-RDP

#keywords for malware based on programming language; keywords for packet sniffer,
keylogger, denial of servoice, ransomware and reverse shell scripts
exemalware_keywords = ["GetAsyncKeyState", "SetWindowsHookEx", "flood", "udp",
"tcp", "encrypt", "decrypt", "ransom", "pcap", "sniff", "capture"]
pymalware_keywords = ['socket.socket', 'subprocess.Popen', 'os.popen', 'pty.spawn',
'pty.fork', 'socket.create_connection', 'socket.settimeout', 'urllib.request.urlopen',
'requests.get', 'requests.post', 'threading.Thread', 'pynput.keyboard.Listener', 'keylogger',
'keyboard', 'keyboard.on_press', 'pyHook.HookManager', 'ctypes.windll.user32.GetKeyState',
'scapy.sniff', 'socket.AF_PACKET', 'socket.SOCK_RAW', 'pcapy.open_live',
'dpkt.ethernet.Ethernet', 'scapy', 'pcapy','sniff','sniffer', 'tcpdump','packet capture','os.walk',
'os.remove', 'os.system', 'os.rename', 'shutil.rmtree','pycryptodome.Cipher',
```

```
'pycryptodome.PublicKey', 'pycryptodome.Random', 'encrypt', 'ransom', 'socket']
csharpmalware_keywords = ["Keylogger|GetAsyncKeyState", "nativekeypressed",
'TcpClient', 'TcpListener', 'Process.Start', 'System.Net.Sockets', 'System.Diagnostics.Process',
"Thread.Sleep", "Task.Delay", 'File.WriteAllText', 'FileStream.Write', 'NetworkInterface',
'PacketCapture', 'System.Net.Sockets.TcpClient', 'System.Net.Sockets.NetworkStream',
'System.Net.WebRequest', 'System.Net.Http.HttpClient', 'InputSimulator',
'KeyboardHookListener', 'System.Windows.Forms.Keys', 'GetAsyncKeyState', 'SharpPcap',
'PacketDevice.Open', 'PacketCommunicator', 'SharpPcap.RawCapture',
'PacketDotNet.EthernetPacket', 'Directory.GetFiles', 'File.Delete', 'File.Move',
'File.ReadAllBytes', 'File.WriteAllBytes', 'File.Encrypt', 'File.Decrypt']
cmalware_keywords = ['socket', 'winsock', 'CreateProcess', 'system', 'ShellExecute',
'getaddrinfo', 'connect', 'send', 'recv', 'Sleep', 'CreateThread', 'GetAsyncKeyState',
'SetWindowsHookEx', 'GetKeyState', 'getch', 'getchar', 'RegisterHotKey', 'usleep', 'fwrite',
'fread', 'keybd_event', 'WH_KEYBOARD_LL', 'socket', 'winsock', 'WSASocket', 'WSARecv',
'libpcap', 'pcap_open_live', 'pcap_compile', 'pcap_setfilter', 'pcap_open_live' 'FindFirstFile',
'FindNextFile', 'DeleteFile', 'MoveFile', 'CreateFile', 'WriteFile', 'CryptEncrypt',
'CryptDecrypt', 'encrypt', 'decrypt']
javamalware_keywords = ['java.net.Socket', 'java.net.ServerSocket', 'java.io.BufferedReader',
'java.io.InputStreamReader', 'java.lang.Runtime', 'java.net.Socket',
'java.net.HttpURLConnection', 'java.net.URL', 'java.net.InetAddress',
'java.util.concurrent.ThreadPoolExecutor', 'java.awt.event.KeyEvent', 'java.awt.Robot',
'java.util.logging.Logger', 'javax.swing.KeyStroke', 'JpcapCaptor', 'JpcapSender',
'JpcapPacket', 'PcapPacketHandler', 'JpcapWriter', 'java.nio.file.Files', 'java.nio.file.Path',
'java.nio.file.Paths', 'java.nio.file.StandardCopyOption', 'javax.crypto.Cipher']

allmalwarekeywords=
exemalware_keywords+pymalware_keywords+csharpmalware_keywords+cmalware_keywor
ds+javamalware_keywords
def Scan_File(file_path,allmalwarekeywords):
    try:
        with open(file_path, 'rb') as file:
            content = file.read()
            file_extension = os.path.splitext(file_path)[1].lower()

