Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

The Improvement of Android Malware Family Detection through System Call Feature Analysis and Machine Learning

Rajif Agung Yunmar

Fakultas Teknologi Informasi, Program Studi Teknik Informatika, Institut Teknologi Sumatera, Lampung Selatan
Jl. Terusan Ryacudu, Way Huwi, Kec. Jati Agung, Kabupaten Lampung Selatan, Lampung, Indonesia
Email: rajif@if.itera.ac.id
Correspondence Author Email: rajif@if.itera.ac.id
Submitted: **10/10/2024;** Accepted: **20/10/2024;** Published: **21/10/2024**

Abstract−Malware poses a significant threat to cybersecurity, particularly for Android users. Each type of malware is categorized into distinct categories and families, each exhibiting unique malicious capabilities. Accurately identifying these categories and families is crucial for developing effective prevention and mitigation strategies, allowing for the control of threats before they worsen. Throughout the years, numerous techniques have been proposed for detecting malware families, with system calls emerging as a vital feature. Collected through dynamic analysis, system calls offer in-depth insights into the activities executed by malware, making them a powerful classification tool. This study aims to enhance the detection of Android malware families and categories by analyzing system calls with feature selection method. Using the Gain Ratio algorithm, significant system calls are identified to improve detection accuracy and reduce the complexity of the feature set. The study assesses machine learning algorithms, particularly Random Forest, J48, Naïve Bayes, and Decision Table. The findings show that Random Forest consistently outperforms other algorithms, achieving an accuracy of 88.01% for malware family detection and 89.65% for category detection, with high precision and recall across most metrics. The application of the Gain Ratio feature selection method led to a 68.83% feature reduction and improved model-building speed by 50.26%. This integration of feature selection and machine learning provides a more effective approach to detecting malware families and categories, thus contributing to enhanced Android security.

Keywords: Malware Family Detection; Dynamic Analysis; System Call; Feature Selection; Machine Learning.

1. INTRODUCTION

Malware refers to software intentionally designed to cause harm, disrupt, or gain unauthorized access. According to a report by GData, a thousand new malware variants surface every day [1]. These new variants continually evolve, making them increasingly challenging to detect using conventional security systems, thus posing a greater challenge to identify and prevent. The repercussions extend beyond data security, resulting in substantial financial losses. By 2031, the projected financial impact caused by ransomware alone is estimated to reach $265 billion [2].

Malware can be categorized into families to facilitate the processes of identification, analysis, and the development of mitigation and prevention strategies. This categorization is predicated on behavioral patterns, distribution mechanisms, and the attack objectives employed by the malware. Ransomware, for instance, restricts access to the victim's data through encryption and demands a ransom for its release, while spyware is designed to surreptitiously monitor users to obtain sensitive information such as login credentials, personal data, or financial details. The identification of malware families constitutes a critical step in detecting and mitigating attacks, as each family exhibits distinct attack patterns and objectives.

Various strategies can be utilized to minimize the impact of malware, with detection being a key component. Timely and efficient detection is crucial in identifying malware at an early stage to prevent widespread damage. A highly effective method is heuristic-based detection, which involves analyzing suspicious or malicious patterns in an app and identifying malware through rule-based systems or machine learning algorithms [3]. Unlike traditional signature-based detection, which only identifies known malware, heuristic-based detection focuses on detecting suspicious behaviors or anomalies that could indicate new or modified malware variants.

The detection process using heuristic methods heavily depends on the features obtained during analysis. These features represent the behavior or attributes of a program that can differentiate between normal and malicious applications. Such features may comprise file access patterns, abnormal network behavior, suspicious resource usage, or unauthorized attempts to alter the system. One of the main sources of features for heuristic detection is dynamic analysis. Dynamic analysis is a method of analyzing malware by executing it in a controlled environment, which allows for direct observation of its activities [4]. This approach differs from static analysis, which only involves examining the malware's source code without executing it [5]. Dynamic analysis enables researchers to observe the actual behavior of the malware as it interacts with the system, such as attempting to access system files, modify the registry, or communicate with external servers.

The advantage of dynamic analysis lies in its ability to detect malware that employs obfuscation techniques [6], which involve concealing code or behavioral patterns to evade detection by static or signature-based methods. Numerous contemporary malware variants utilize obfuscation to disguise themselves as legitimate software or to conceal indicators of their malicious behavior until a specific point in time. However, dynamic analysis reveals the malware's malicious actions during execution, irrespective of the obfuscation techniques employed. This capability renders dynamic analysis an essential tool in identifying and analyzing sophisticated threats that might otherwise elude detection by other methods.

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

Previous studies have used features derived from dynamic analysis for detecting malware families. For example, Abuthawabeh and Mahmoud [7], Fallah and Bidgoly [8], and Gohari et al. [9] used network traffic features to identify communication patterns and data transfer behavior related to malware families. Additionally, Taheri et al. [10] not only employed network traffic features but also integrated API calls reflecting system-level interactions and functions invoked by malware for family detection. However, dynamic features may not cover all malicious activities, as network activity represents only a part of malicious behavior and cannot capture all forms of malicious activity.

