Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.052637

ARTICLE

Multi-Binary Classifiers Using Optimal Feature Selection for Memory-Saving Intrusion Detection Systems

Ye-Seul Kil[1,#], Yu-Ran Jeon[1,#], Sun-Jin Lee[1] and Il-Gu Lee[1,2,*]

[1]Department of Future Convergence Technology Engineering, Sungshin Women's University, Seoul, 02844, Republic of Korea

[2]Department of Convergence Security Engineering, Sungshin Women's University, Seoul, 02844, Republic of Korea

*Corresponding Author: Il-Gu Lee. Email: iglee@sungshin.ac.kr

[#]Ye-Seul Kil and Yu-Ran Jeon contributed equally

Received: 10 April 2024 Accepted: 14 July 2024 Published: 27 September 2024

ABSTRACT

With the rise of remote work and the digital industry, advanced cyberattacks have become more diverse and complex in terms of attack types and characteristics, rendering them difficult to detect with conventional intrusion detection methods. Signature-based intrusion detection methods can be used to detect attacks; however, they cannot detect new malware. Endpoint detection and response (EDR) tools are attracting attention as a means of detecting attacks on endpoints in real-time to overcome the limitations of signature-based intrusion detection techniques. However, EDR tools are restricted by the continuous generation of unnecessary logs, resulting in poor detection performance and memory efficiency. Machine learning-based intrusion detection techniques for responding to advanced cyberattacks are memory intensive, using numerous features; they lack optimal feature selection for each attack type. To overcome these limitations, this study proposes a memory-efficient intrusion detection approach incorporating multi-binary classifiers using optimal feature selection. The proposed model detects multiple types of malicious attacks using parallel binary classifiers with optimal features for each attack type. The experimental results showed a 2.95% accuracy improvement and an 88.05% memory reduction using only six features compared to a model with 18 features. Furthermore, compared to a conventional multi-classification model with simple feature selection based on permutation importance, the accuracy improved by 11.67% and the memory usage decreased by 44.87%. The proposed scheme demonstrates that effective intrusion detection is achievable with minimal features, making it suitable for memory-limited mobile and Internet of Things devices.

KEYWORDS

Endpoint detection and response; feature selection; machine learning; malware detection

Nomenclature

EDR	Endpoint detection and response
Conv-All	Conventional multi-classification model using all features
Conv-6	Conventional multi-classification top-6 model using selected top-6 features with high importance weights

1 Introduction

Cyberattacks are on the rise worldwide, accompanying the accelerated digital transformation of companies following the coronavirus disease of 2019 (COVID-19) pandemic. During COVID-19, many ransomware, distributed denial of service (DDoS), and phishing attacks occurred, and the main targets were medical institutions. Most hospital systems are composed of computer systems with low-end operating systems; therefore, they were directly attacked by advanced persistent threat (APT) 29 and others during COVID-19 [1].

As medical institutions are directly connected to life, strong security for information technology (IT) systems is required. In addition, with the development of Industry 4.0, cyberattacks have become more advanced, exhibiting various attack paths and types as well as complex characteristics. Conventional malware is operationally simple, does not cause heavy damage, and is easily detectable. However, advanced malware exhibits complex operational structures and performs long-term attacks, causing massive damage to specific targets. Thus, detecting advanced cyberattacks using conventional intrusion detection methods is challenging [2].

Typical conventional intrusion detection methods include the signature-based method [3–5], which detects attacks based on predetermined detection rules corresponding to previously detected attacks. However, these methods can only detect previously identified attacks; they cannot respond adequately to novel malware variants [6]. Machine learning (ML)-based intrusion detection methods, which can actively detect malware to overcome this limitation, are divided into binary classification models, which determine the existence of an attack, and multi-classification models, which classify malware into detailed types. However, the practical use of these classification models is limited because their detection accuracy decrease as the number of malware types increases.

Endpoint detection and response (EDR) tools, which monitor security threats at endpoints in real-time, have garnered significant attention for overcoming the limitations of conventional methods. These tools collect data to enable automated responses to threats. They facilitate automatic real-time monitoring; however, they constantly generate massive security alarms, including unnecessary data, which wastes memory resources and degrades intrusion detection performance [7]. Thus, existing intrusion detection systems are not optimized for detection performance and resources used owing to the increase in types of malicious code.

This paper is a supplementary and extended version of the paper presented at the 7th International Symposium on Mobile Internet Security (MobiSec'23) Conference. This study proposes a binary classifier-based malware multi-classification technique using optimal features for each attack type to address the limitations of conventional studies. A ransomware classifier and a trojan classifier, which are binary classifiers that detect individual attacks, were configured in parallel. Subsequently, the results of the two classifiers were comprehensively analyzed based on their classification probabilities. The proposed model can improve the accuracy and memory usage compared to conventional models.

The main contributions of this study are as follows:

- A malware detection technique based on parallel binary classifiers and optimal feature selection that addresses the limitations of conventional detection methods was proposed.
- It was proven that binary classifiers alone can detect multiple types of malicious attacks.
- The accuracy and memory usage of the proposed model were improved compared to conventional models, enabling memory-efficient attack detection.

The remainder of this paper is organized as follows: First, previous related works on conventional feature selection methods and malware detection are discussed in Section 2. In Section 3, the proposed malware detection method is introduced; its performance improvements are demonstrated in Section 4 by describing comparative experiments with conventional models. Finally, Section 5 concludes the paper and discusses future research directions.

2 Related Work

Intrusion detection methods can be classified into tool-based and ML-based intrusion detection methods. Tool-based intrusion detection methods use a malicious behavior detection tool, such as EDR, to detect attacks. In contrast, ML-based intrusion detection methods classify abnormal traffic by training on network traffic data, including the Internet Protocol (IP) address and port information. Table 1 lists the contributions and limitations of previous studies regarding the two types of intrusion detection methods.

Table 1: Previous studies on intrusion detection methods

Category	Ref.	Contributions	Limitations
Tool-based intrusion detection	[8]	Google Rapid Response (GRR) and Osquery were used together to detect APT attacks. The APT attack detection coverage was evaluated using the MITRE adversarial tactics, techniques and common knowledge (ATT&CK) framework.	The APT attack was configured as a simple attack; hence, the detection performance for complex attacks could not be confirmed. In addition, intrusion detection was not performed using various pieces of log information.
	[9]	GRR, Osquery, and open-source host-based intrusion detection security (OSSEC) were used to detect RAASNet ransomware attacks. In addition, the intrusion detection technique of each EDR tool was analyzed, and the differences among and the differences between the tools were identified.	As an integrity-based intrusion detection method was used, detecting advanced attacks was difficult.