            # Binary file extensions
            binary_extensions = ['.exe']
            if file_extension in binary_extensions:
                if any(keyword.encode('utf-8') in content for keyword in exemalware_keywords):
                    alert_message = f"Alert! Malicious keywords found in file: {file_path}"
                    return True

                exe_shellcode_pattern = re.compile(rb'\\x[0-9a-fA-F]{2}')  # checks for shellcode
in an exe file.
                if exe_shellcode_pattern.search(content):
                    alert_message = f"Alert! Malicious shellcode pattern found in file: {file_path}"
                    return True
            else:
                try:
                    content_str = content.decode('utf-8', errors='ignore')
```

```
            # Additional checks for text file extensions
            file_extension = os.path.splitext(file_path)[
                1].lower()
            document_extensions = ['.doc', '.pdf', '.docx', '.xlsx', '.pptx']
            if file_extension in document_extensions:
                # Handle each document type separately
                if file_extension == '.doc':  # For .doc files
                    doc_content = docx.Document(file_path).read_text()
                    keyword_count = sum(
                        keyword.lower() in doc_content.lower() for keyword in
allmalwarekeywords)
                    if keyword_count >= 2:
                        return True

                elif file_extension == '.txt':
                    # Check for keywords in TXT files
                    keyword_count = sum(
                        keyword.lower() in content_str.lower() for keyword in
allmalwarekeywords)
                    if keyword_count >= 2:
                        return True

                elif file_extension == '.pdf':  # For .pdf files
                    pdf_content = ""
                    pdf_reader = PyPDF2.PdfReader(file_path)
                    for page in pdf_reader.pages:
                        pdf_content += page.extract_text()
                    keyword_count = sum(
                        keyword.lower() in pdf_content.lower() for keyword in
allmalwarekeywords)
                    if keyword_count >= 2:
                        return True

                elif file_extension == '.docx':  # For .docx files
                    docx_content = ""
                    doc = docx.Document(file_path)
                    for paragraph in doc.paragraphs:
                        docx_content += paragraph.text
                    keyword_count = sum(
                        keyword.lower() in docx_content.lower() for keyword in
allmalwarekeywords)
                    if keyword_count >= 2:
                        return True

                elif file_extension == '.xlsx':  # For .xlsx files
                    wb = openpyxl.load_workbook(file_path)
                    keyword_count = 0
                    for sheet in wb.worksheets:
                        for row in sheet.iter_rows():
                            for cell in row:
```

```
                        if cell.value and any(keyword.lower() in str(cell.value).lower() for
keyword in
                                              allmalwarekeywords):
                            keyword_count += 1
                            if keyword_count >= 2:
                                return True

                elif file_extension == '.pptx':  # For .pptx files
                    prs = Presentation(file_path)
                    pptx_content = ""
                    for slide in prs.slides:
                        for shape in slide.shapes:
                            if hasattr(shape, "text"):
                                pptx_content += shape.text
                    keyword_count = sum(
                        keyword.lower() in pptx_content.lower() for keyword in
allmalwarekeywords)
                    if keyword_count >= 2:
                        return True

            image_extensions=['.jpeg', '.jpg', '.png']
            if file_extension in image_extensions:
                # Perform OCR to extract text from the image
                image = Image.open(file_path)
                extracted_text = pytesseract.image_to_string(image)
                # Check the extracted text for malicious keywords
                if any(keyword.lower() in extracted_text.lower() for keyword in
allmalwarekeywords):
                    alert_message = f"Alert! Malicious keywords found in image: {file_path}"
                    return True

            elif file_extension == '.py':#Python
                keyword_count = 0
                for keyword in pymalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:#check for at least 2 keywords
                        return True
            elif file_extension == '.cs':#C Sharp
                keyword_count = 0
                for keyword in csharpmalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
                    if keyword_count >= 2:
                        return True
            elif file_extension == '.c':#C
                keyword_count = 0
                for keyword in cmalware_keywords:
                    if keyword.lower() in content_str.lower():
                        keyword_count += 1
```