System calls are a key feature that can be identified through dynamic analysis and are the most commonly used features for malware detection using dynamic analysis methods [11]. These calls act as the interface between the executing application and the kernel [12], managing interactions between the hardware and software in the operating system. During application execution, any activity that requires interaction with the system is carried out through system calls. This includes tasks such as reading and writing files, sending data over a network, or allocating memory. By monitoring these system calls, researchers can gain valuable insights into the behavior of malware, allowing for the detection of potentially malicious activities that may not be obvious through other analysis methods. This focus on system calls improves the ability to detect advanced malware that depends on stealthy operations to bypass traditional security measures.

System calls can be classified into various functional groups, including File Operations, Network Communication, and Memory Management, among others [13]. As system calls capture all the actions performed by an application during execution, logs of these calls offer valuable insights into the application's behavior. This underscores the significance of system calls in malware detection. Different types of malware generate distinct patterns of system calls that correspond to their malicious activities. For example, ransomware typically exhibits system call patterns associated with file encryption, while spyware may demonstrate increased activity involving hardware access and unusual network connections. Analyzing these patterns enables researchers to effectively identify and categorize malware, thereby bolstering overall cybersecurity efforts.

Moreover, system calls are not only useful for identifying the presence of malware, but they can also be employed to classify malware according to its family. Each type of malware tends to display unique patterns of behavior. System calls also have a distinct advantage when dealing with obfuscation techniques commonly used by malware to hide their code [14], [15]. While the code of the malware may change to avoid static detection, the malicious behavior patterns reflected in system calls remain difficult to significantly alter. As a result, system calls can function as a robust feature for more accurately detecting and categorizing malware, even when the malware uses disguising techniques or modifies its code. This resilience enhances the effectiveness of dynamic analysis in the ongoing fight against sophisticated malware threats.

Numerous prior studies have concentrated on the identification of malware families through the analysis of system calls. For example, Ding et al. [16] utilized system call graphs for malware family detection. However, their research incorporated all available system calls without considering that some might not be relevant to detection. Similarly, Manzil and Naik [17] as well as El Fiky et al. [18] employed feature selection to reduce the feature set in their malware detection using system calls, but did not provide insights into the significance of the improvements in detection performance or the reduction in time to detect malware. Meanwhile, Shakya and Dave [19] used system calls for malware detection but did not analyze the most influential features using a heuristic approach, relying instead on the occurrence frequency of system calls in benign and non-benign samples, which might introduce bias. Similarly, Malik and Khatter [20] investigated malware families without employing heuristic methods, potentially leading to biased outcomes.

The objective of this study is to investigate methods for identifying malware families and enhancing their detection by focusing on improving the accuracy and efficiency of categorizing different types of malware based on their activity patterns. First, the study will pinpoint significant system calls and propose the most relevant calls for the detection process, categorizing them based on specific functions such as file operations, network communication, and memory management. This approach will provide a clearer understanding of the distinct behaviors exhibited by malware. Second, the study aims to improve the accuracy of malware family detection through automated feature selection methods, addressing the limitations of previous manual approaches that are often prone to bias. By implementing objective feature selection, the goal is to enhance the model's ability to effectively distinguish between different malware families. Third, the research will employ machine learning algorithms to classify malware based on the extracted and selected system call data, with the aim of achieving higher classification accuracy and adaptability to new malware variants.

2. RESEARCH METHODOLOGY

2.1 Research Stages

Malware detection is a complex and multidimensional process that encompasses a series of interrelated stages aimed at achieving effective results. The process commences with the initial setup, wherein the analysis environment is configured to ensure optimal conditions for detection. The subsequent step entails the preparation

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

of the dataset, involving the collection and organization of data containing samples of both malware and non-malware for the purpose of training and testing the model. This meticulous organization of data is paramount as it significantly impacts the model's capacity to accurately differentiate between malicious and benign software, thereby bolstering the overall effectiveness of the detection process.

Afterward, the feature extraction stage is carried out to capture crucial elements of the malware, such as significant behaviors and activities, which will form the basis for further analysis. Subsequently, feature selection is conducted to identify the most relevant and informative features, thereby improving the efficiency and effectiveness of detection by eliminating unnecessary noise. Finally, the classification stage utilizes machine learning algorithms to categorize the malware based on the identified patterns. This research methodology can be visualized in Figure 1.

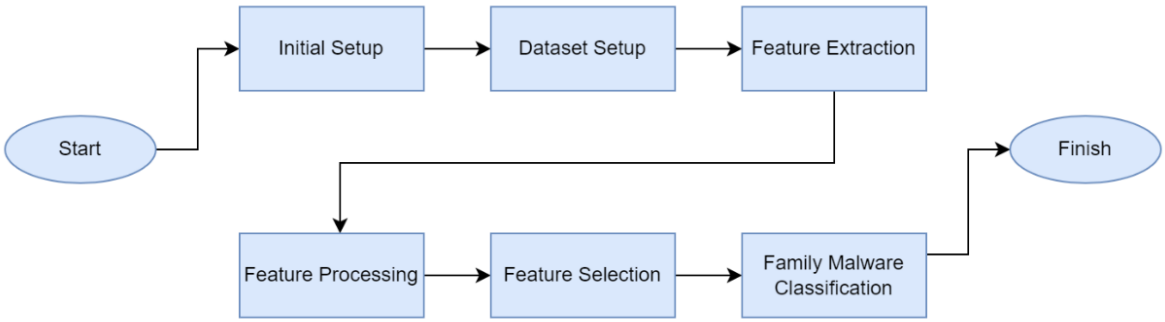

Figure 1. Overview of the research methodology process.

