International Journal of Computational and Experimental Science and ENgineering (IJCESEN)

Vol. 10-No.3 (2024) pp. 480-490
http://www.ijcesen.com

ISSN: 2149-9144

Research Article

Dynamic Malware Analysis Using a Sandbox Environment, Network Traffic Logs, and Artificial Intelligence.

Mesut GÜVEN*

TOBB University of Economics and Technology, TR-06560 Ankara,
* **Corresponding Author Email:** mesuttguven@gmail.com - **ORCID:** 0000-0002-0957-8541

Article Info:

DOI: 10.22399/ijcesen.460
Received : 19 September 2024
Accepted : 26 September 2024

Keywords:

Artificial Intelligence
Machine Learning
Cyber Security
Malware Analysis
Sandbox

Abstract:

Dynamic malware analysis plays a pivotal role in modern cybersecurity, offering insights into malware behavior through dynamic execution and network traffic analysis. In this study, we present a comprehensive approach to dynamic malware analysis using a sandbox environment and network traffic logs. Our methodology involves the extraction of relevant features from network traffic captured in pcap files. We conducted experiments using a virtualized Oracle VirtualBox environment, where benign and malicious software samples were executed within a Windows virtual machine controlled by Python scripts. For network emulation, we utilized tools from the REMnux distribution, including InetSim and FakeDNS, to simulate realistic network interactions during malware execution. The collected pcap data underwent preprocessing and feature extraction to capture essential behavioral patterns and network indicators. Machine learning and artificial intelligence models were developed to classify malware based on these extracted features. Our findings underscore the efficacy of dynamic analysis coupled with machine learning in detecting and classifying malware variants based on their network behavior. This research contributes to advancing techniques for real-time threat detection and response in cybersecurity, emphasizing the importance of dynamic malware analysis in mitigating evolving cyber threats.

1. Introduction

Malware, short for "malicious software," encompasses a diverse range of programs designed to infiltrate computer systems with harmful intent. According to the National Institute of Standards and Technology (NIST), malware is defined as "a program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victim" [1]. This definition highlights malware's primary objectives of compromising security and disrupting normal operations.

TechTarget further elaborates, describing malware as "any program or file that is harmful to a computer user," capable of stealing, encrypting, or deleting sensitive data, altering computing functions, and monitoring user activities without consent [2]. BullGuard underscores the intrusive nature of malware, emphasizing its design to infiltrate and damage computers without user consent [3]. Kaspersky notes that malware exists in various forms, including viruses, worms, Trojans, and spyware, each posing unique threats to system integrity [4].

Fundamentally, malware is stealthy software engineered to gain unauthorized access or cause harm to computers and devices. Norton succinctly defines it as "software that is specifically designed to gain access or damage a computer without the knowledge of the owner" [5]. These definitions collectively emphasize two core traits of malware: malicious intent and its ability to execute actions surreptitiously, often without the user's awareness.

Malware classification is made on several aspects such as by type, by malicious behaviour, by privilege, etc. Generally, there are two different types of approach in detecting whether a software is malicious or not. These are respectively, the static analysis and dynamic analysis. In static analysis, the softwaare under examination is not executed and

investigated via it's code structure. On the other hand, in dynamic investigation method, the program is executed inside an isolated environment called as sandbox for logging it's network behaviours, API calls, and other system logs to detect the programs malicious actions.

Form this point of view, the dynamic malware analysis plays a crucial role in modern cybersecurity by providing insights into the behavior of malicious software through controlled execution environments and comprehensive log analysis. This study focuses on automating the analysis process, incorporating virtualized environments to ensure the integrity of the analysis environment and leveraging various log types for thorough behavioral analysis.

The dataset used in this research comprises a diverse collection of malware samples sourced from reputable repositories such as MalwareBazaar [6]. These samples encompass a broad range of malicious behaviors, ensuring robust testing of detection methods. Benign software samples were carefully selected from legitimate sources, including licensed applications and validated Windows Dynamic Link Libraries [7].

In the automated analysis phase, each sample undergoes execution in a virtualized environment using Oracle VirtualBox, ensuring a clean state for every analysis session. This approach mitigates contamination risks from previous analyses, preserving the integrity of results. System and kernel logs are captured alongside network traffic logs (in pcap format) to capture comprehensive behavioral indicators. Our approach also includes automatic collection and analysis of system and kernel logs using tools like Win32 APIs for event log retrieval and processing. This comprehensive log analysis provides deeper insights into malware activities beyond network interactions, contributing to a holistic understanding of malware behaviors [8].

The feature extraction phase plays a crucial role in developing effective malware detection systems. It begins with processing packet capture (pcap) files, which are comprehensive logs of network traffic. This process involves extracting a diverse set of features from the raw network data. In our approach, we have utilized a total of 39 features, which include various metrics such as packet sizes, inter-arrival times, protocol types, and traffic volume. These features capture detailed network traffic patterns and behaviors that may indicate malicious activities. For instance, metrics like mean packet size, standard deviation of packet size, and counts of TCP and UDP packets provide insights into the nature of the traffic, while ratios and distributions of these metrics help in identifying anomalies.