```
                    if keyword_count >= 2:
                        return True
                elif file_extension == '.java':#Java
                    keyword_count = 0
                    for keyword in javamalware_keywords:
                        if keyword.lower() in content_str.lower():
                            keyword_count += 1
                        if keyword_count >= 2:
                            return True
                exe_shellcode_pattern = re.compile(b"\\\\x[0-9a-fA-F]{2}")  # checks for
shellcode in the above file types
                if exe_shellcode_pattern.search(content):
                    return True
                else:
                    pass  # skips any file that isn't python, C#, visual basic, c or java

            except (UnicodeDecodeError, PermissionError, FileNotFoundError) as e:
                logging.error(f"Error reading {file_path}: {e}", exc_info=True)

    except (PermissionError, FileNotFoundError) as e:
        logging.error(f"Error reading {file_path}: {e}", exc_info=True)
    return False

def Capture_Malicious_File(file_path, destination_folder):#Copies any malicious file and
sends them to the destination folder(in this instance "Captured_Scripts")
    try:
        shutil.copy(file_path, destination_folder)
        logging.info(f"Malicious file captured and contained: {file_path}")
    except Exception as e:
        logging.error(f"Error capturing malicious file: {e}", exc_info=True)

def Scan_Directory(directory,allmalwarekeywords,output_text):
    malware_found = False

    for root, dirs, files in os.walk(directory):
        dirs = dirs or []

        for file in files:
            file_path = os.path.join(root, file)
            file_extension = os.path.splitext(file_path)[1].lower()

            if file_extension not in ['.py', '.cs', '.c', '.java','.doc', '.pdf', '.docx', '.xlsx', '.pptx', '.jpeg',
'.jpg', '.png']:
                continue  # Skip scanning this file

            if Scan_File(file_path,allmalwarekeywords):
                malware_found = True
                alert_message = f"Alert! Signature matched in file: {file_path}"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)
```

```
                Capture_Malicious_File(file_path, "Captured_Scripts")

            for subdir in dirs:  # Checks subfolders within chosen folder.
                subdir_path = os.path.join(root, subdir)
                Scan_Directory(subdir_path,allmalwarekeywords, output_text)

        if not malware_found:
            output_text.insert(tk.END, "All (other) files here are safe.\n")
            logging.info("No malware found")

    def Start_Scanner(allmalwarekeywords, output_text):  # starts scan of the file
        global stop_folder_scanner_flag
        global scan_folder
        scan_folder = filedialog.askdirectory()
        output_text.insert(tk.END, f"Scan folder selected: {scan_folder}\n")
        while not stop_folder_scanner_flag:
            Scan_Directory(scan_folder, allmalwarekeywords, output_text)
            time.sleep(60)  # Sleep for 60 seconds before the next scan

    def DoS_Detection(output_text):
        try:
            # Define a list of commonly targeted ports for DoS attacks
            dos_ports = target_ports  # Example ports commonly targeted for HTTP and HTTPS
    DoS attacks

            # Get the IP address of the local machine
            local_ip = socket.gethostbyname(socket.gethostname())

            # Initialize a dictionary to store counts of packets per source IP address
            packet_counts = {}

            # Packet callback function
            def Packet_Callback(packet):
                if packet.haslayer(IP) and packet[IP].dst == local_ip:
                    src_ip = packet[IP].src
                    dst_port = packet[IP].dport

                    # Check if the packet's destination port matches any of the DoS target ports
                    if dst_port in dos_ports:
                        if src_ip in packet_counts:
                            packet_counts[src_ip] += 1
                        else:
                            packet_counts[src_ip] = 1

                        # Check if the number of packets from a single source IP exceeds a threshold
                        dos_threshold = 100  # Adjust as needed based on expected traffic
                        if packet_counts[src_ip] >= dos_threshold:
                            dos_message = f"Potential DoS attack detected from {src_ip} on port
    {dst_port}"
                            output_text.insert(tk.END, str(dos_message) + "\n")
```