2.2 Initial Setup

The initial setup stage is a critical step in malware analysis, where the experimental environment is carefully prepared to ensure maximum security and control. This phase is essential to ensure that the analysis process is conducted safely, preventing the infection of the primary system or other network infrastructure. By creating a controlled environment, researchers can effectively isolate the malware under investigation, allowing for a comprehensive examination of its behavior and characteristics without risking unintended consequences. This meticulous preparation is crucial for maintaining the integrity of the analytical process and the broader cybersecurity framework, ultimately leading to more accurate and reliable findings in malware research.

In this research, a physical device, specifically a Samsung J330 running Android operating system version API 24, will be used. The device will be connected to the internet via Wi-Fi during the malware execution process. To simulate real-world conditions in which malware may exploit information, this device has been populated with contact data, call logs, and SMS messages. This configuration allows for a more representative experimental environment, enabling a comprehensive analysis of malware behavior in everyday use. By doing so, the findings not only gain ecological validity but also offer valuable insights into how malware operates within a typical user environment, thus informing more effective detection and mitigation strategies.

2.3 Datasets

The data used in this study is sourced from CIC-AndMal2017 [21] and MalDroid-2020 [22], [23], focusing specifically on malware with a minimum API level of 21. This choice of API level is based on the observation that the majority of currently used Android devices run on versions with API level 21 or higher [24], corresponding to Android version 5.0. From this dataset, we collected a total of 1,585 malware samples, covering six distinct types of malware categorized into 17 families. This classification enables a more specific and structured analysis, providing a comprehensive understanding of activity patterns and facilitating the detection of each malware family present in the dataset. Table 1 presents the distribution of malware within the dataset, offering a visual representation of the diverse malware samples and their classifications. This organized approach to data collection not only enhances the rigor of the analysis but also contributes to the overall reliability of the research findings.

Table 1. Distribution of the malware dataset across various types and families.

Type	Family	Num. of Malware
Adware	Dowgin	120
Adware	Ewind	45
Adware	Mobidash	48
Ransomware	Congur	38
Ransomware	Svpeng	39
Riskware	Dnotua.xww	129
Riskware	PornVideo	37
Riskware	Yoga	82

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

Type	Family	Num. of Malware
SMS Malware	Fakeinst	153
SMS Malware	Jifake	39
SMS Malware	Opfake.a	247
Spyware	Agent	34
Spyware	Recal	25
Spyware	SmForw	203
Trojan	Boogr	41
Trojan	Fakeapp	190
Trojan	Piom	115

2.4 Feature Extraction through Dynamic Analysis

In this section, we will extract system calls using dynamic analysis. The process involves running the malware on the device for a specific duration to observe its malicious activities. These activities manifest through the system calls made by the malware during execution. The feature extraction process, depicted in Figure 2, can be outlined as follows:

1. Each malware sample will be installed on a device that has been configured according to the specifications outlined in the Initial Setup section.
2. Once the malware is installed on the Android device, we will execute it, allowing it to run for ten seconds to ensure the application fully launches.
3. After the malware application has successfully opened, we will inject 2,000 UI events into the malware using a built-in ADB tool called Monkey, which simulates user interactions [25]. These UI events will replicate the interaction between the user and the malware application, prompting the malware to exhibit its malicious activities, making it easier to detect these activities when they occur.
4. Finally, after being executed for a total of four minutes on the Android device, the malware application will be terminated, and a log will be generated. The system call log will reflect the malicious activities of the malware application during its execution.

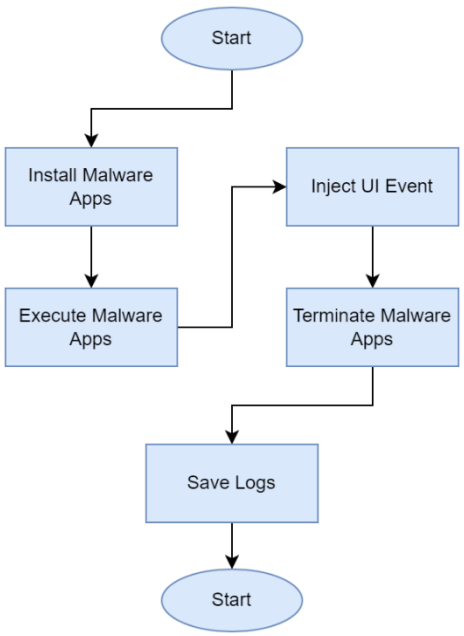

Figure 2. The detailed process of system call extraction through dynamic analysis.

2.5 Feature Processing

The process of dynamic analysis generates raw system call logs, which need to be formatted in a way that is suitable for machine learning algorithms. These logs record all system calls made while the application is running on an Android device. However, in their original state, this data is too diverse and unorganized to be directly utilized in a machine learning model. Therefore, a transformation process, such as preprocessing, is necessary. During this stage, the logs are scrutinized and converted into extractable features. These features represent specific system call activities, offering a clearer understanding of the application's behavior during its execution.

In the process of transforming system call logs into feature vectors for machine learning, several crucial steps are involved. Firstly, a set of 324 system calls [26], denoted as $SC = \{sc_1, sc_2, \dots, sc_{324}\}$, is defined. Each log entry, which records a system call invocation during application execution, is represented as $L = \{l_1, l_2, \dots, l_n\}$,

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

where n is the total number of log entries. For each system call sc_i, a count vector $V = \{v_1, v_2, \dots, v_n\}$ is created, where v_i represents the occurrence count of sc_i in the log, initially set to zero.