(Continued)

Table 1 (continued)

Category		Ref.	Contributions	Limitations
		[10]	GRR and Graylog were combined to detect APT attacks; the limitations of the GRR tool were mitigated, and the detection coverage was extended.	Both the GRR and Graylog tools were used manually; thus, the delay was increased.
		[11]	GRR and Auditbeat were used to monitor the behavior of APT attacks and collect log data, providing earlier detection than conventional methods with higher accuracy.	As additional tools were used, more logs were collected than with conventional methods.
ML-based intrusion detection	Binary classification	[12]	A model was proposed to detect TrickBot, the first banking trojan horse. Dynamic analysis of malicious programs was performed to collect packets, and the performance was evaluated using random forest (RF), multilayer perceptron (MLP), sequential minimal optimization (SMO), and logistic regression (LR) models.	The proposed model struggled to detect attack types other than trojan horses, and an imbalanced dataset was used in the experimental evaluation.
		[13]	The deep learning-based binary fruit fly algorithm (DL-BFFA) model was proposed for synchronization (SYN) flooding attack detection.	A single class was used for DoS, Probing, u2R, and remote attacks, rendering the classification of various attack types difficult.
		[14]	A dynamic malware detection algorithm was proposed using feature selection and ML based on genetic algorithms.	The proposed method only determines the presence or absence of malware; performance may be degraded if various types of attacks are performed.

(Continued)

Table 1 (continued)

Category		Ref.	Contributions	Limitations
		[15]	Frequency differential selection (FDS) and weight measurement were used with newly proposed feature selection algorithms to detect Android malware.	FDS mainly considers the characteristics of malware that may occur on Android; its performance may be poor in detecting malware in other operating system (OS) environments.
		[16]	A semi-supervised federated IoT malware detection framework was proposed to address privacy issues in collecting user logs.	The authors did not measure the memory efficiency, making it difficult to confirm its performance in an IoT environment.
		[17]	An automated feature selection method for behavior-based ransomware detection was proposed, and a particle swarm optimization (PSO) algorithm with wrapper-based feature selection for ransomware detection was used.	The performance was lower when the malware classification model was applied to multi-classification than when it was applied to binary classification.
	Multi-classification	[18]	DDoS attack types in the KDD99 dataset were classified using support vector machine (SVM), decision tree (DT), naïve Bayes (NB), and unsupervised ML (USML) models. An accuracy of 94.78% was revealed for the USML model.	Only the sub-types of DDoS attacks were classified. In addition, multi-classification with SVM exhibited increased overheads (latency and memory usage) during training and testing compared to a binary classifier.
		[19]	RF, DT, k-nearest neighbor (KNN), and ridge classifier (RC) were used to classify DoS, Probing, u2R, and remote attacks, and the PSO method was applied to optimize the performance.	The performance improvement using PSO was not evaluated.

(Continued)

Table 1 (continued)

Category	Ref.	Contributions	Limitations
	[20]	A behavior-based ransomware classification model was proposed, and feature selection was performed by removing features with similar attributes to increase classification accuracy.	The proposed model manually selected a feature for every training run.
	[21]	A multilayer architecture was proposed to perform classification in three stages and to maintain high performance in noisy environments.	The memory usage and latency increased with the use of the three-stage model.
	[22]	Proposed feature extraction algorithm reduced the dimensionality of high-dimensional features, thus improving the detection speed.	The proposed method did not consider the behavioral characteristics of each attack.
DL-based intrusion detection	[23]	A combined convolutional neural network (CNN) and bidirectional long short-term memory (BiLSTM) model was proposed to solve the problem of the low detection rate caused by high-dimensional data.	The process of learning and training by combining two models uses a large amount of resources.
	[24]	A low-complexity DL-based intrusion detection system for IoT nodes was proposed.	Insufficient consideration of computing costs that occur when learning large amounts of data
	[25]	The proposed algorithm selects important features using high-level and low-level features.	The runtime calculation criteria are not clearly explained.
	[26]	To overcome limitation of Transnet, the proposed algorithm accelerated the feature learning process.	Selecting the transfer method of the feature learning process is not automated.

Park et al. [8] proposed a tool-based intrusion detection method and suggested a framework for detecting APT attacks by combining GRR and Osquery. They also evaluated the APT attack coverage regarding the tactics and techniques provided by MITRE ATT&CK. The detection coverage using the proposed method has been expanded by integrating multiple tools; the intrusion detection performance was not verified for complex attacks because the APT attacks were composed of a combination of simple malware. In addition, various pieces of log data were not used for intrusion detection. Lee et al. [9] used GRR, Osquery, and OSSEC to detect RAASNet ransomware. The authors analyzed the characteristics of each tool, which enabled faster attack detection using EDR tools. However, an integrity-based intrusion detection method was primarily used in their study, making it challenging to detect attacks bypassing integrity. This method also exhibits the limitation of not using various pieces of log data for detection. Jeon et al. [10] proposed a method for detecting APT attacks by combining Graylog and GRR to address the limitation of GRR that only some data types can be collected owing to limited output plug-ins. An APT attack environment was configured using the Carbanak attack scenario emulator and the attack detection coverage was measured. The MITRE ATT&CK detection range of the proposed method was expanded by 11% compared to the conventional method. However, although the GRR and Graylog tools were interworked, the two tools were each used manually. Park et al. [11] proposed a rapid detection model using GRR and Auditbeat. As GRR monitors the behavior of APT attacks and Auditbeat collects specific log data, the proposed method can provide earlier detection than conventional methods with higher accuracy. However, more logs are collected compared to conventional methods because additional tools are used to detect APT attacks. The traditional tool-based intrusion detection methods are restricted by the continuous generation of unnecessary logs, degrading detection performance. To overcome these challenges in tool-based intrusion detection, our study applies an optimal feature selection and ensures that only a small number of features, optimized for each type of attack, are used.

Gezer et al. [12] proposed an ML-based detection method to collect packets and perform dynamic analysis of the environment of TrickBot, which is a banking trojan horse targeting bank customers globally. Subsequently, ML models were used to classify TrickBot traffic flows. TrickBot infects the hosts of victims with binary malware, injects malware into application programs, and then leaks user data via redirection. In experiments using RF, MLP, SMO, and LR models, the RF model was observed to classify attacks with an accuracy of 99.94%. The study provided insights into the detection of a TrickBot infection, but it only evaluated trojan attacks, and other types of attacks were not considered. Nevertheless, as an imbalanced dataset was used for evaluation, higher accuracy than that with a balanced dataset was obtained. Nagaraju et al. [13] proposed the DL-BFFA to classify SYN flooding attacks, which is a type of DDoS attack that acquires the handshake process of the Transmission Control Protocol. The BFFA model was used to select optimal parameters in a DL model, and binary BFFA was applied to the binary classification model. A comparison of the performances of a simple SYN network model, tuning DL using the hybrid bat algorithm, the binary bat algorithm, and the DL-BFFA revealed that the proposed model achieved the highest accuracy of 99.96%. However, the study proposed a binary classifier that detects only SYN flooding attacks; thus, its detection performance may be poor in an environment with various types of attacks. Irshad et al. [14] proposed a genetic algorithm-based feature selection method and a dynamic malware detection algorithm using ML. They used Cuckoo and the genetic algorithm for malware behavior analysis to select the optimal features. Experiments with SVMs, NB, and RF revealed accuracies of 81.3%, 64.7%, and 86.8%, respectively. However, the algorithm proposed in [14] is an intrusion detection method that determines only the presence or absence of malware, which limits its performance when handling various types of attacks. Sun et al. [15] used newly proposed feature selection