Once these features are extracted, they serve as the input for training machine learning and artificial intelligence models. These models are trained to recognize patterns and behaviors that distinguish normal traffic from potentially malicious activities. The training process involves learning from the extracted features to develop a robust classification mechanism capable of detecting both known and novel threats.

Integrating machine learning and AI into this process significantly enhances the accuracy and effectiveness of malware detection. Traditional detection methods often rely on static signatures or heuristic rules, which can be limited in scope and adaptability. In contrast, machine learning models leverage the extensive feature set to continuously learn and adapt, improving their ability to detect sophisticated and previously unknown malware variants. This dynamic and data-driven approach not only boosts detection accuracy but also enhances response capabilities in cybersecurity operations.

Within the scope of this research, we have developed several machine learning and deep learning models by using these network-based features. Through this research, we aim to advance dynamic malware analysis methodologies, emphasizing automated, integrated approaches for real-time threat detection and response in cybersecurity. Recent developments in malware analysis highlight the need for adaptive systems that not only detect new and evolving threats but also counter evasion techniques and obfuscation strategies. By integrating our models with these dynamic methodologies, we strive to enhance both detection accuracy and processing speed, contributing to the development of resilient cybersecurity frameworks capable of addressing the complexity and scale of modern malware threats.

2. Related Works

In recent years, machine learning techniques have garnered significant attention for classifying malware and benign samples. For example, introduced risk signals to enhance Android security, achieving detection accuracies of 68.5% for benign applications and 93.38% for malicious programs [9]. Similarly, evaluated several machine learning techniques, including AdaBoost, Naive Bayes, Decision Trees, and Support Vector Machines, reporting that Naive Bayes could detect malware with an accuracy of 81%. [10].

Additionally, performed static analysis using a limited feature set and four supervised learning techniques, demonstrating that the random forest algorithm achieved a detection accuracy of 98.6% with a 1.8% false positive rate [11]. Likewise, in another study, six machine learning techniques are assessed to identify malware based on anomalies, utilizing feature selection techniques such as Chi-square and information gain to optimize feature sets. Their results indicated an impressive 99.9% detection accuracy when using the decision tree classifier on a custom malware dataset [12]. Lastly, a heterogeneous deep learning framework is proposed via combining AutoEncoders with multi-layer Restricted Boltzmann Machines (RBMs) to detect previously unknown malware using Windows API calls derived from portable executable profiles [13]. This framework consists of a two-phase process: pre-training and fine-tuning, where both labeled and unlabeled samples are utilized for feature learning. The authors conducted a comprehensive study on a large dataset from Comodo Cloud Security Center, demonstrating that their method outperformed traditional shallow learning approaches, although at the cost of increased system complexity. Notably, their work focused exclusively on API-based features for training the proposed method.

From the dynamic analysis perspective, a method is proposed decompiling malware specimens into assembly language to extract feature vectors that encode information regarding API calls and bytecode [14]. While this approach is intuitive, it is particularly sensitive to obfuscation techniques and incurs high computational costs for both decompilation and feature extraction. In contrast, in a study, it is suggested utilizing Q-learning for feature selection, where features are derived from the binary format and byte sequences [15]. Their approach successfully reduced a feature vector from a size of 4000 to 204 elements, although the resulting number of features remains substantial for training machine learning classifiers.

Further emphasizing the role of API calls, it is proposed logging API call sequences by encoding each API as a numerical value [16]. This sequence was then employed to train a two-layer long short-term memory network, achieving an impressive accuracy of around 98%. However, this method did not consider the argument values of the APIs, which could expand the input space and lead to potential misinterpretations of behavior.

Despite their utility, API call extraction can result in massive data volumes, posing challenges for indexing and querying. In response to this, it is suggested monitoring overall system behavior, asserting that detecting abnormal activities could facilitate the identification of various malware types, including zero-day, metamorphic, and polymorphic threats. [17]. This aligns with the approach taken by Dini et al. who introduced a real-time anomaly behavior detector using directed acyclic graphs, learning the host's execution behavior and encoding it into a model similar to a Markov Chain [18].

In a recent work, to detect Windows malware by executing programs in controlled environments and logging their behavior is carried-out. The findings of the work are open-sourced [19]. In this work, to address this, we introduce Nebula, a Transformer-based model that integrates diverse information from dynamic logs. Through experiments, Nebula outperforms state-of-the-art models in malware detection and classification, achieving up to 12% improvement. We also show that self-supervised pre-training rivals fully supervised models using just 20% of the data. In another recent research, over 1,500 malicious Android applications were examined, finding 18.31% equipped with anti-analysis techniques. To address this, the dynamic analyzer DOOLDA was introduced, effectively invalidating such techniques through dynamic instrumentation, outperforming other analyzers in identifying and neutralizing evasive behaviors [20]. Lastly, in Android devices, frequent attackers are aiming to steal data and push ads. While dynamic analysis is effective at detecting Android malware, many sandboxes like DroidBox rely on outdated emulators, making them vulnerable to evasion. To overcome this, DroidHook was introduced as an automated sandbox that supports multiple Android versions and works on real devices, providing more precise and fine-grained results than emulator-based tools [21].