The next step involves reading each log entry l_j, and every time system call sc_i is invoked, the corresponding value v_i is incremented by one. After processing the entire log, the result is a feature vector V containing the frequency of occurrence for each system call. This feature vector is then ready to be used as input for machine learning models.

2.6 Feature Selection

This section is focuses on identifying the system calls that play the most significant role in classifying malware, thereby enhancing the accuracy and efficiency of detection models. Not all system calls within the Android API 24 hold equal importance in detecting malware. Certain system calls have a greater impact on detection results than others. The Gain Ratio method is used to pinpoint the most influential system calls. This method assesses the contribution of each feature, in this case, system calls, to the classification process based on the information derived from that feature. The Gain Ratio measures the ratio between a feature's information gain and its entropy, reducing bias towards features with numerous distinct values.

2.7 Detection of Malware Families

In this section, we will delve into the process of detecting malware, which involves the application of various machine learning algorithms to identify and categorize different malware families. This detection process commences after the feature selection stage with the goal of enhancing the accuracy of malware classification and optimizing the performance of the detection model. The classification and detection of malware families take place subsequent to the feature selection process in order to improve detection accuracy. In this study, several algorithms are employed, each with its own distinct advantages:

1. Decision Table: This algorithm yields a simple, easy-to-understand interpretation by generating a decision table. Its strength lies in effectively handling small datasets while producing models that are highly interpretable by humans.
2. J48: Serving as an implementation of the C4.5 decision tree algorithm, J48 is effective in handling data with missing values and can generate comprehensive decision trees. It is also versatile, capable of handling both numerical and categorical attributes.
3. Naive Bayes: Renowned for its speed and efficiency when working with large datasets, Naive Bayes assumes independence among features. Despite this simplification, it can still deliver competitive results even with a relatively simple model structure.
4. Random Forest: This algorithm excels in both accuracy and generalization. By constructing numerous decision trees randomly and aggregating their results through a voting mechanism, Random Forest reduces the risk of overfitting and consistently performs well on complex datasets.

The performance of each machine learning algorithm in detecting malware will be systematically evaluated using a variety of key metrics. These metrics collectively provide a comprehensive understanding of the model's ability to identify malware while minimizing errors. They serve different but complementary purposes in assessing the effectiveness of the algorithms:

1. Accuracy: This metric represents the overall proportion of correct predictions made by the model across all instances. It provides a general measure of how reliably the model performs in identifying both malware and non-malware.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

2. Precision: This quantifies the accuracy of the model's positive predictions, indicating the proportion of true malware predictions. A high precision score signifies that the model effectively minimizes false positives by accurately classifying malware.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

3. Recall (True Positive Rate): This evaluates the model's ability to detect all actual malware instances. A higher recall ensures that the model captures a larger portion of true malware, reducing the likelihood of false negatives.

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

4. F-Measure (F1-Score): This is the harmonic mean of precision and recall, offering a balanced view of the model's performance. A high F1-Score reflects the model's ability to maintain a balance between detecting malware and avoiding incorrect positive classifications.

$$F1 = 2x \frac{Precision\ x\ Recall}{Precision+Recall} \quad (4)$$

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

3. RESULT AND DISCUSSION

In the Results and Discussion section, we will delve into the findings of applying the research methods for detecting malware families. This section will be segmented into three main parts: feature selection results, their influence on detection, and detection results. The discussion will assess the detection outcomes based on the performance of each machine learning algorithm, as well as the impact of feature selection on the classification process. We will employ WEKA version 3.8.6 software as a tool to aid in the implementation of the feature selection and detection algorithms.

3.1 Feature Selection Result

In this phase, we will outline how feature selection algorithms, such as Gain Ratio, are used to decrease the number of features. Table 2 presents a comparison of the number of features before and after applying the Gain Ratio algorithm, specifically highlighting those with a rank value greater than 0. By using this method for feature selection, the number of system calls employed in the detection process is significantly reduced, achieving an overall reduction of 68.83%. This indicates that more than half of the initially employed features can be eliminated. Table 2 contains detailed information about the reduction achieved by the Gain Ratio feature selection algorithm.

Table 2. Feature selection reduces the number of features after the application of the Gain Ratio algorithm.

Name	Number of Features	Reduction
Original Features	324	-
After Gain Ratio implementation	101	68.83%

Error! Reference source not found. outlines the top ten system calls that are crucial for detecting malware f amilies based on feature analysis. Each system call listed significantly contributes to the identification of malware, as they are commonly utilized or manipulated by different types of malicious software. System calls are grouped according to their functions, each serving a specific utility:

1. File Operations: This category pertains to system calls related to file manipulation, playing a vital role in identifying malware involved in file manipulation within the system. Malware often engages in activities such as opening, reading, writing, or deleting files for purposes such as data theft or file corruption.
2. Network Communication: System calls in this category are used for network activities. Malware involved in network-based attacks frequently uses these calls to communicate with command-and-control (C2) servers or to transfer stolen data to external entities.
3. Process and Thread Management: This group manages the execution of processes and threads. Malware often creates new processes or modifies existing ones to execute payloads or conceal its activities.
4. Memory Management: This category identifies malware that manipulates memory allocation. Ransomware and worms frequently use techniques such as obfuscation or memory corruption to hide their malicious code or exploit system vulnerabilities.
5. Inter-Process Communication (IPC): IPC system calls are used for communication between processes. Malware often exploits these mechanisms to communicate with other processes or to coordinate malicious activities distributed across various parts of the system.