algorithms, FDS, and weight measurement for Android malware detection. The representative data required by the Android application function were analyzed, and the FDS algorithm, which aims to increase accuracy while reducing complexity by selecting optimal features, was applied. Reference [15] applied the FDS algorithm to extract the feature importance and remove irrelevant features, but the proposed FDS is a feature selection algorithm that considers the characteristics of malware that can occur only on Android, and it may be poor at detecting malware in other operating system (OS) environments. In addition, only the performance results for malware detection could be verified; the results for the malware classification performance of multiple classes could not be validated. Pei et al. [16] proposed a semi-supervised federated IoT malware detection framework to address privacy issues in collecting user logs and ensure the quality and reliability of data labels. The authors utilized knowledge transfer techniques to infer labels by analyzing the correlation between unlabeled and labeled data. This study proposed an IoT malware detection system that ensures privacy and reliability, but it did not measure the memory performance of the proposed model, making it difficult to confirm its performance in a resource-constrained IoT environment. Abbasi et al. [17] proposed an automated feature selection method for behavior-based ransomware detection and classification. PSO algorithms were used for ransomware detection and classification with wrapper method-based feature selection. According to the experimental results, the RF-based binary classification and multi-classification exhibited detection accuracies of approximately 97.06% and 55.33%, respectively. In previous studies [12–16], a binary classifier that detects only specific attacks was proposed, limiting the intrusion detection performance when multiple types of attacks are introduced. In addition, Abbasi et al. [17] showed that when a malware classification model was applied to multi-classifiers, the performance was 43% lower than when it was applied to binary classifiers.

Extensive multi-classification research has been conducted to overcome the limitations of binary classification. Tuan et al. [18] evaluated the attack classification performances of SVM, NB, DT, and USML models for botnet DDoS attacks. The experimental evaluation indicated that the classification accuracies of SVM, DT, NB, and USML were 91.55%, 93.3%, 96.74%, and 98.08%, respectively. This study provided guidelines for choosing a suitable ML method for each case study, but only sub-types of DDoS attacks were considered. In addition, among the models used in this study, multi-classification with SVMs exhibited increased overheads (latency and memory usage) during training and testing compared to a binary classifier. Saheed et al. [19] applied a PSO algorithm to ML models to perform multi-classification of attacks in the Internet of Medical Things environment. This study applied the PSO algorithm, a meta-heuristic optimization algorithm, and selected optimal features from a dataset. PSO-RF, PSO-DT, PSO-KNN, and PSO-RC achieved accuracies of 99.76%, 99.58%, 98.9%, and 97.6%, respectively. However, because a performance comparison between the simultaneous and independent use of feature selection and PSO was not conducted, proving the effect of the application of PSO was difficult. Sethi et al. [20] proposed a behavior-based ransomware classification model, classifying 10 families. They used behavior features with weights of high importance for high classification accuracy. Subsequently, the calculated behavior features were grouped with those with similar properties, and features with similar properties were removed. In contrast to previous studies that only considered the importance weights of features, overlapping features were considered through feature grouping; however, the features must be selected manually for every training run. Piskozub et al. [21] proposed a multi-layer architecture for malware detection. They classified malware through a three-stage process of binary, malware type, and malware family classification. This method proved robust to noise when the number of malicious flows is relatively large. However, the proposed architecture has limitations in that it uses memory excessively and increases the latency through

the three-step classification process. Zhang et al. [22] efficiently reduced the dimensionality of high-dimensional features using normalized mutual antibodies information feature selection (NMAIFS) and implemented a classifier using the best feature vectors. In this study, the NMAIFS algorithm improved the detection speed but did not consider the behavioral characteristics of each attack, so it has the limitation of using unnecessary features when multi-classifying each attack. The conventional ML-based intrusion detection techniques lack optimal feature selection for each attack type. To overcome these limitations in ML-based intrusion detection, our study configures binary classifiers for each attack type and uses features optimized for each type of attack.

In addition, DL-based intrusion detection has been studied recently. Hnamte et al. [23] pointed out that conventional DL-based intrusion detection systems often use binary classification. At this time, the attack detection rate of the training model may decrease owing to the influence of high-dimensional data. Therefore, this study presents a methodology that combines a CNN and BiLSTM with more hidden layers. This method enables the LSTM model to process the input sequence based on its structure to design a robust model. However, a limitation exists in that it uses a large amount of resources during learning and training by combining the two models. Awajan [24] proposed a real-time intrusion detection mechanism for Internet of Things (IoT) nodes. By using a four-layer fully connected layer architecture, the complexity of the system was reduced, and a precision of 93% was achieved for actual intrusions. However, the computing costs arising from learning large amounts of data were not considered. Zhang et al. [25] proposed a pyramid channel-based feature attention network (PCFAN) that selects important features by simultaneously utilizing high-level and low-level features. Utilizing the complementarity between features at different levels for single image restoration, it improved the problem of ignoring low-level features in the conventional deep learning-based image restoration method. However, the PCFAN proposed in this study considers features at all levels and consists of multiple modules, which may increase complexity, and the runtime calculation criteria are not clearly explained, so the applicability in real environments has not been proven. Hu et al. [26] proposed a deep subdomain adaptation network with an attention mechanism (DSAN-AT) to identify malware variant traffic accurately and efficiently at the IoT edge gateway. The proposed DSAN-AT reduces computing resources at the IoT edge gateway by accelerating feature learning of traffic classified into different domains. This study claims to improve the slow convergence speed and memory limitation of Transnet, a state-of-the-art technique for malware variant traffic classification. However, this study has limitations in that the selecting transfer method of the feature learning process is not automated. To overcome these challenges in DL-based intrusion detection, our study measures memory usage to demonstrate effectiveness in IoT environments. Additionally, we implement automation in the feature selection process to better detect variant malware in realistic scenarios.

In summary, conventional intrusion detection methods suffer from various limitations. Among ML-based methods, the binary classification method classifies data into only benign and attack classes, making individual responses to different attack types difficult. Thus, with the progressive diversification of attacks and increasing similarity to benign behavior, its classification performance is likely to be degraded. However, as the multi-classification method uses the same features simultaneously during the training process for classification, its detection accuracy is lower than that of the binary classification model. In addition, higher memory usage is required to handle more classes. Finally, tool-based intrusion detection can only detect attacks that are defined by the rule set, making it challenging to detect advanced attacks.

3 Proposed Method

The structure of the proposed method is illustrated in Fig. 1.

1) Data collection: The EDR tool GRR [27] collects malicious logs of ransomware and trojan attacks. Benign logs are collected by executing everyday actions, such as searching, downloading, and streaming, after logging into a personal computer.

2) Data preprocessing: The extracted data are preprocessed to derive 18 features.

3) Feature selection: The importance weights of the ransomware and trojan classifiers are calculated to select optimal features for each attack type.

4) Classification: The classification comprises two sub-classifiers. The ransomware and trojan classifiers simultaneously classify each malware type during the first phase classification. During the second-phase classification, the classification results of the ransomware and trojan classifiers are combined to determine the final classification into ransomware, trojan, or benign classes.