```
                logging.warning(dos_message)

        sniff(prn=Packet_Callback, store=False, timeout=60)

    except Exception as e:
        logging.error(f"Error in DoS detection: {e}", exc_info=True)

def Is_File_Encrypted(file_path, output_text):
    encryption_signatures = [
        b"gAAAA",  # Fernet
        b"Salted__",  # OpenSSL
        b"BEGIN RSA PRIVATE KEY",  # RSA private key
        b"BEGIN DSA PRIVATE KEY",  # DSA private key
        b"BEGIN EC PRIVATE KEY",  # EC private key
        b"-----BEGIN PGP PRIVATE KEY BLOCK-----",  # PGP private key block
        b"-----BEGIN ENCRYPTED PRIVATE KEY-----",  # Encrypted private key
        b"-----BEGIN CERTIFICATE-----",  # Certificate
        b"-----BEGIN ENCRYPTED MESSAGE-----",  # Encrypted message
        b"-----BEGIN PKCS7-----",  # PKCS7 (Cryptographic Message Syntax Standard)
        b"-----BEGIN PGP MESSAGE-----",  # PGP message
        b"-----BEGIN GPG MESSAGE-----",  # GPG message
    ]
    try:
        # Check file extension
        _, file_extension = os.path.splitext(file_path)
        file_extension = file_extension.lower()

        # Check if the file extension matches common encrypted file extensions
        encrypted_extensions = ['.wcry', '.wnry','.wncry','.locky', '.cerber', '.zepto', '.crysis',
'.lockyv2']
        if file_extension in encrypted_extensions:
            output_text.insert(tk.END, f"File {file_path} has a known encrypted file
extension.\n")
            return True

        # Get the MIME type of the file
        mime_type, _ = mimetypes.guess_type(file_path)
        if mime_type is not None:
            # Extract the extension from the MIME type
            _, file_extension = os.path.splitext(file_path)
            file_extension = file_extension.lower()
            mime_extension = mimetypes.guess_extension(mime_type)

            # Compare the file extension to the MIME type's extension
            if file_extension != mime_extension:
                output_text.insert(tk.END,f"Warning: File extension ({file_extension}) does not
match MIME type ({mime_extension}).\n")

        # Open the file in binary mode for reading
        with open(file_path, 'rb') as file:
```

```
            # Read a portion of the file for signature analysis
            data = file.read(1024)  # Read the first 1024 bytes of the file
            # Check for known encryption signatures
            for signature in encryption_signatures:
                if data.startswith(signature):
                    output_text.insert(tk.END, f"File {file_path} appears to be encrypted.\n")
                    return True
    except (FileNotFoundError, PermissionError, UnicodeDecodeError) as e:
        output_text.insert(tk.END, f"Error checking file {file_path}: {e}\n")
        return True
    except Exception as e:
        # Handle other exceptions gracefully
        output_text.insert(tk.END, f"Error checking file {file_path}: {e}\n")
        return True
    return False

def Scan_For_Locked_Files(output_text):
    global selected_folder
    selected_folder = filedialog.askdirectory()
    output_text.insert(tk.END, f"Scanning for locked files...\n")
    locked_files_found = False

    for root, dirs, files in os.walk(selected_folder):
        for file in files:
            file_path = os.path.join(root, file)

            if Is_File_Encrypted(file_path, output_text):
                alert_message = f"Alert! Locked file found"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)
                locked_files_found = True

    if not locked_files_found:
        output_text.insert(tk.END, f"No locked files found.\n")

    output_text.insert(tk.END, f"Scan for locked files complete.\n")

def Keylogger_Detector(output_text, interval_seconds=185, threshold=3):
    try:
        cap_filter = 'port 20 or port 21 or port 587'
        message_times = {}
        log_files_created = {}

        def Keylogger_Packet_Callback(packet):
            if TCP in packet and packet[TCP].dport in [20, 21, 587]:
                dst_ip = packet[IP].dst
                current_time = datetime.now()
                message_times.setdefault(dst_ip, []).append(current_time)
                repeated_messages = [time for time_list in message_times.values() for time in
time_list if
```