3. Material and Methods

In this section, we provide an in-depth overview of the proposed methodology and framework used in our study. Our approach begins with the collection of network logs generated from malware and benign samples. The data set includes network traffic captured from both malicious and benign sources. We use a VirtualBox virtual machine running a Windows operating system, which is connected to a REMnux Linux environment for network emulation. This setup allows us to simulate realistic network conditions and capture the requisite data for analysis.

Subsequently, we extract a total of 39 features from these network logs, focusing on parameters that are

indicative of malicious activities. This feature extraction phase involves processing packet capture files to identify patterns and anomalies associated with malware. The extracted features encompass various aspects such as protocol usage, traffic volume, and connection patterns, which are critical for distinguishing between benign and malicious traffic.

Following feature extraction, we prepare the data for model training. We apply a rigorous preprocessing pipeline to handle missing values, scale features, and encode categorical variables. We then use this preprocessed data to train and evaluate several machine learning models. The models are trained on 80% of the features, with the remaining 20% reserved for validation to assess classification accuracy and model performance.

The overall framework of our methodology is illustrated in Figure 1, which provides a visual representation of the entire process from log collection to model evaluation. The subsequent subsections will delve into the specifics of each phase, including the setup of the analysis environment, detailed feature extraction techniques, and the training and testing of machine learning algorithms.

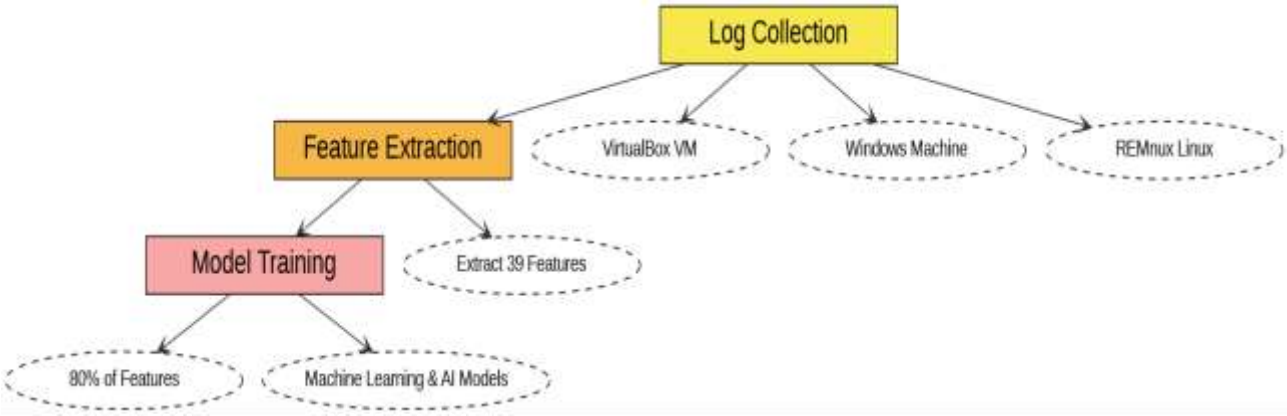

Figure 1. *Flowchart of the dynamic malware analysis methodology presented in this work.*

3.1 Data Set

The data set used in this study comprises a comprehensive collection of network logs derived from both benign and malicious sources. The benign samples were obtained from a free repository. This repository contains a vast array of normal .NET executable files collected from various reputable sources. Specifically, the repository is organized into several folders: files from the CNET website, netwindows, files from Windows, files from Softonic, files from SourceForge, and netexe files from other miscellaneous sources. These benign samples are well-documented in existing literature, including a notable reference [7] that utilized similar benign data for malware detection research.

For the malicious samples, we sourced data directly from a web-based malware repository, a well-known platform for malware analysis. The malware data set includes a diverse range of malicious executables, such as ELF botnets and Windows executables, each identified by unique hash values. Examples of the collected malware include botnets like Mirai and Moobot, as well as tools like KMSAuto. These samples represent various threats and are crucial for training and evaluating the robustness of our detection models.

In total, our data set consists of 953 benign network logs and 1408 malware logs. This diverse and extensive data set provides a robust foundation for extracting features and training machine learning models to improve malware detection accuracy.