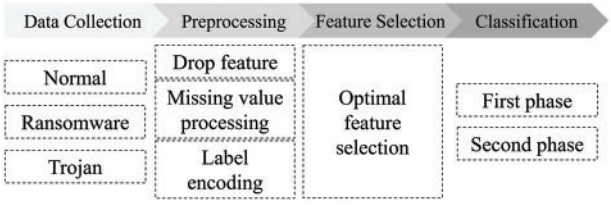

Figure 1: The structure of proposed method

3.1 Data Collection

An experimental environment was constructed using GRR to collect logs, as shown in Fig. 2. A GRR server and GRR clients were installed on virtual machine software (VMware). GRR version 3.4.0-1 was used for the GRR clients. In addition, Windows and Linux clients were employed. In an attack environment, the GRR server monitors the behavior of the client and collects relevant log data, while the client acts as the victim being infected by the malware. When the attacker deploys and runs malware on the client, the GRR server collects the behavior of the client.

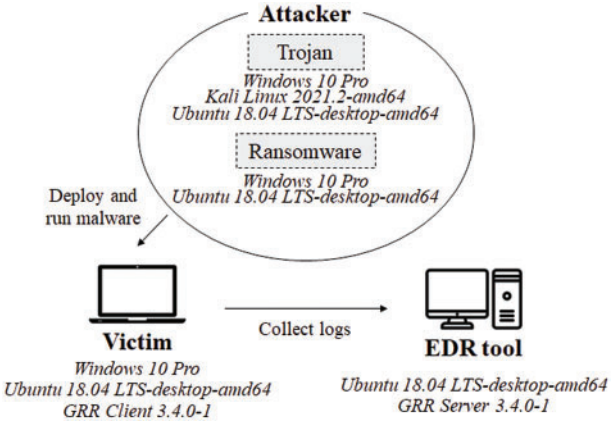

Figure 2: Data collection environment

Considering the significant social impact and recent increase in the prevalence of ransomware and trojan, we assumed an environment infected with these two types of malware as our attack scenario [28]. MSFVenom [29], which is a Metasploit framework-based independent payload generator, and MEMZ trojan [30], which is a trojan that is tailored to the Windows environment, were used to collect trojan logs. MSFVenom was executed in the Ubuntu environment after creating a backdoor shell in Kali Linux. Experiments on MEMZ trojan were performed after constructing an attack environment in Ubuntu and Windows. PSRansom [31], which is a PowerShell-based ransomware simulator that provides C2 server functionality, was used as an attack environment for collecting ransomware logs. It was executed in the Windows and Ubuntu environments, and logs were collected using the EDR tool. Benign logs were collected using the EDR tool after performing benign operations in the environment of the victim. In addition, we assumed security while collecting the ransomware, trojan, and benign logs and that the GRR Server could be trusted.

3.2 Preprocessing

The dataset comprised logs that are extracted from Netstat, FileFinder, and Process in the GRR flow. Netstat is a type of flow artifact provided by the GRR server, and information regarding the network status flow of the attacker can be obtained by tracking the attack timestamps of the Windows attacker. FileFinder tracks file paths to provide information regarding the creation, deletion, and changes of a specific file. Process provides information regarding a specific process; the process name, execution time, and username of the process can be obtained based on these features [28]. For preprocessing of the dataset, some features unrelated to classification were dropped, as they were device information, system configuration, and system time features.

In addition, privacy-related features were dropped and 18 features were ultimately selected. The selected features were related to the standard system or process, and features containing personal information were not collected. Thereafter, missing values in the data were filled in with zeros and label encoding was applied to the categorical variables. Table 2 lists the 18 dataset features derived through the data preprocessing.

Table 2: Features of dataset

Type	Feature	Description
Process	memory_percent	Percentage of memory used
	rss_size	Memory information used by process
	vms_size	Virtual memory size
	num_threads	Number of threads used
	user_cpu_time	Central processing unit (CPU) time consumed for user's task
	system_cpu_time	System CPU time
	status	Process status information
	real_uid	User identifier information used to execute process
	real_gid	Group identifier information used to execute process
Netstat	state	Network state information
	local_address.ip	Target system's IP information
	remote_address.port	Remote port information
FileFinder	urn	Event path
	st_blocks	Number of blocks allocated to file

(Continued)

Table 2 (continued)

Type	Feature	Description
	st_mode	File's mode (read, write, execute) information
	st_ino	File's inode number
	st_size	File's size
	modified	File's integrity information

3.3 Feature Selection

In previous studies [18–22], which applied the same feature selection method irrespective of the malware type, the accuracy decreased with increasing malware diversity, making the detection of various advanced attacks difficult. For example, the primary malicious behavior of ransomware is file encryption. In addition, a trojan is the connection of an attacker to a session to upload malware or introduce a backdoor. As the characteristics of malware types differ, some features may cause noise during specific malware training if the same features are used in large numbers for various malware types. Furthermore, in the case of a recent DL-based intrusion detection system [23–26], experiments were conducted using the same features regardless of the malware type. As most research focuses only on achieving high performance using fixed datasets, little research has been conducted on developing systems to prepare for intelligent attacks. Therefore, the proposed model performs feature selection with ransomware and trojan classifiers to train the model using optimal features for each attack type classification.

First, the feature importance for each classifier is derived using permutation importance [32]. Permutation importance calculates the importance weight of each feature by evaluating the predictive performance of the verification dataset when data from a specific feature are shuffled after training the model. When calculating the importance of permutation, we performed five-fold cross-validation by dividing the entire dataset into training and validation sets. Fig. 3 presents a visualization of the feature importance weights of the ransomware and trojan classifiers for the 18 features.

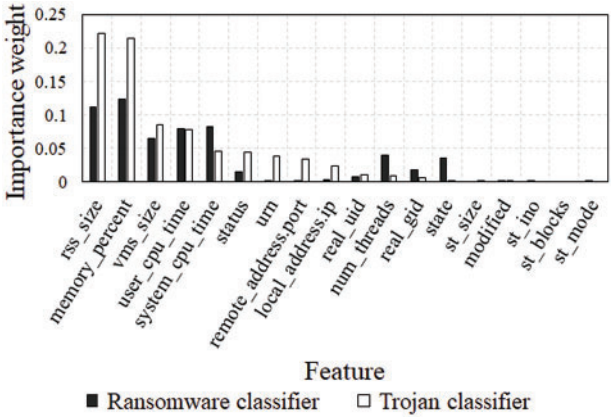

Figure 3: Relative importance weight of each feature in ransomware and trojan classifier

The feature importance weight was derived as the average of 1000 iterations. It can be observed that the feature importance differs depending on the type of attack to be detected. For example,

system_cpu_time, *num_threads*, and *state* are essential features for the ransomware classifier but not for the trojan classifier. In contrast, *status*, *urn*, *remote_address.port*, and *local_address.ip* are essential features for the trojan classifier but not for the ransomware classifier. This is because ransomware often performs intensive encryption, which can significantly increase CPU usage or create multiple threads and may communicate remotely with the server. In contrast, *status*, *urn*, *remote_address.port*, and *local_address.ip* are essential features for the trojan classifier but not for the ransomware classifier. This is because trojan typically exploits specific paths to execute malicious processes in the background and may establish remote connections with an attacker.