```
                        (current_time - time).total_seconds() <= interval_seconds]
            if len(repeated_messages) >= threshold:
                alert_message = f"Repeated messages detected to port {packet[TCP].dport} at IP
{dst_ip} within {interval_seconds} seconds! This indicates a potential keylogger active on
your device"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)

        # Check for newly created log files
        for root, dirs, files in os.walk('/var/log'):  # Adjust the directory path as needed
            for file in files:
                filepath = os.path.join(root, file)
                if filepath not in log_files_created:
                    creation_time = os.path.getctime(filepath)
                    if (current_time - creation_time) <= interval_seconds:
                        log_files_created[filepath] = current_time
                        alert_message = f"New log file created: {filepath}"
                        output_text.insert(tk.END, str(alert_message) + "\n")
                        logging.warning(alert_message)

        sniff(filter=cap_filter, prn=Keylogger_Packet_Callback, store=False, timeout=60)

    except Exception as e:
        logging.error(f"Error in tracking port activity: {e}", exc_info=True)

def Packet_Sniffer_Detection(output_text):
    try:

        # List of popular packet sniffing processes/applications
        sniffing_processes = ["Wireshark.exe", "tcpdump.exe", "WinPcap.exe"]

        # Check for popular packet sniffing processes
        for proc in psutil.process_iter(["name"]):
            if proc.info["name"] in sniffing_processes:
                alert_message = f"Potential packet sniffing process detected: {proc.info['name']}"
                output_text.insert(tk.END, str(alert_message) + "\n")
                logging.warning(alert_message)

        # Check for raw socket creation on both Linux and Windows
        raw_sockets = []
        for proc in psutil.process_iter(['pid', 'name']):
            try:
                connections = proc.connections(kind='all')
            except (psutil.NoSuchProcess):
                continue
            for conn in connections:
                if conn.family in (psutil.AF_INET, psutil.AF_INET6) and conn.type ==
socket.SOCK_RAW:
                    raw_sockets.append((proc.pid, conn.fd))
                    break
```

```
        if raw_sockets:
            alert_message = "Raw socket(s) created by program(s):"
            for pid, fd in raw_sockets:
                alert_message += f"\nPID: {pid}, FD: {fd}"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

        # Check for promiscuous mode
        network_interfaces = psutil.net_if_addrs()
        for interface, addresses in network_interfaces.items():
            for address in addresses:
                if address.family == psutil.AF_LINK and address.address == "00:00:00:00:00:00":
                    alert_message = f"Promiscuous mode detected on interface: {interface}"
                    output_text.insert(tk.END, str(alert_message) + "\n")
                    logging.warning(alert_message)

        # Check for suspicious network privileges
        for proc in psutil.process_iter():
            try:
                if proc.connections(kind='inet') and proc.username() != 'SYSTEM':
                    alert_message = f"Suspicious process {proc.name()} with raw sockets or
elevated network privileges!"
                    output_text.insert(tk.END, str(alert_message) + "\n")
                    logging.warning(alert_message)
            except psutil.AccessDenied:
                continue

    except Exception as e:
        logging.error(f"Error in packet sniffing activity: {e}", exc_info=True)

def Is_IPAdress_Private(ip):
    try:
        return ipaddress.ip_address(ip).is_private
    except ValueError:
        return False

def get_my_ip():
    return socket.gethostbyname(socket.gethostname())

def Check_For_Reverse_Shell(output_text):
    my_ip=get_my_ip()
    try:
        while not stop_live_scanner_flag:
            # Get all network connections
            connections = psutil.net_connections(kind='inet')

            # Check for outbound connections
            for conn in connections:
                if conn.status == psutil.CONN_ESTABLISHED:
```

```
                local_ip, local_port = conn.laddr
                remote_ip, remote_port = conn.raddr