3.2 The Impact of Feature Selection into Detection Result

In this section, we will explore the impact of the Gain Ratio feature selection algorithm on detection results. The machine learning techniques used in this study employ 5-fold cross-validation to evaluate performance. Table 3 illustrates the detection results before and after applying feature selection. It is evident from Table 3 that the Random Forest algorithm exhibits the highest accuracy in detecting malware families, achieving 88.01%, while the Naive Bayes algorithm shows the lowest accuracy at 38.54%. The Gain Ratio algorithm not only improves the accuracy of certain models but also reduces the time required to build them. For example, Naive Bayes experienced a significant reduction in model-building time by 70%, decreasing from 0.10 seconds to just 0.03 seconds, while Random Forest reduced its model-building time by 50.26%, from 1.93 seconds to 0.96 seconds.

Table 3. The performance of machine learning algorithms before and after the implementation of the Gain Ratio feature selection algorithm.

Machine Learning Algorithm	Accuracy (%)			Time to Build (seconds)			
	With Original Feature	After Gain Ratio	Improvement	With Original Feature	After Gain Ratio	Improvement	%
Decision Table	64.35	63.91	-0.44	10.88	4.26	6.62	60.76
J48	82.83	82.46	-0.37	0.74	0.24	0.50	67.57
Naive Bayes	38.29	38.54	0.25	0.10	0.03	0.07	70.00
Random Forest	87.57	88.01	0.44	1.93	0.96	0.97	50.26

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

However, not all algorithms benefit from increased accuracy. The decrease in accuracy observed in certain algorithms, such as Decision Table and J48, after applying the Gain Ratio feature selection algorithm may be partially due to the removal of specific features that could still contribute, albeit marginally, to the model's performance. Additionally, the Gain Ratio algorithm evaluates features individually, potentially overlooking important interactions between features, which could lead to a decline in accuracy when these interactive features are removed.

The Gain Ratio algorithm has been shown to reduce detection time while improving the performance of machine learning detection methods. In the following sections, we will delve into the detailed performance of machine learning algorithms in detecting malware families using the input features selected through the Gain Ratio method.

3.3 Detection Result and Discussion

In this section, we will analyze the results of detecting malware families and categories using machine learning algorithms. The input data has been processed using the Gain Ratio feature selection algorithm. We will evaluate the performance of the malware family detection models using various machine learning algorithms based on predefined evaluation metrics such as True Positive Rate, Precision, Recall, and F1-Score. The machine learning algorithms used in this study comprise Decision Table, J48, Naive Bayes, and Random Forest.

The results in Table 4 illustrate the performance of various machine learning algorithms in detecting malware families, along with their corresponding evaluation metrics. It is evident from the table that Random Forest demonstrates the highest performance, achieving an accuracy of 88.01%. Following Random Forest, J48, Decision Table, and Naive Bayes achieved accuracies of 82.46%, 63.91%, and 38.54% respectively. Based on precision, recall, F-Measure, and accuracy metrics, Random Forest emerges as the most effective algorithm for malware detection. These findings indicate that Random Forest is adept at identifying malware, successfully detecting the majority of existing malware while maintaining a favorable balance between precision and recall.

Table 4. The performance of machine learning in detecting malware families using various evaluation metrics.

Name	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
Decision Table	65.9	63.9	62.3	63.91
J48	82.5	82.5	82.4	82.46
Naive Bayes	54.1	38.5	37.9	38.54
Random Forest	88.2	88.0	87.7	88.01

The Random Forest algorithm demonstrates the highest precision at 88.2%, indicating its superior ability to accurately identify malware compared to other algorithms. On the other hand, Naive Bayes exhibits the lowest precision at 54.1%, suggesting that a significant number of positive results are, in fact, false positives. Both Random Forest and J48 achieve high recall rates of 88% and 82.5%, respectively, signifying their effectiveness in detecting the majority of existing malware. In contrast, Naive Bayes has the lowest recall at 38.5%, indicating that a considerable amount of malware remains undetected. Furthermore, Random Forest also boasts the highest F-Measure at 87.7%, reflecting a strong balance between precision and recall. Conversely, Naive Bayes has the lowest F-Measure at 37.9%, indicating overall poor performance.

In addition to detecting malware families, we also evaluate the efficacy of machine learning algorithms in identifying malware categories. As presented in Table 1, the dataset comprises six malware categories: Adware, Ransomware, Riskware, SMS Malware, Spyware, and Trojan. The input data utilized for this analysis has undergone feature selection employing the Gain Ratio algorithm. The machine learning algorithms implemented in this study include Decision Table, J48, Naive Bayes, and Random Forest.

In Table **5**, it is evident that the Random Forest algorithm consistently shows the best performance across all metrics, achieving an accuracy of 89.65%. This suggests that Random Forest is highly effective in classifying malware categories, achieving high accuracy with minimal false positives and false negatives. The performance in detection is closely followed by J48, with an accuracy of 82.08%. Decision Table produces satisfactory results but remains relatively inferior, with an accuracy of 73.81%, while Naive Bayes shows inadequate performance and may need alternative approaches or feature adjustments to enhance its effectiveness.

Table 5. The performance of machine learning in detecting malware categories.