Fig. 4 depicts the training accuracy when accumulating the features in order of high importance weights for the feature importance derived for each attack type. In some cases, the slope of the accuracy variation is large when certain features are added. This indicates that the contributions of certain features to the classification accuracy improvement are more considerable than those of others. In contrast, in cases in which the slope is maintained or decreased, no significant change or degradation in accuracy is observed after adding features. This is because the feature importance weight is calculated by considering the contribution of each feature to the statistical performance of a fitted model [33]. Thus, a high feature importance weight does not necessarily correlate with a high contribution to accuracy improvement.

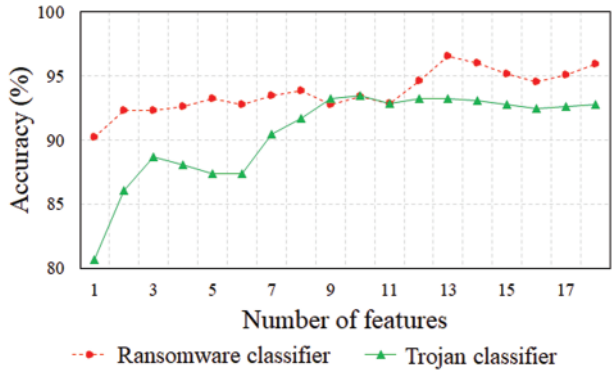

Figure 4: Accuracy based on feature importance by attack type

Let $F_m = \{f_1, f_2, \cdots, f_m\}$ be a set of m feature vectors. V_m is the array that is concatenated vertically from f_1 to f_m, and is used for training the ML model. Acc (V_m) is the accuracy derived by the model trained using V_m. In addition, F_{top} is defined as the set of top-k features with high feature importance weights selected among F_m, whereas F_{noise} and $F_{optimal}$ are defined as the sets of features classified as noise and optimal features among F_{top}, respectively. The accuracy contribution rate (ACR) in Eq. (1) takes the argument f_k, and ACR (f_k) means the accuracy contribution rate when the feature vector f_k is added for the training dataset. If ACR (f_k) is 1.5%p or more, f_k is classified as an optimal feature, and if ACR (f_k) is 0.5%p or less, f_k is classified as a noise feature using Eqs. (2) and (3).

$$ACR\ (f_k) = \text{Acc}\ \ (V_m) - \text{Acc}\ \ (V_{m-1}) \tag{1}$$

$$F_{optimal} = \{f_k \mid f_k \in \arg\ (ACR\ (f_k) \geq 1.5), k \leq m\} \tag{2}$$

$$F_{noise} = \{f_k \mid f_k \in \arg\ (ACR(f_k) \leq 0.5), k \leq m\} \tag{3}$$

Table 3 summarizes F_{noise} and $F_{optimal}$ in the ransomware and trojan classifiers. $F_{optimal}$ differs according to the two attack types, and some features contribute uniquely to the identification of each

attack type. For example, *st_mode* is classified as $F_{optimal}$ for the ransomware classifier and F_{noise} for the trojan classifier. In contrast, *local_address.ip* is classified as $F_{optimal}$ for the trojan classifier and F_{noise} for the ransomware classifier. Therefore, four and five features are used in the ransomware and trojan classifiers, respectively, whereas six features are used in the aggregate, excluding redundant features.

Table 3: Optimal and noise features by attack type

	$F_{optimal}$	F_{noise}
Ransomware classifier	memory_percent, rss_size, st_mode, urn	system_cpu_time, user_cpu_time, num_threads, real_gid, status, local_address.ip, remote_address.port, modified, st_ino
Trojan classifier	rss_size, memory_percent, vms_size, urn, local_address.ip	user_cpu_time, system_cpu_time, status, real_uid, num_threads, real_gid, state, st_size, modified, st_blocks, st_ino, st_mode

3.4 Classification

Fig. 5 depicts the overall classification flow of the proposed model. During the first phase, the ransomware and trojan classifiers operate in parallel to classify activities into the ransomware, trojan, or benign classes. If both classifiers determine the activity to be an attack, the process progresses to the second phase. In the second phase, the probabilities of the ransomware and trojan classifier results are combined to identify the class as ransomware or trojan. Only data points with clear decisions are classified in the first-phase classification, and data points with high uncertainty are eventually classified in the second-phase classification.

1) First-phase classification

During the first phase classification, the binary ransomware and trojan classifiers operate in parallel. If a new attack is identified, a binary classifier can be added to detect the specific attack. The Python threading module was used to implement each binary classifier [34]. Before training, the ransomware classifier assigns non_ransomware to the trojan and benign labels, and the trojan classifier assigns non_trojan to the ransomware and benign labels. The ransomware classifier uses four ransomware-optimized features for training, and the trojan classifier uses five trojan-optimized features for training. Both classifiers were modeled using a DT [35]. The parameters were selected as default parameters provided by the scikit-learn library to create a generalized model that can handle various types of data. We conducted experiments using three popular ML models: DT, SVM, and RF. We found that the classification accuracy of our proposed model using DT was higher than that of the other classification models; thus, we utilized DT for classification.

Table 4 lists the classification results of the first-phase classification. The ransomware classifier returns the values as true for ransomware and false for non-ransomware, and the trojan classifier returns the values as true for trojan and false for non-trojan activity. When the results of the two models differ, the attack is classified as ransomware if the ransomware classifier result is true and as a trojan if the trojan classifier result is true. If the results of both classifiers are false, the activity is classified as benign. If the results of both classifiers are true, this indicates high uncertainty for the data point, so it is difficult to classify it as either ransomware or trojan. If the classifiers cannot make

a clear decision in the first-phase classification, the system proceeds to the second-phase classification, and the uncertainty is reduced.