                # Check if the remote IP address is private-will ca
                if Is_IPAdress_Private(remote_ip) and remote_ip != my_ip:
                # Check if the connection is outbound and not to a common service port-
127.0.0.1 is localhost. port 80 is HTTP and 443 is HTTPS
                    if local_ip != '127.0.0.1' and remote_ip != '127.0.0.1' and remote_port not in
[80, 443]:
                        alert_message = f"Potential reverse shell detected: {local_ip}:{local_port} -
> {remote_ip}:{remote_port}"
                        output_text.insert(tk.END, str(alert_message) + "\n")
                        logging.warning(alert_message)

            time.sleep(3)
        except Exception as e:
            logging.error(f"Error in reverse shell detection: {e}", exc_info=True)

class FileTransferHandler(FileSystemEventHandler):
    def __init__(self, output_text):
        super().__init__()
        self.output_text = output_text

    def on_created(self, event):
        if isinstance(event, FileCreatedEvent):
            alert_message = f"Alert! File created: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_deleted(self, event):
        if isinstance(event, FileDeletedEvent):
            alert_message = f"Alert! File deleted: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_modified(self, event):
        if isinstance(event, FileModifiedEvent):
            alert_message = f"Alert! File modified: {event.src_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

    def on_moved(self, event):
        if isinstance(event, FileMovedEvent):
            alert_message = f"Alert! File moved: {event.src_path} -> {event.dest_path}"
            self.output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)

def start_file_monitoring(base_directory, output_text):
    current_user = os.getlogin()
    base_directory = os.path.join("C:\\Users", current_user, "OneDrive", "Documents")
```

```
    event_handler = FileTransferHandler(output_text)
    observer = Observer()
    observer.schedule(event_handler, base_directory, recursive=True)
    observer.start()
    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()

def Read_Standard_Actions():#reads standard_actions.txt to get a list of acceptable actions by the pc(applications regularly accessed by PC.
    standard_actions = []
    file_path = "standard_actions.txt"

    if os.path.exists(file_path):
        with open(file_path, 'r') as file:
            standard_actions = [line.strip() for line in file.readlines()]

    return standard_actions
standard_actions = Read_Standard_Actions()

def Anomaly_Action_Checking(output_text):#Checks for actions occuring that aren't normal to PC based on standard_actions.txt
    try:
        running_processes = [proc.name().lower() for proc in psutil.process_iter(attrs=['name'])]

        # Load standard actions from the file
        with open('standard_actions.txt', 'r') as file:
            standard_actions = set(file.read().splitlines())

        # Check if any process is not in the list of standard actions
        anomalies = [process for process in running_processes if process not in standard_actions]

        if anomalies:
            alert_message = f"Alert! Anomaly detected. Unexpected processes running:{', '.join(anomalies)}"#edited out bit which lists anomalies
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

    except Exception as e:
        logging.error(f"Error in anomaly-based process running detection: {e}", exc_info=True)
    return False

def Read_Thresholds_From_File():
    # Get the current directory
    current_directory = os.getcwd()
```

```
    # Construct the absolute path to the file
    file_path = os.path.join(current_directory, "resource_usage.txt")

    try:
        with open(file_path, "r") as file:
            lines = file.readlines()
            # Extract CPU threshold from the first line and RAM threshold from the second line
            cpu_threshold = float(lines[0].split(':')[1].strip())
            ram_threshold = float(lines[1].split(':')[1].strip())
            return cpu_threshold, ram_threshold
    except FileNotFoundError:
        # Handle the case when the configuration file is not found
        print("Configuration file not found.")
        return None, None
    except Exception as e:
        # Handle other exceptions (e.g., invalid format)
        print(f"Error reading configuration file: {e}")
        return None, None
# Function to check anomaly-based resource usage
def Anomaly_Resource_Usage(output_text):
    try:
        # Read CPU and RAM thresholds from the configuration file
        cpu_threshold, ram_threshold = Read_Thresholds_From_File()

        if cpu_threshold is None or ram_threshold is None:
            return False  # Return False if thresholds couldn't be read

        # Get CPU usage percentage
        cpu_usage = psutil.cpu_percent(interval=1)
        # Get memory usage percentage
        memory_usage = psutil.virtual_memory().percent

        # Check if CPU usage exceeds the threshold
        if cpu_usage > cpu_threshold:
            alert_message = f"Alert! Abnormal CPU usage detected: {cpu_usage}%"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