Name	Precision (%)	Recall (%)	F-Measure (%)	Accuracy (%)
Decision Table	74.9	73.8	72.8	73.81
J48	82.1	82.1	82.1	82.08
Naive Bayes	54	43.1	41.3	43.09
Random Forest	89.7	89.7	89.6	89.65

Meanwhile, we can further analyze Table **5** by detailing the performance metrics of precision, recall, and F-measure for each malware category. Precision, which denotes the ratio of true positive predictions to all positive predictions, is a crucial metric. A higher precision indicates fewer false positives (i.e., incorrectly identifying

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

benign instances as malware). The Random Forest algorithm consistently achieves the highest precision across almost all malware categories, showcasing its superior ability to identify true positives while minimizing false positives. Notably, the precision of the Random Forest algorithm reaches an impressive 94.5% for the SMS Malware category, marking the top performance among all evaluated algorithms. Conversely, Naive Bayes demonstrates extremely low precision (6.5%) for the Ransomware category, highlighting significant difficulties in detecting this type of malware. The detailed performance of malware detection based on precision metrics is depicted in Figure 3.

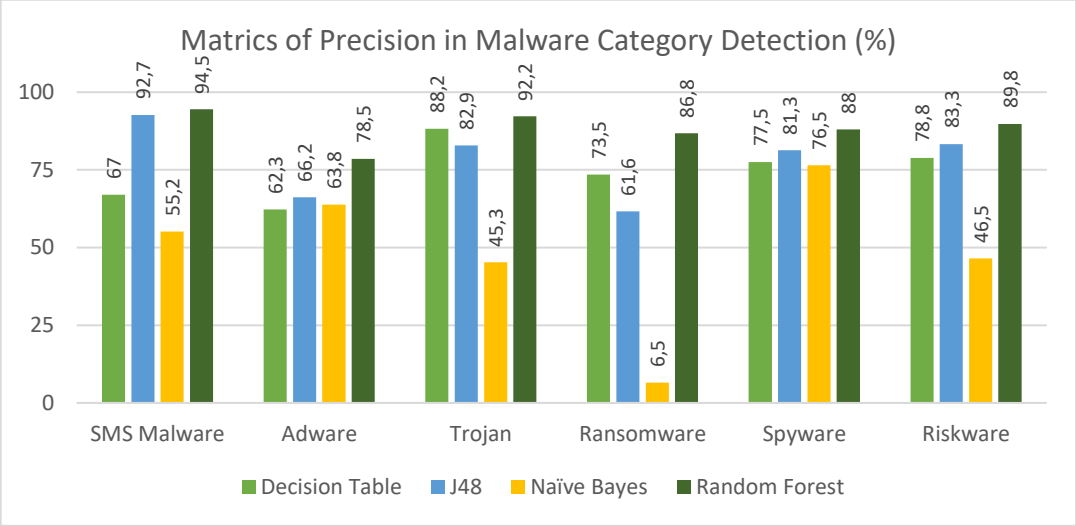

Figure 3. Precision metrics in malware category detection.

Based on the recall metrics, which measure the percentage of correctly identified positive instances, it is clear that higher recall values are associated with fewer false negatives, such as failures to detect malware. The Random Forest algorithm has proven to be highly effective in detecting malware, boasting the highest recall rates across all categories: 98.2% for SMS Malware, 80.8% for Adware, 85.5% for Trojan, and 91.9% for Riskware, thus establishing its superiority over other methods. J48 also delivers commendable performance, with recall values of 95.2% for SMS Malware and 83.8% for Trojan. In contrast, Naive Bayes produces notably low results, particularly for Trojan (13.9%) and Spyware (19.8%). Decision Table demonstrates varying performance, achieving the highest recall of 97.7% for SMS Malware but displaying a notably low recall of 32.5% for Ransomware. The detailed performance of malware detection based on recall metrics is depicted in Figure 4.

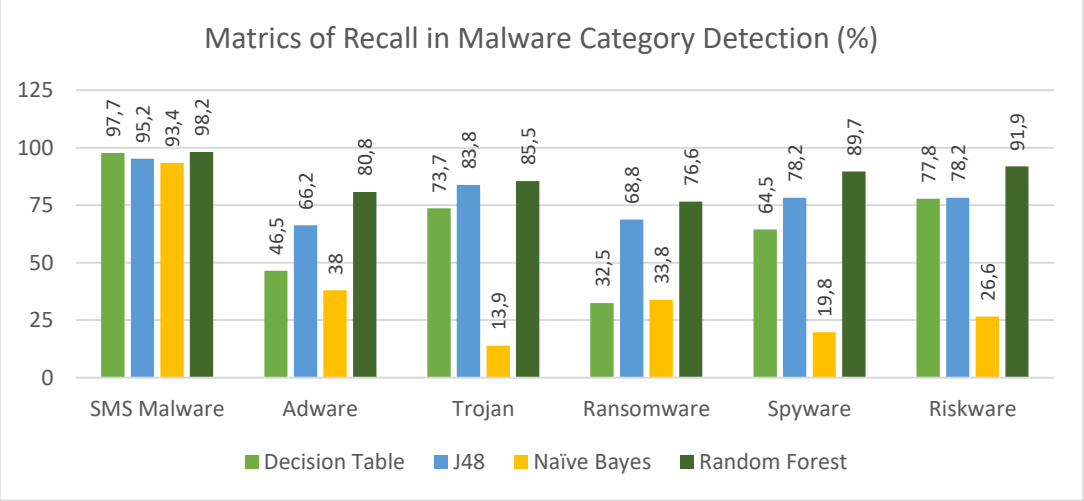

Figure 4. Recall metrics in malware category detection.