3.2 Environmental Setup

In this study, the environmental setup consists of a host machine with an Intel Core i7-8700K CPU running at 3.70 GHz and 16 GB of RAM, operating on Windows 10 Pro that is located on TOBB ETU Cyber Security Labarotary. To create a controlled environment for malware analysis, a VirtualBox virtual machine (VM) is utilized. Within this VM, both benign and malicious samples are stored and executed. The VM is configured to revert to a clean state, known as "state zero," before each sample execution. This state ensures that the VM starts with the initial conditions, free from any residual effects of previous executions, providing a consistent and isolated environment. A custom program developed for this work selects the appropriate folders containing benign and malicious samples, runs them

within the VM, and extracts network-based logs, API calls, and system logs. This setup guarantees that each sample's execution is analyzed in an environment unaffected by previous malware activities. To illustrate the environmental setup, Figure 2 is presented below.

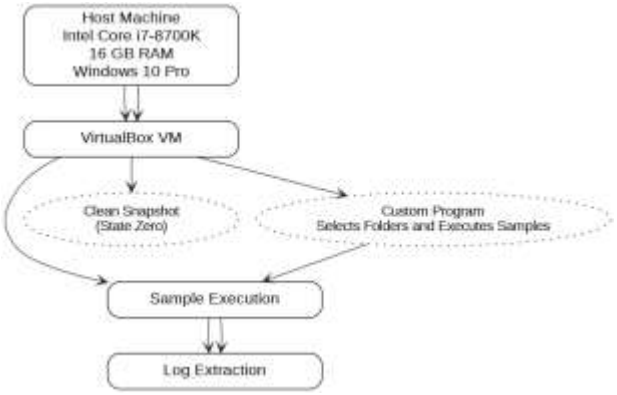

Figure 2. *Environmental setup and log extraction process.*

3.3 Execution Process

The execution process is carried out in two distinct stages. First, Dynamic Analysis: Each sample is executed within a clean state virtual machine (VM) to ensure isolation from previous malware actions. The VM is reset to a pristine "state zero" before every execution. A custom Python program is employed to run the samples, capture network traffic into .pcap files, and log API calls and system events. The logs are collected and analyzed to assess the behavior of each sample.

Second, the log extraction and analysis: After sample execution, the network logs in ".pcap" format and system logs in ".evtx" format are processed to generate comprehensive reports. The Python code provided manages the execution environment, captures network packets, and extracts system logs, ensuring a thorough examination of the sample behavior.

3.4 Feature Extraction

In this section, we describe the 39 features extracted from network logs and the preprocessing steps applied to them. The extraction process captures various aspects of network behavior and communication patterns essential for distinguishing between benign and malicious activities. We also provide reasoning for the relevance of each feature.

Number of Packets: Represents the total number of packets captured in the pcap file. Reasoning: A higher number of packets might indicate extensive communication or data transfer, which can be characteristic of both benign and malicious activities.

Total Size: The cumulative size of all packets in the pcap file. Reasoning: Large total sizes could suggest substantial data transfer, potentially signaling data exfiltration or large data dumps, common in malicious activities.

Mean Packet Size: Average size of packets. Reasoning: Anomalies in packet size, such as unusually large or small sizes, may indicate attempts to obfuscate traffic or unusual communication patterns.

Standard Deviation of Packet Size: Measures variability in packet size. Reasoning: High variability might be indicative of irregular or malicious traffic, where traffic patterns are intentionally varied to evade detection.

Mean Inter-arrival Time: Average time between successive packets. Reasoning: Regular or irregular intervals between packets can provide clues about the nature of the communication. Consistent intervals may be indicative of benign, predictable traffic, while irregular intervals might suggest malicious behavior.

Standard Deviation of Inter-arrival Time: Captures variability in the time between packets. Reasoning: High variability can point to unusual traffic patterns or communication attempts designed to evade detection.

TCP Packet Count: Total number of TCP packets. Reasoning: TCP packets are often used for establishing reliable connections. A high count might indicate extensive communication or data transfer.

UDP Packet Count: Total number of UDP packets. Reasoning: UDP is used for quicker, less reliable communication. A high count could indicate real-time communication applications or potentially malicious activities using less stringent connection protocols.

TCP Ratio: Ratio of TCP packets to the total number of packets. Reasoning: Provides insight into the proportion of reliable versus unreliable communication in the traffic, which can help differentiate between typical and anomalous patterns.

UDP Ratio: Ratio of UDP packets to the total number of packets. Reasoning: A higher UDP ratio might suggest applications that do not require reliable delivery, which can be characteristic of certain types of malicious activities.

TCP Flags (syn, ack, fin, psh, urg, rst): Counts of specific TCP flags. Reasoning: Different flags represent various stages and types of TCP connections. Anomalies or unusual counts in these flags can indicate attempts to establish connections or communicate in non-standard ways.

Unique Source Ips: Number of distinct source IP addresses. Reasoning: Multiple source IPs may suggest a distributed attack or botnet activity, while fewer unique IPs may indicate communication with a limited set of endpoints.

Unique Destination IPs: Number of distinct destination IP addresses. Reasoning: Many destination IPs might indicate a wide reach, potentially signaling a botnet or data exfiltration attempt.