        # Check if memory usage exceeds the threshold
        elif memory_usage > ram_threshold:
            alert_message = f"Alert! Abnormal memory usage detected:  {memory_usage}%"
            output_text.insert(tk.END, str(alert_message) + "\n")
            logging.warning(alert_message)
            return True

    except Exception as e:
        logging.error(f"Error in anomaly-based resource usage detection: {e}", exc_info=True)

    return False
```

```
def Customization(output_text):
    try:
        # Add standard actions
        password = tk_simpledialog.askstring("Password", "Enter password:", show='*')
        if password == "admin":
            new_processes = tk_simpledialog.askstring("Add Standard Actions", "Enter process
names separated by commas:")
            if new_processes:
                # Split the input by commas and add each process individually
                new_processes_list = [process.strip() for process in new_processes.split(',')]
                for new_process_name in new_processes_list:
                    standard_actions.append(new_process_name.lower())
                    output_text.insert(tk.END, f"Added '{new_process_name}' to standard
actions.\n")
                    with open("standard_actions.txt", 'a') as file:
                        file.write(f"{new_process_name}\n")

        # Change CPU/RAM thresholds
        cpu_threshold = tk_simpledialog.askfloat("Change CPU Threshold", "Enter CPU
threshold (0-100):", minvalue=0, maxvalue=100)
        ram_threshold = tk_simpledialog.askfloat("Change RAM Threshold", "Enter RAM
threshold (0-100):", minvalue=0, maxvalue=100)

        # Write the new thresholds to the configuration file
        with open('resource_usage.txt', 'w') as file:
            file.write(f"CPU Threshold: {cpu_threshold}\n")
            file.write(f"RAM Threshold: {ram_threshold}\n")

        output_text.insert(tk.END, f"Customization successful. CPU Threshold changed to
{cpu_threshold}% and RAM Threshold changed to {ram_threshold}%.\n")

    except Exception as e:
        output_text.insert(tk.END, f"Error during customization: {e}\n")

def Live_Scan(output_text, base_directory, target_ports):
    output_text.insert(tk.END, "Live scan started\n")
    global stop_live_scanner_flag
    stop_live_scanner_flag = False

    # Start file monitoring in a separate thread
    file_monitor_thread = threading.Thread(target=start_file_monitoring,
args=(base_directory, output_text), daemon=True)
    file_monitor_thread.start()

    while not stop_live_scanner_flag:
        if Anomaly_Action_Checking(output_text):
            alert_message = "Alert!-Anomaly unlisted process detected!"
            output_text.insert(tk.END, alert_message + "\n")
            logging.warning(alert_message)
```

```
        if Anomaly_Resource_Usage(output_text):
            alert_message = "Alert!-Anomaly CPU/RAM usage detected!"
            output_text.insert(tk.END, alert_message + "\n")
            logging.warning(alert_message)

        Packet_Sniffer_Detection(output_text)

        Keylogger_Detector(output_text)

        reverse_shell_thread = threading.Thread(target=Check_For_Reverse_Shell,
args=(output_text,), daemon=True)
        reverse_shell_thread.start()

        sniff(prn=lambda packet: DoS_Detection(output_text), store=False, timeout=60)

def Stop_Live_Scan(output_text):
    global stop_live_scanner_flag
    stop_live_scanner_flag = True
    output_text.insert(tk.END, "Live scan stopped.\n")

def clear_interface(output_text):
    output_text.delete('1.0', tk.END)

def Exit_Program():  # Ends the program
    quit()

def splash_screen():
    splash_root = tk.Tk()
    splash_root.title("Malware scanner")
    splash_root.geometry("450x200")
    # splash_root.configure(bg='blue')

    # Load background image
    background_image = Image.open("ss_background.jpg")
    background_photo = ImageTk.PhotoImage(background_image)