The F-Measure metrics, which provide a comprehensive assessment of the model's performance by combining precision and recall, reveal that the Random Forest algorithm is the top-performing model for malware detection. It achieves the highest recall values in the SMS Malware category (96.3%), Trojan (88.8%), Riskware (90.8%), and Spyware (88.8%). The J48 algorithm also demonstrates strong results, particularly in SMS Malware (93.9%) and Trojan (83.3%), although it falls short of the overall performance of the Random Forest. In contrast, Naive Bayes shows the weakest performance, with a recall of only 21.2% for Trojan and 10.9% for Ransomware,

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

making it less effective for these malware categories. For a detailed breakdown of malware detection performance based on F-Measure metrics, refer to Figure 5.

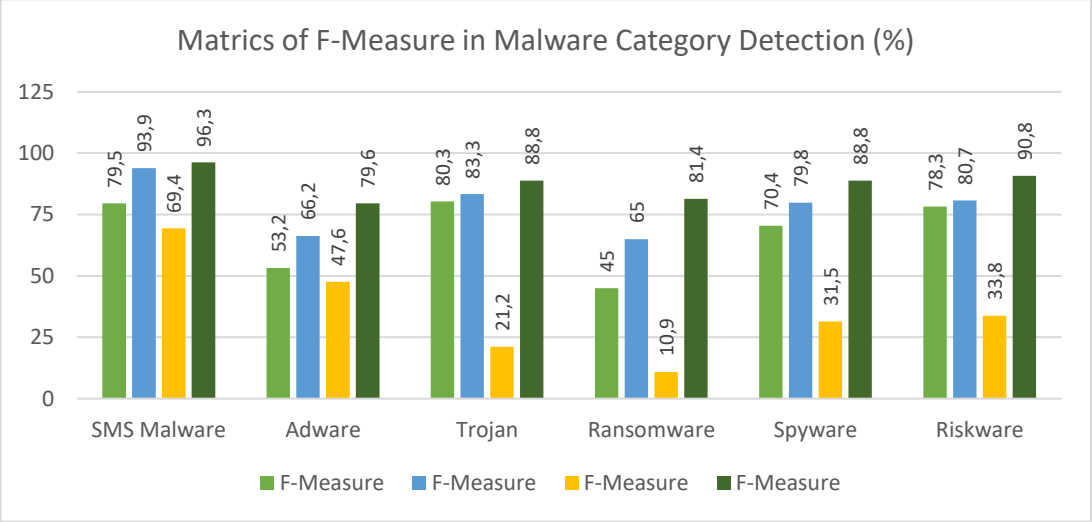

Figure 5. F-Measure metrics in malware category detection.

4. CONCLUSION

Feature selection plays a critical role of malware detection, as it significantly enhances the efficiency and accuracy of models. By eliminating irrelevant features, machine learning algorithms can operate more efficiently and focus on the essential features that impact malware detection. However, in some cases, feature selection can have an adverse effect, as observed in certain algorithms like Decision Table and J48, where the removal of features results in a decline in detection performance. In more complex models such as Random Forest, feature selection typically leads to a substantial improvement in detection performance. Random Forest consistently demonstrates improved results in malware detection after feature selection, applicable to both malware categories and families. Conversely, for simpler algorithms like Naive Bayes, feature selection can reduce accuracy and recall, particularly in certain malware categories, due to the removal of features that significantly influence the classification process.

REFERENCES

[1] G DATA, "Mobile Malware Report, No Let Up with Android Malware." https://www.gdatasoftware.co.uk/news/2019/07/35228-mobile-malware-report-no-let-up-with-android-malware (accessed Mar. 15, 2020).
[2] S. Morgan, "Global Ransomware Damage Costs Predicted To Exceed $265 Billion By 2031," 2024. https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-250-billion-usd-by-2031/ (accessed Oct. 09, 2024).
[3] Kaspersky, "What is Heuristic Analysis?," 2019. https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis (accessed Jan. 29, 2021).
[4] T. Sutter, T. Kehrer, M. Rennhard, B. Tellenbach, and J. Klein, "Dynamic Security Analysis on Android: A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 57261–57287, 2024, doi: 10.1109/ACCESS.2024.3390612.
[5] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A Systematic Literature Review of Android Malware Detection Using Static Analysis," *IEEE Access*, vol. 8, pp. 116363–116379, 2020, doi: 10.1109/ACCESS.2020.3002842.
[6] Z. Wang, Q. Liu, and Y. Chi, "Review of Android Malware Detection Based on Deep Learning," *IEEE Access*, vol. 8, pp. 181102–181126, 2020, doi: 10.1109/ACCESS.2020.3028370.
[7] M. Abuthawabeh and K. Mahmoud, "Enhanced android malware detection and family classification, using conversation-level network traffic features," *Int. Arab J. Inf. Technol.*, vol. 17, no. 4 Special Issue, pp. 607–614, 2020, doi: 10.34028/iajit/17/4A/4.
[8] S. Fallah and A. J. Bidgoly, "Android malware detection using network traffic based on sequential deep learning models," *Softw. Pract. Exp.*, vol. 52, no. 9, pp. 1987–2004, Sep. 2022, doi: 10.1002/spe.3112.
[9] M. Gohari, S. Hashemi, and L. Abdi, "Android Malware Detection and Classification Based on Network Traffic Using Deep Learning," *2021 7th Int. Conf. Web Res. ICWR 2021*, pp. 71–77, 2021, doi: 10.1109/ICWR51868.2021.9443025.
[10] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible android malware detection and family classification using network-flows and API-calls," *Proc. - Int. Carnahan Conf. Secur. Technol.*, vol. 2019-Octob, no. December, 2019, doi: 10.1109/CCST.2019.8888430.
[11] W. Wang *et al.*, "Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019, doi: 10.1109/ACCESS.2019.2918139.
[12] Q. Wang, M. Tang, and J. Fu, "EavesDroid: Eavesdropping User Behaviors via OS Side Channels on Smartphones," *IEEE Internet Things J.*, vol. 11, no. 3, pp. 3979–3993, 2024, doi: 10.1109/JIOT.2023.3298992.
[13] "Different Types of System Calls in OS." https://www.geeksforgeeks.org/different-types-of-system-calls-in-os/