Protocols Used: A list of protocols observed in the traffic. Reasoning: The variety of protocols used can provide insights into the types of services and applications involved. Malicious traffic may use uncommon or suspicious protocols.

DNS Query Count: Number of DNS queries made. Reasoning: High DNS query counts can indicate domain generation algorithms or attempts to contact command-and-control servers, often used in malware operations.

Unique Domain Count: Number of unique domains queried. Reasoning: A large number of unique domains might suggest attempts to evade detection or establish a connection with multiple servers.

DNS Query Types: Types of DNS queries observed. Reasoning: Different query types can reveal the nature of domain lookups and potential malicious domain resolution patterns.

Most Common TCP Port: The TCP port with the highest frequency of use. Reasoning: Common ports can indicate standard services, while deviations or unusual ports might suggest unconventional or malicious services.

Most Common UDP Port: The UDP port with the highest frequency of use. Reasoning: Similar to TCP ports, this helps identify standard versus unusual traffic patterns.

Unique TCP Ports: Number of distinct TCP ports used. Reasoning: The variety of ports used can provide insights into the complexity and diversity of communication, which may be higher in malicious traffic.

Unique UDP Ports: Number of distinct UDP ports used. Reasoning: Helps understand the range of services accessed or attempted to be accessed during communication.

Mean Entropy: Average entropy of payload data. Reasoning: High entropy might indicate encrypted or obfuscated data, which is often used to conceal malicious activities.

Total Payload Size: Cumulative size of payload data in packets. Reasoning: Larger payloads can signify substantial data transfers, which might be indicative of data exfiltration attempts.

Mean Payload Size: Average size of payload data. Reasoning: Helps understand typical data transfer sizes and identify deviations that may indicate anomalies.

Standard Deviation of Payload Size: Measures variability in payload size. Reasoning: Variability can indicate irregular data transfer patterns, which might be a sign of malicious behavior.

Minimum Payload Size: The smallest payload size observed. Reasoning: Identifies the smallest data chunks, which can be useful for understanding communication patterns.

Maximum Payload Size: The largest payload size observed. Reasoning: Provides insights into the largest data transfers, which might be indicative of significant or suspicious activities.

HTTP GET Method Count: Number of HTTP GET requests observed. Reasoning: High counts might indicate extensive web data retrieval, which can be typical in both benign and malicious activities.

HTTP POST Method Count: Number of HTTP POST requests observed. Reasoning: Indicates data submission activities, which could be normal or indicative of data exfiltration if unusually high.

HTTP PUT Method Count: Number of HTTP PUT requests observed. Reasoning: Used for file uploads, which might be a part of a benign application or an attempt to upload malicious content.

HTTP DELETE Method Count: Number of HTTP DELETE requests observed. Reasoning: Indicates attempts to remove data, which could be part of benign or malicious activities.

HTTP HEAD Method Count: Number of HTTP HEAD requests observed. Reasoning: Typically used to retrieve header information, and unusual frequencies might indicate probing or scanning activities.

HTTP OPTIONS Method Count: Number of HTTP OPTIONS requests observed. Reasoning: Provides information about communication capabilities, and high counts might suggest reconnaissance activities.

HTTP PATCH Method Count: Number of HTTP PATCH requests observed. Reasoning: Used for partial updates, and its presence can indicate modifications to resources, which could be part of an attack.

TCP SYN Flag Count: Number of TCP packets with the SYN flag set. Reasoning: SYN packets indicate connection attempts. High counts might suggest scanning or connection attempts.

TCP ACK Flag Count: Number of TCP packets with the ACK flag set. Reasoning: ACK packets represent acknowledgments. High counts might indicate ongoing communication or abnormal patterns.

TCP FIN Flag Count: Number of TCP packets with the FIN flag set. Reasoning: FIN packets indicate connection termination. Anomalous counts could suggest unusual connection behaviors.

TCP PSH Flag Count: Number of TCP packets with the PSH flag set. Reasoning: PSH packets indicate urgent data. High counts might be indicative of real-time or critical data transfers.

TCP URG Flag Count: Number of TCP packets with the URG flag set. Reasoning: URG packets indicate urgent data delivery. Unusual counts can be a sign of attempts to prioritize specific data.

3.5 Preprocessing Steps

First, a scaling and normalization operation is made. This process is very important especially for machine learning algorithms that use distance information since they are sensitive for the range of the features. So, the input features such as packet sizes, inter-arrival times, and payload sizes are scaled and normalized to ensure comparability. This helps improve the performance of machine learning models by bringing all features to a similar scale.

Secondly, categorical features are encoded. In this phase, the categorical features such as protocols and DNS query types are encoded into numerical values. This conversion is necessary for machine learning algorithms that require numerical input.

Third, missing data or zero values in features like entropy or payload sizes are handled by imputing default values or applying statistical techniques to ensure data completeness and consistency.