    # Create a label with the background image
    background_label = tk.Label(splash_root, image=background_photo)
    background_label.image = background_photo
    background_label.place(x=0, y=0, relwidth=1, relheight=1)

    label = tk.Label(splash_root, text="Malware scanner by J.Russell", font=("Unispace", 20),
bg='green')
    label.pack(expand=True)

    splash_root.after(8000, splash_root.destroy)
    splash_root.mainloop()

def start_splash_and_gui():
```

```
    splash_thread = Thread(target=splash_screen)
    splash_thread.start()
    splash_thread.join()
    Create_GUI()


def Create_GUI():
    root = tk.Tk()
    root.title("Malware Scanner")
    root.configure(bg='cyan')

    # Set up grid for layout
    root.columnconfigure(0, weight=1)
    root.columnconfigure(1, weight=3)
    root.rowconfigure(0, weight=1)
    root.rowconfigure(1, weight=3)  # Added row configuration

    # Create frame for output text on the left
    output_frame = tk.Frame(root)
    output_frame.grid(row=0, column=0, padx=10, pady=10, sticky='nsew')

    output_text = scrolledtext.ScrolledText(output_frame, width=60, height=20, bg='gold')  #
Colour of output text box
    output_text.pack(expand=True, fill='both')

    # Create frame for buttons on the right
    buttons_frame = tk.Frame(root, bg='navy')
    buttons_frame.grid(row=0, column=1, rowspan=2, padx=10, pady=10, sticky='nsew')  #
rowspan changed to 2

    Folder_scan_button = tk.Button(buttons_frame, text="Malware Script Search",
bg='green2', relief=tk.GROOVE,
                           command=lambda: Thread(target=Start_Scanner,
args=(allmalwarekeywords, output_text),
                                                  daemon=True).start())
    Folder_scan_button.pack(fill='x', padx=5, pady=5)

    Live_scan_button = tk.Button(buttons_frame, text="Live Scan", bg='cyan',
relief=tk.GROOVE,
                         command=lambda: Thread(target=Live_Scan,
                                                args=(output_text, selected_folder, target_ports),
                                                daemon=True).start())
    Live_scan_button.pack(fill='x', padx=5, pady=5)

    Stop_live_scan_button = tk.Button(buttons_frame, text="End live scan", bg='yellow',
relief=tk.GROOVE,
                              command=lambda: Stop_Live_Scan(output_text=output_text))
    Stop_live_scan_button.pack(fill='x', padx=5, pady=5)

    Customization_button = tk.Button(buttons_frame, text="Customization", bg='darkviolet',
```

```
relief=tk.GROOVE,
                                command=lambda: Customization(output_text))
    Customization_button.pack(fill='x', padx=5, pady=5)

    Lock_scan_button = tk.Button(buttons_frame, text="Locked Files Search",
bg='darkorange', relief=tk.GROOVE,
                             command=lambda: Thread(target=Scan_For_Locked_Files,
args=(output_text,),
                                                    daemon=True).start())
    Lock_scan_button.pack(fill='x', padx=5, pady=5)

    # Add the button to clear the output text
    clear_interface_text_button = tk.Button(buttons_frame, text="Clear Interface", bg='white',
relief=tk.GROOVE,
                                    command=lambda: clear_inteface(output_text))
    clear_interface_text_button.pack(fill='x', padx=5, pady=5)

    # Create frame for the exit button in the bottom left corner
    exit_frame = tk.Frame(root, bg='black')
    exit_frame.grid(row=1, column=0, padx=10, pady=10, sticky='sw')  # Placed at bottom left
    Quit_button = tk.Button(exit_frame, text="Exit program", bg='red2', relief=tk.GROOVE,
command=Exit_Program,
                        width=10, height=2)
    Quit_button.pack(padx=5, pady=5)

    # Create frame for the image near the buttons
    image_frame = tk.Frame(root, bg='black')  # Outer frame of logo
    image_frame.grid(row=1, column=1, padx=6, pady=5, sticky='se')  # Adjusted grid
placement

    image = Image.open("Logo.jpg")
    image = image.resize((100, 100), Image.LANCZOS)
    photo = ImageTk.PhotoImage(image)

    label = tk.Label(image_frame, image=photo, bg='black')  # Inner frame
    label.image = photo
    label.pack(side="right", padx=5, pady=5)

    root.mainloop()

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