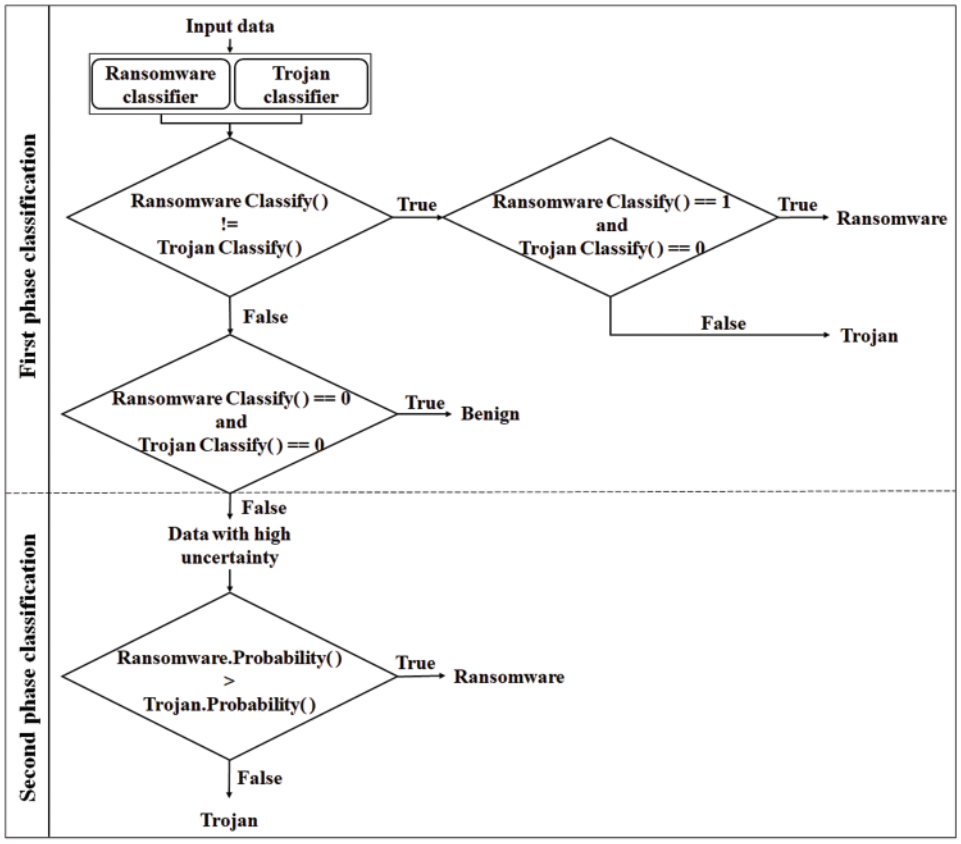

Figure 5: Procedures of proposed model

Table 4: First-phase classification results

Ransomware classifier output	Trojan classifier output	First-phase classification result
False	True	Trojan
True	False	Ransomware
False	False	Benign
True	True	Second-phase classification

2) Second-phase classification

The second-phase classification takes data with high uncertainty as input and finally classifies the activity as trojan or ransomware. If both classifiers determine the activity to be an attack during the first-phase classification, a function is used to determine the attack type based on the prediction probability that is associated with each classifier. Subsequently, the probabilities of the ransomware and trojan classifiers are compared and the classifier result corresponding to the higher probability is used to classify the attack. Algorithm 1 displays the pseudo-code of the proposed model, which

operates through data preprocessing, training model generation, optimal feature set derivation based on feature importance ranking, and attack classification. In step 1, the collected data are loaded and preprocessed, and in step 2, an individual attack classifier is created. In step 3, the optimal feature sets are derived using the ACR function. In step 4, the classification prediction result is determined using two-step classification.

Algorithm 1: Pseudo-code of proposed model

```
Input: Collected dataset
Output: Classification result
Step 1: Data loading and preprocessing
 dataset = data_load(collected_dataset)
 preprocessed_dataset = preprocessor(dataset)
Step 2: Generation of detection models
 ransomware_classifier() = DecisionTreeClassifier()
 trojan_classifier() = DecisionTreeClassifier()
Step 3: Feature importance calculation and selection of optimal feature set
 ransomware_feature_importance = PermutationImportance(ransomware_classifier)
 trojan_feature_importance = PermutationImportance(trojan_classifier)
 ransomware_top_featureset = ransomware_feature_importance[0:N]
 trojan_top_featureset = trojan_feature_importance[0:N]
 for i in range(N)
  if ACR(ransomware_top_featureset[i]) >= 1.5:
   ransomware_optimal_featureset.append(ransomware_top_featureset[i])
  if ACR(trojan_top_featureset[i]) >= 1.5:
   trojan_optimal_featureset.append(trojan_top_featureset[i])
Step 4: Attack classification
predict_ransom = ransomware_classifier(preprocessed_dataset, ransomware_optimal_featureset)
predict_trojan = trojan_classifier(preprocessed_dataset, trojan_optimal_featureset)
for m in range(len(predict_trojan))
  if prediction_ransom[m] == 0 and prediction_trojan[m] == 0:
      result.append(0)                                   ▷ Classification as benign
  elif prediction_ransom[m] == 1 and prediction_trojan[m] == 0:
      result.append(1)                                   ▷ Classification as ransomware
  elif prediction_ransom[m] == 0 and prediction_trojan[m] == 1:
      result.append(2)                                   ▷ Classification as trojan
  else
      z = 1 if prediction_ransom_proba[m] [1] > prediction_trojan_proba[m] [1] else 2
      result.append(z)                  ▷ Second-phase classification that compares probability
```

4 Evaluation and Analysis

In this section, we report the experiments performed to evaluate the performance of the ransomware and trojan attack detection using the proposed model. The proposed model was evaluated using Python 3 with scikit-learn on the Jupyter Notebook in Windows 10. Consequently, 2735 ransomware-related logs, 3470 trojan-related logs, and 4195 benign logs were collected, totaling 10,400 logs. The superiority of the proposed method was demonstrated by comparing the accuracies and

memory usage of the proposed and conventional models in identical experimental environments. A multi-classification model (Conv-All) using all 18 features [19] was selected as a conventional model. Conv-All uses all features without a feature selection process for multi-classification of malware types. A multi-classification top-6 model (Conv-6) that selects the top-6 features with high feature importances [17] was selected as another conventional model. The number of features of Conv-6 was set to six, which is the same as the number of features derived by the feature selection technique of the proposed model.

Fig. 6 shows the accuracies of the conventional and proposed models in the noise environment. The noise was defined as a labeling error in an ML environment and it was implemented to select a label randomly among ransomware, trojan, and benign. The accuracy was compared by increasing the labeling errors, starting from 0% labeling error and incrementally increasing it by 2% up to 50%. All models exhibited decreased accuracy as the noise increased. Except for some sections, the accuracy followed the order of the proposed model, Conv-All model, and Conv-6 model. The accuracy of Conv-6 model was significantly reduced in the noise environment. In the zero-noise environment, the accuracy of the Conv-All model was 85.76% and that of the Conv-6 model was 78.7%. In contrast, the accuracy of the proposed model was 87.1% when using only six features, which was approximately 33.3% of the total of 18 features. In the 50% labeling error environment, the accuracies of the Conv-All and Conv-6 models were 66.29% and 63.12%, respectively, whereas the proposed model exhibited 68.9% accuracy. On average, the proposed model improved the accuracy by approximately 2.95% over the Conv-All model and 11.67% over the Conv-6 model.

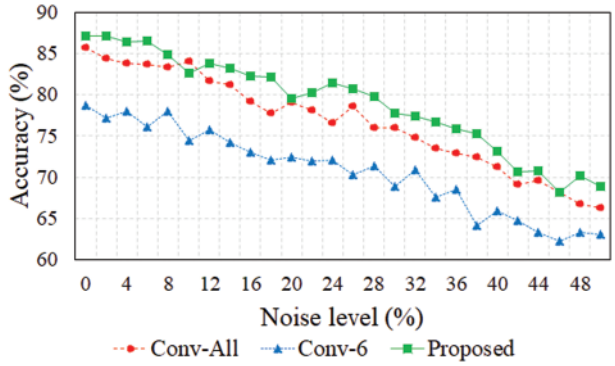

Figure 6: Accuracy based on noise level

The accuracy and memory usage according to the training data size were measured to evaluate the performance of the conventional and proposed models. The dataset ratio was varied with under-sampling from 10% to 90% using stratified sampling [36], which assumes the current dataset ratio as 100% and maintains the data distribution ratio. The accuracy for each training data size is shown in Fig. 7. The overall model training performance gradually improved as the data sampling ratio increased. In particular, the proposed model exhibited higher performance despite using 66.7% fewer features compared to the Conv-All model. In the environment with the smallest data size, the accuracy of the Conv-All and Conv-6 models were 58.65% and 58.64%, and that of proposed model was 65.8%. On average, the proposed model achieved an accuracy improvement of approximately 6.39% over the Conv-All model and 14% over the Conv-6 model.