3.6 Model Training

In this section, we describe the training process for two different models used to classify network traffic data: a deep learning model and a Random Forest model. The Random Forest model, implemented using scikit-learn, was chosen for its robustness and ability to handle imbalanced datasets through ensemble methods.

3.6.1 Random Forest Model

The Random Forest Classifier was implemented using scikit-learn to address the classification of network traffic into benign and malicious categories. The initial step involved loading the dataset, which was stored in a CSV file containing extracted features. The dataset included both numerical and categorical features, which required preprocessing to prepare for model training.

The preprocessing pipeline used was constructed using the ColumnTransformer from scikit-learn. This pipeline standardized numerical features and applied one-hot encoding to categorical features. Numerical features were first imputed using the median value and then scaled using StandardScaler to normalize their distribution. Categorical features were handled by imputing missing values with 'unknown' and then one-hot encoded to convert them into numerical format suitable for machine learning algorithms.

The dataset was then split into training and validation sets using an 80 to 20 percent split ratio, ensuring that the class distribution was preserved in both sets. To address the class imbalance problem,

where the number of benign samples significantly exceeded the number of malware samples, the training set was balanced through downsampling.

The malware samples were resampled to match the number of benign samples, thus creating a balanced training dataset.

A Pipeline was created to streamline the preprocessing and model training process. This pipeline included the preprocessing steps followed by the Random Forest classifier. The model was trained on the balanced dataset and evaluated on the validation set. The final accuracy of the Random Forest model was approximately 96.31%, demonstrating its effectiveness in distinguishing between benign and malicious network traffic. To illustrate working mechanism of the model, Figure 3 is presented.

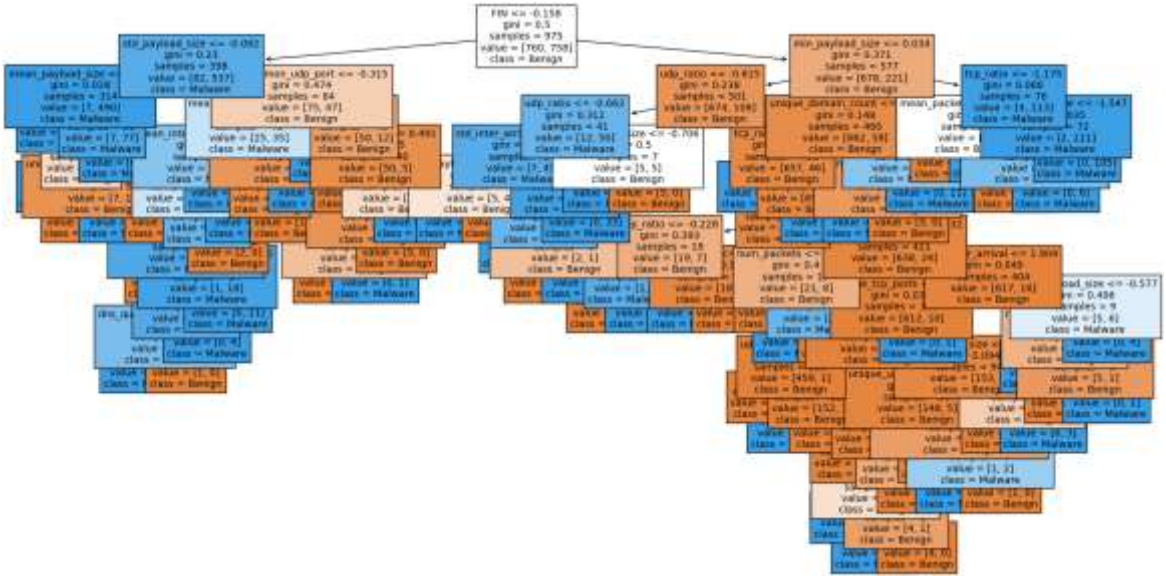

Figure 3. *Visualization of One Decision Tree in the random forest model*.

The figure visualizes a single decision tree from a Random Forest model, offering insights into the model's decision-making process. In a Random Forest, multiple decision trees are trained on various subsets of the data, each tree making decisions based on different features and thresholds. The decision tree in the figure is structured with a root node at the top, branching out into internal nodes and ultimately leading to leaf nodes. Each internal node represents a decision point where the data is split based on a feature and a threshold value. For example, a node might split the data based on feature_1 <= 5, indicating that data points with feature_1 values less than or equal to 5 follow one path, while those greater follows another.

As you move down the tree, each node's decision refines the classification, guiding the data through a series of splits. Leaf nodes, at the end of each path, provide the final classification outcome such as "Benign" or "Malware" based on the majority class of the data points that end up in that node. These leaf nodes also display class distribution and impurity measures, indicating the confidence and certainty of the model's predictions for those data points.