Journal of Information System Research (JOSH)
Volume 6, No. 1, Oktober 2024, pp 349−358
ISSN 2686-228X (media online)
https://ejurnal.seminar-id.com/index.php/josh/
DOI 10.47065/josh.v6i1.6053

(accessed Oct. 01, 2024).
[14] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-Based Malware Detection System for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, Oct. 2011, pp. 15–26, doi: 10.1145/2046614.2046619.
[15] T. S. John, T. Thomas, and S. Emmanuel, "Graph Convolutional Networks for Android Malware Detection with System Call Graphs," in *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, Feb. 2020, pp. 162–170, doi: 10.1109/ISEA-ISAP49340.2020.235015.
[16] Y. Ding, X. Xia, S. Chen, and Y. Li, "A malware detection method based on family behavior graph," *Comput. Secur.*, vol. 73, pp. 73–86, 2018, doi: 10.1016/j.cose.2017.10.007.
[17] H. H. R. Manzil and S. Manohar Naik, "Android malware category detection using a novel feature vector-based machine learning model," *Cybersecurity*, vol. 6, no. 1, 2023, doi: 10.1186/s42400-023-00139-y.
[18] A. Hashem, E. Fiky, A. El Shenawy, and M. A. Madkour, "Android Malware Category and Family Detection and Identification using Machine Learning," 2021.
[19] S. Shakya and M. Dave, "Analysis, Detection, and Classification of Android Malware using System Calls," *Arxiv*, 2022, [Online]. Available: http://arxiv.org/abs/2208.06130.
[20] S. Malik and K. Khatter, "System call analysis of Android Malware families," *Indian J. Sci. Technol.*, vol. 9, no. 21, 2016, doi: 10.17485/ijst/2016/v9i21/90273.
[21] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification," *Proc. - Int. Carnahan Conf. Secur. Technol.*, vol. 2018-Octob, no. Cic, pp. 1–7, 2018, doi: 10.1109/CCST.2018.8585560.
[22] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, *Effective and Efficient Hybrid Android Malware Classification Using Pseudo-Label Stacked Auto-Encoder*, vol. 30, no. 1. Springer US, 2022.
[23] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," in *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522, doi: 10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094.
[24] Statcounter, "Android Version Market Share." https://gs.statcounter.com/os-version-market-share/android (accessed Sep. 02, 2022).
[25] Z. Ni, M. Yang, Z. Ling, J. N. Wu, and J. Luo, "Real-Time Detection of Malicious Behavior in Android Apps," *Proc. - 2016 Int. Conf. Adv. Cloud Big Data, CBD 2016*, pp. 221–227, 2017, doi: 10.1109/CBD.2016.046.
[26] Android, "System Call Android API 24/Nougat." https://android.googlesource.com/platform/bionic/+/refs/heads/nougat-mr2.1-release/libc/SYSCALLS.TXT (accessed Oct. 07, 2024).
[27] O. S. Community, "pselect6." https://man7.org/linux/man-pages/man3/pselect.3p.html (accessed Oct. 10, 2024).
[28] O. S. Community, "setsid." https://man7.org/linux/man-pages/man2/setsid.2.html (accessed Oct. 10, 2024).
[29] O. S. Community, "inotify_init1." https://man7.org/linux/man-pages/man2/inotify_init1.2.html (accessed Oct. 10, 2024).
[30] O. S. Community, "getpgid." https://man7.org/linux/man-pages/man3/getpgid.3p.html.
[31] O. S. Community, "ppoll." https://man7.org/linux/man-pages/man2/poll.2.html (accessed Oct. 10, 2024).
[32] O. S. Community, "fchmod." https://man7.org/linux/man-pages/man3/fchmod.3p.html (accessed Oct. 10, 2024).
[33] O. S. Community, "inotify_add_watch." https://man7.org/linux/man-pages/man2/inotify_add_watch.2.html (accessed Oct. 10, 2024).
[34] O. S. Community, "rt_sigtimedwait." https://man7.org/linux/man-pages/man2/sigwaitinfo.2.html (accessed Oct. 10, 2024).
[35] O. S. Community, "_llseek." https://man7.org/linux/man-pages/man2/llseek.2.html (accessed Oct. 10, 2024).
[36] O. S. Community, "getppid." https://man7.org/linux/man-pages/man3/getppid.3p.html (accessed Oct. 10, 2024).