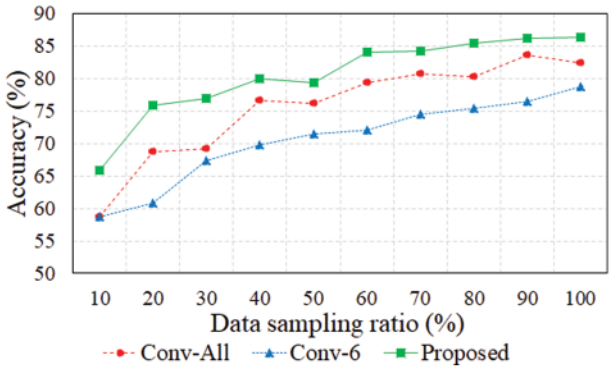

Figure 7: Accuracy based on data sampling ratio

We used memory_profiler [37], which is a process memory utilization monitoring package in Python, to evaluate the memory usage. The memory_profiler package can calculate the total usage and increase or decrease in memory for each code based on the psutil module. In our experiment, we calculated the memory usage of our proposed model from the data preprocessing phase to the classification phase using memory_profiler. Fig. 8 depicts the memory usage according to the data sampling ratio.

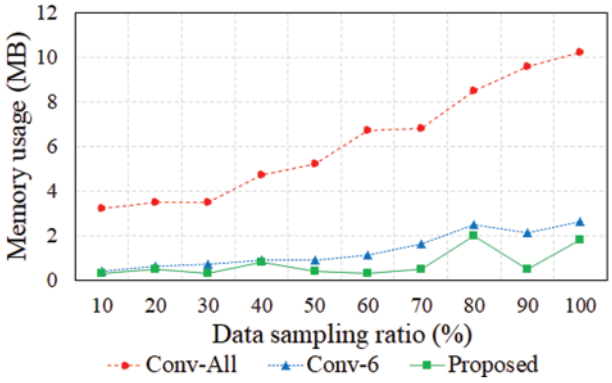

Figure 8: Memory usage based on data sampling ratio

The memory usage is related to the complexity and computational overhead of the algorithm. It can be observed that memory usage increased as the data sampling ratio increased. This is because a more extensive dataset results in more loaded and trained data following preprocessing. The memory usage was the highest for the Conv-All model because all 18 features were used. In contrast, the Conv-6 model [17] exhibited relatively low memory usage because it used only some features, and the proposed model had the lowest memory usage even when the data sampling ratio was high. This is because, in contrast to the Conv-6 model, which is a multi-classification model that uses the same number of features, the proposed model processes data on a parallel processing basis during binary classification. Thus, the proposed model used approximately 88.05% less memory than the conventional Conv-All model and approximately 44.78% less memory than the Conv-6 model on average, making it the most efficient from a memory perspective. Therefore, the proposed model can effectively collect and train data from memory-limited mobile and IoT devices using a small number of features, even in a small training data environment.

5 Conclusion

We have proposed binary classifier-based multi-classification techniques using optimal features for each attack type. The classification performance of conventional intrusion detection methods deteriorates as the diversity of the attack types increases. In addition, signature-based intrusion detection methods can only detect predefined attacks, making it difficult to detect new malware variants effectively. In the case of EDR, many security warnings, including unnecessary data, are generated continuously, thereby wasting memory resources and degrading the intrusion detection performance. The proposed method configures parallel binary classifiers to detect ransomware and trojan attacks. It includes two classification phases to classify each activity as benign, ransomware, or trojan, with the associated probabilities. Each classifier is trained using optimal features that contribute to accuracy improvement based on the feature importance derived from the attack type. We compared the performance of the proposed model to that of a conventional multi-classification model (Conv-All) and a multi-classification top-6 model (Conv-6) using the top six features based on feature importance. On average, the proposed model improved the accuracy by 2.95% over the Conv-All model and 11.67% over the Conv-6 model. In addition, the proposed model improved the memory usage by 88.05% over the Conv-All model and 44.78% over the Conv-6 model. Our proposed model improves the accuracy and memory usage by selecting features specific to each type of malware and deploying binary classifiers in parallel. Thus, this approach not only effectively responds to advanced and diverse new malware, but also allows for the use of memory-intensive DL techniques. However, the proposed model has a limitation in that it has not been tested in a network environment in which an actual attack is performed. In future research, the proposed model will be optimized by configuring the classifier with DL models to operate with low latency and adapt dynamically to new attack environments. It will also be tested in a real-world environment with substantial traffic to demonstrate its effectiveness.

Acknowledgement: Not applicable.

Funding Statement: This work was partially supported by MOTIE under Training Industrial Security Specialist for High-Tech Industry (RS-2024-00415520) supervised by the Korea Institute for Advancement of Technology (KIAT), and by MSIT under the ICT Challenge and Advanced Network of HRD (ICAN) Program (No. IITP-2022-RS-2022-00156310) supervised by the Institute of Information & Communication Technology Planning & Evaluation (IITP).