The depth of the tree and the number of splits illustrate the model's complexity. Deeper trees with many splits can capture intricate relationships in the data but may risk overfitting. Conversely, simpler trees are generally more interpretable but might not capture all complexities. By analyzing the features and thresholds at each node, along with the class distribution at the leaves, we can understand how the Random Forest model combines the decisions from multiple trees to make robust and accurate predictions. This visualization helps us interpret feature importance, decision rules, and the overall functioning of the Random Forest classifier in making predictions.

3.6.2 Deep Learning Model

In addition to the Random Forest model, a deep learning model that is presented in Figure 4 was also trained.

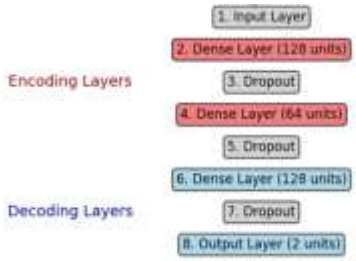

Figure 4. *Representation of the deep learning model*.

The process begins by loading the dataset from a specified CSV file into a Pandas DataFrame. This dataset contains features extracted from network traffic logs, with labels indicating whether each sample is benign or malicious. The data is subsequently processed to define which columns are categorical and which are numeric, facilitating the preprocessing steps that follow.

A comprehensive preprocessing pipeline is established using ColumnTransformer. For numerical columns, missing values are imputed using the median value, followed by standard scaling to normalize the data, ensuring that all numeric features contribute equally to the model's performance. Categorical features are also imputed, with any missing values replaced by a constant. “unknown”, and these columns are then one-hot encoded to convert them into a numerical format suitable for the model.

The dataset is split into training and validation sets, with 80% of the data used for training and 20% reserved for validation. This split is stratified based on the labels, ensuring that both classes are represented proportionally in each subset. To address class imbalance, the majority class is downsampled to match the size of the minority class. This step helps prevent the model from being biased towards the more prevalent class during training, and the data is shuffled to ensure randomness in the training process.

After balancing, the training and validation data are preprocessed using the defined pipeline, transforming the features into a format suitable for input into the neural network. An encoder-decoder architecture is constructed using Keras. The model consists of an input layer followed by two dense layers with ReLU activation functions, each followed by a dropout layer to reduce overfitting. This section of the model compresses the input features into a lower-dimensional representation. The decoder mirrors the encoder structure, expanding the learned representation back to the output dimensions. The final layer uses a softmax activation function to produce class probabilities for benign and malware classifications.

The model is compiled with the Adam optimizer and a loss function suited for multi-class classification, the sparse categorical crossentropy, while accuracy is tracked as a performance metric during training. The model is trained on the processed training data with a validation set to monitor performance, incorporating early stopping to halt training if the validation loss does not improve for a specified number of epochs, thereby preventing overfitting.

After training, the model is evaluated on the validation set to determine its accuracy, providing insights into its performance in classifying unseen data and reflecting its generalization capabilities. This encoder-decoder model architecture is designed to effectively learn from the complex patterns in network traffic data, making it a robust solution for malware detection tasks in cybersecurity.

4. Results and Discussions

In this study, we developed and evaluated two algorithms: a Random Forest classifier and a deep learning model. Both models were trained and tested on the same dataset, adhering to an 80/20 train-validation split.

The performance of the Random Forest model was promising, as evidenced by the confusion matrix and classification report. The model achieved a precision of 0.99 for benign samples and 0.97 for malware samples. The recall was similarly high, with values of 0.97 and 0.99, respectively. The F1-scores for both classes were approximately 0.98, indicating a strong balance between precision and recall. Overall, the Random Forest model achieved an accuracy of 98% on the validation set, demonstrating its effectiveness in distinguishing between benign and malware samples. The confusion matrix is presented in table 1.

Table 1. *Confusion matrix for the random forest model.*

	Predicted Benign	**Predicted Malware**
Actual Benign	184	6
Actual Malware	2	188

In contrast, the deep learning model exhibited even greater performance metrics. The classification report for the deep learning model indicated a precision of 0.99 for both classes, with recall values of 0.98 for benign samples and 0.99 for malware samples. The F1-scores for the deep learning model were also impressive, reaching 0.98 for benign and 0.99 for malware. This model achieved an overall accuracy of 99% on the validation set, reflecting its capability to accurately classify the data. The confusion matrix is presented in table 2.

Table 2. *Confusion matrix for the random forest model.*

	Predicted Benign	**Predicted Malware**
Actual Benign	186	4
Actual Malware	1	189

The results from both models indicate that machine learning and deep learning approaches can effectively classify network traffic as benign or malicious. While the Random Forest model provided robust performance, the deep learning model's slightly higher accuracy and consistent precision and recall metrics highlight its potential advantages for complex classification tasks in cybersecurity. Further research may explore the integration of these models into real-time detection systems to enhance cybersecurity measures.

5. Conclusions

In this study, we developed and evaluated two distinct algorithms for malware detection: a Random Forest model and a deep learning model based on an encoder-decoder architecture. Both models were trained and tested using the same dataset, ensuring a fair comparison of their performance.