Author Contributions: The authors confirm their contributions to the paper as follows: Ye-Seul Kil: Conceptualization, Methodology, Software, Validation, Investigation, Resources, Data Curation, Visualization, Writing—Original Draft. Yu-Ran Jeon: Conceptualization, Methodology, Software, Investigation, Resources, Data Curation, Writing—Original Draft. Sun-Jin Lee: Methodology, Software, Investigation, Resources, Data Curation, Writing—Original Draft. Il-Gu Lee: Conceptualization, Validation, Writing—Review and Editing, Supervision, Project Administration, Funding Acquisition. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data underlying the results presented in this paper are not publicly available at this time but may be obtained from the authors upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Pranggono B, Arabo A. COVID-19 pandemic cybersecurity issues. Internet Technol Lett. 2021;4(2):e247. doi:10.1002/itl2.247.
2. Aslan Ö.A, Samet R. A comprehensive review on malware detection approaches. IEEE Access. 2020;8:6249–71. doi:10.1109/ACCESS.2019.2963724.
3. Al-Asli M, Ghaleb TA. Review of signature-based techniques in antivirus products. In: 2019 International Conference on Computer and Information Sciences (ICCIS), 2019; Sakaka, Saudi Arabia; p. 1–6. doi:10.1109/ICCISci.2019.8716381.
4. Scott J. Signature based malware detection is dead. In: Institute for Critical Infrastructure Technology, The 2017 Critical Infrastructure Forum; 2017; Washington, DC, USA.
5. Sathyanarayan VS, Kohli P, Bruhadeshwar B. Signature generation and detection of malware families. In: ACIS 2008: Information Security and Privacy, Wollongong, Australia, 2008 July 7–9, Berlin, Heidelberg, Springer; 2008; p. 336–49. doi:10.1007/978-3-540-70500-0_25.
6. Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic malware analysis in the modern era—a state of the art survey. ACM Comput Surv. 2020;52(6):1–48. doi:10.1145/3365001.
7. George AS, George ASH, Baskar T, Pandey D. XDR: the evolution of endpoint security solutions-superior extensibility and analytics to satisfy the organizational needs of the future. Int J Adv Res Sci. 2021;8:493–501. doi:10.48175/568.
8. Park SH, Yun SW, Jeon SE, Park NE, Shim HY, Lee YR, et al. Performance evaluation of open-source endpoint detection and response combining Google Rapid Response and Osquery for threat detection. IEEE Access. 2022;10(4):20259–69. doi:10.1109/ACCESS.2022.3152574.
9. Lee SJ, Shim HY, Lee YR, Park TR, Park SH, Lee IG. Study on systematic ransomware detection techniques. In: 24th International Conference on Advanced Communication Technology (ICACT), 2022; Pyeongchang, Republic of Korea; p. 297–301. doi:10.23919/ICACT53585.2022.9728909.
10. Jeon SE, Lee SJ, Lee EY, Lee YJ, Ryu JH, Moon JH, et al. An effective threat detection framework for advanced persistent cyberattacks. Comput Mater Contin. 2023;75(2):4231–53. doi:10.32604/cmc.2023.034287.
11. Park NE, Lee YR, Joo S, Kim SY, Kim SH, Park JY, et al. Performance evaluation of a fast and efficient intrusion detection framework for advanced persistent threat-based cyberattacks. Comput Electr Eng. 2023;105(4):108548. doi:10.1016/j.compeleceng.2022.108548.
12. Gezer A, Warner G, Wilson C, Shrestha P. A flow-based approach for Trickbot banking trojan detection. Comput Secur. 2019;84(2):179–92. doi:10.1016/j.cose.2019.03.013.
13. Nagaraju V, Raaza A, Rajendran V, Ravikumar D. Deep learning binary fruit fly algorithm for identifying SYN flood attack from TCP/IP. Mater Today. 2023;80(3):3086–91. doi:10.1016/j.matpr.2021.07.171.
14. Irshad A, Maurya R, Dutta MK, Burget R, Uher V. Feature optimization for run time analysis of malware in Windows operating system using machine learning approach. In: 2019 42nd International Conference on Telecommunications and Signal Processing (TSP), 2019; Budapest, Hungary; p. 255–60. doi:10.1109/TSP.2019.8768808.
15. Sun H, Xu G, Wu Z, Quan R. Android malware detection based on feature selection and weight measurement. Intell Autom Soft Comput. 2022;33(1):585–600. doi:10.32604/iasc.2022.023874.
16. Pei X, Deng X, Tian S, Zhang L, Xue K. A knowledge transfer-based semi-supervised federated learning for IoT malware detection. IEEE Trans Dependable Secure Comput. 2022 May 10;20(3):2127–43. doi:10.1109/TDSC.2022.3173664.
17. Abbasi MS, Al-Sahaf H, Mansoori M, Welch I. Behavior-based ransomware classification: a particle swarm optimization wrapper-based approach for feature selection. Appl Soft Comput. 2022;121(9):108744. doi:10.1016/j.asoc.2022.108744.

18. Tuan TA, Long HV, Son LH, Kumar R, Priyadarshini I, Son NTK. Performance evaluation of Botnet DDoS attack detection using machine learning. Evol Intell. 2019;13(2):283–94. doi:10.1007/s12065-019-00310-w.
19. Saheed YK, Arowolo MO. Efficient cyber attack detection on the Internet of Medical Things-smart environment based on deep recurrent neural network and machine learning algorithms. IEEE Access. 2021;9:161546–54. doi:10.1109/ACCESS.2021.3128837.
20. Sethi K, Kumar R, Sethi L, Bera P, Patra PK. A novel machine learning based malware detection and classification framework. In: 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2019; Oxford, UK; p. 1–4. doi:10.1109/CyberSecPODS.2019.8885196.
21. Piskozub M, de Gaspari F, Barr-Smith F, Mancini L, Martinovic I. MalPhase: fine-grained malware detection using network flow data. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, 2021; New York, NY, USA; p. 774–86. doi:10.1145/3433210.3453101.
22. Zhang X, Fan M, Wang D, Zhou P, Tao D. Top-k feature selection framework using robust 0–1 integer programming. IEEE Trans Neur Netw Learn Syst. 2020 Jul 31;32(7):3005–19. doi:10.1109/TNNLS.2020.3009209.
23. Hnamte V, Hussain J. DCNNBiLSTM: an efficient hybrid deep learning-based intrusion detection system. Telemat Inform Rep. 2023;10:100053 doi:10.1016/j.teler.2023.100053.
24. Awajan A. A novel deep learning-based intrusion detection system for IoT networks. Computers. 2023;12(2):34. doi:10.3390/computers12020034.
25. Zhang X, Wang T, Wang J, Tang G, Zhao L. Pyramid channel-based feature attention network for image dehazing. Comput Vis Image Underst. 2020 Aug 1;197(9):103003. doi:10.1016/j.cviu.2020.103003.
26. Hu X, Zhu C, Cheng G, Li R, Wu H, Gong J. A deep subdomain adaptation network with attention mechanism for malware variant traffic identification at an IoT edge gateway. IEEE Internet Things J. 2022 Mar 21;10(5):3814–26. doi:10.1109/JIOT.2022.3160755.
27. GRR Team. What is GRR? 2017. Available from: https://grr-doc.readthedocs.io/en/latest/what-is-grr.html. [Accessed 2023].
28. Almeida F, Imran M, Raik J, Pagliarini S. Ransomware attack as hardware trojan: a feasibility and demonstration study. IEEE Access. 2022 Apr 20;10:44827–39.
29. OffSec. MSFVENOM. 2015. Available from: https://www.offensive-security.com/metasploit-unleashed/Msfvenom/. [Accessed 2023].
30. Endermanch. MalwareDatabase. 2020. Available from: https://github.com/Endermanch/MalwareDatabase/tree/master/trojans. [Accessed 2023].
31. JoelGMSec. PSRansom. 2022. Available from: https://github.com/JoelGMSec/PSRansom. [Accessed 2023].
32. GRR Team. GRR. 2017. Available from: https://grr-doc.readthedocs.io/en/v3.2.1/index.html. [Accessed 2023].
33. Altmann A, Toloşi L, Sander O, Lengauer T. Permutation importance: a corrected feature importance measure. Bioinformatics. 2010;26(10):1340–7. doi:10.1093/bioinformatics/btq134.
34. Threading. Available from: https://docs.python.org/3/library/threading.html. [Accessed 2023].
35. Decision tree classifier. 2007. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Accessed 2023].
36. Parsons VL. Stratified Sampling. In: Balakrishnan N, Colton T, Everitt B, Piegorsch W, Ruggeri F, Teugels JL, editors. Variance reduction. Statistics Reference Online: Wiley StatsRef. John Wiley & Sons; 2017.
37. Fabian P, Philippe G. memory_profiler. 2011. Available from: https://github.com/pythonprofilers/memory_profiler. [Accessed 2024].