The random forest model demonstrated impressive results, achieving a classification accuracy of 98% on the validation set. The confusion matrix indicated that the model effectively distinguished between benign and malware samples, with high precision and recall values for both classes. This confirms the model's robustness in identifying malicious activities while minimizing false positives.

In contrast, the deep learning model exhibited even better performance, achieving an accuracy of 99% on the same validation set. The classification report revealed excellent precision and recall metrics, indicating the model's capability to generalize well across unseen data. This highlights the effectiveness of deep learning techniques in handling complex patterns within the dataset.

The results suggest that both algorithms are viable options for real-time malware detection systems. However, the deep learning model's superior performance may make it a more suitable choice for applications requiring higher accuracy and reliability. Future work will focus on optimizing these models further, exploring additional feature extraction techniques, and conducting tests in real-world scenarios to assess their practical applicability in cybersecurity environments.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] National Institute of Standards and Technology. (n.d.). *Glossary of key information security terms*. Retrieved September 10, 2024, from https://csrc.nist.gov/Glossary/?term=5373

[2] TechTarget. (n.d.). *Malware*. Retrieved September 10, 2024, from https://searchsecurity.techtarget.com/definition/malware

[3] BullGuard. (n.d.). *Malware definition, history, and classification*. Retrieved September 10, 2024, from https://www.bullguard.com/bullguard-security-center/pc-security/computer-threats/malware-definition,-history-andclassification.aspx

[4] Kaspersky. (n.d.). *What is malware and how to protect against it*. Retrieved September 10, 2024, from https://www.kaspersky.com/resource-center/preemptive-safety/what-is-malware-and-how-to-protect-against-it

[5] Norton. (n.d.). *Malware*. Retrieved September 10, 2024, from https://us.norton.com/internetsecurity-malware.html

[6] MalwareBazaar. (n.d.). *Free automated malware analysis platform*. Retrieved from https://malwarebazaar.com/

[7] Bormaa. (n.d.). *Open-source benign samples*. Retrieved from https://github.com/bormaa/Benign-NET

[8] Szor, P. (2005). *The art of computer virus research and defense*. Pearson Education.

[9] Bhaskar Pratim Sarma, Ninghui Li, Chris Gates, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Android permissions: a perspective combining risks and benefits. *In Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 13–22. ACM, 2012.

[10] Chun-Ying Huang, Yi-Ting Tsai, and Chung-Han Hsu. Performance evaluation on permission-based detection for android malware. *In Advances in Intelligent Systems and Applications-Volume 2*, pages 111–120. Springer, 2013.

[11] Rushabh Vyas, Xiao Luo, Nichole McFarland, and Connie Justice (2017). Investigation of malicious portable executable file detection on the network

using supervised learning techniques. *In Integrated Network and Service Management (IM), IFIP/IEEE Symposium*, pages 941–946. IEEE, 2017.

[12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and YaelWeiss (2012). andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190.

[13] Yanfang Ye, Lingwei Chen, Shifu Hou, William Hardy, and Xin Li (2017). Deepam: a heterogeneous deep learning framework for intelligent malware detection. *Knowledge and Information Systems*, pages 1–21.

[14] D. Gibert, C. Mateu, and J. Planes (2020). HYDRA: A multimodal deep learning framework for malware classification. *Comput. Secur*, 95; 101873.

[15] Z. Fang, J. Wang, J. Geng, and X. Kan (2019). Feature selection for malware detection based on reinforcement learning. *IEEE Access,*7;176177–176187.

[16] F. O. Catak, A. F. Yazı, O. Elezaj, and J. Ahmed (2020). Deep learning based sequential model for malware analysis using windows exe API calls. *Peer J. Comput. Sci., 6;*e285, doi: 10.7717/peerj-cs.285.

[17] C. M. Chen, G.-H. Lai, T.-C. Chang, and B. Lee (2020). Detecting pe-infection based malware. *in Proc. Future Inf. Commun. Conf. Cham, Switzerland: Springer*, pp. 774–781.

[18] M. E. Ahmed, S. Nepal, and H. Kim (2018), "MEDUSA: Malware detection using statistical analysis of system's behavior," *in Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, pp. 272–278.

[19] Trizna, Dmitrijs et al. (2024) Nebula: Self-Attention for Dynamic Malware Analysis. *IEEE transactions on information forensics and security, 19,* DOI 10.1109/TIFS.2024.3409083

[20] Lee, Sunjun et al. (2024). Hybrid Dynamic Analysis for Android Malware Protected by Anti-Analysis Techniques with DOOLDA. *Journal of internet technolog,* 25(2). DOI 10.53106/160792642024032502003

[21] Cui, Yuning et al. (2023). DroidHook: a novel API-hook based Android malware dynamic analysis sandbox. *Automated software engineering*, 30(1). DOI 10.1007/s10515-023-00378-w