Malware detection through windows system call analysis

Badis Hammi, Joel Hachem, Ali Rachini, Rida Khatoun

Malware Detection Through Windows System Call Analysis

Badis HAMMI*, Joel HACHEM†, Ali RACHINI‡, Rida KHATOUN§
*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France
badis.hammi@telecom-sudparis.eu
† Kettering University, Michigan, United States
jhachem@kettering.edu
‡Holy Spirit University of Kaslik, Lebanon
alirachini@usek.edu.lb
§Télécom Paris, Institut Polytechnique de Paris, France
rida.khatoun@telecom-paris.fr

***Abstract*—Detecting malware remains a significant challenge, as malware authors constantly develop new techniques to evade traditional signature-based and heuristic-based detection methods. This paper proposes a novel approach to malware detection that analyzes patterns in Windows system calls sequences to identify malicious behaviors. We use a voting classifier, a machine learning model that aggregates predictions from multiple individual models. It determines the final output based on the class that receives the highest likelihood or majority vote from the ensemble of models. We trained the model on large datasets of benign and malicious system call traces to detect anomalies indicative of malware. By focusing on system call behavior rather than static code characteristics, the approach is able to identify novel malware variants without requiring prior knowledge of their signatures. Experiments using a dataset of 42,797 API call sequences from malware samples and 1,079 sequences from benign software demonstrate that voting classifier can achieve high detectionrates while maintaining low false positive rates. This type of Machine Learning-based malware detection could be integrated into an Endpoint Detection and Response (EDR) tool to provide advanced, behavior-based malware detection capabilities.**

I. Introduction

Malware is a serious security threat to critical applications and sectors like education, communication, hospitals, banking, and so on [1]. There is a proliferation in the variety and exotic behavior of malware in recent years. Further, the associated intensity of damage caused on users' data, time, and productivity has increased significantly. For instance a malware attack can completely paralyze an entire system and destroy the data and the hardware infrastructure. In 2023, organizations globally detected 317.59 million ransomware attack attempts according to *Statista*[1]. The *Windows* Operating System (OS) holds the largest global market share and is the most extensively used OS, as reported by *Netmarketshare.com*[2]. As a result, the widespread adoption and popularity of *Windows* OS make it the main target for malware attacks across the globe[3]. Indeed, according to *Dailyhost news*[4], most of the new malware attacks (more than 95%) discovered in 2022 target *Windows* desktop devices, making them the top most devices infected by malware attacks. For this reason, in this work, we primarily focus on *Windows* OS malware detection.

The *Windows* operating system employs two distinct operating modes: user mode and kernel mode. The user mode restricts applications to accessing only their own memory spaces, without direct hardware interaction. While the kernel mode grants full access to the system's hardware and entire physical memory. A system call is an instruction in assembly language that allows a program running in user mode to switch to kernel mode. This is so the kernel can execute a specific task like accessing hardware, reading a file, or sending network packets which can only be done by the kernel. In other words, *Windows* system calls or syscalls provide an interface for programs to interact with the operating system, allowing them to request specific services such as reading or writing to a file, creating a new process, or allocating memory. More precisely, the transition from user mode to kernel mode is being performed through the syscall instruction that resides in each Native API function found in *NTDLL.dll*.

The syscall instruction needs to be specified with a special number (stored in the *eax 32-bit* register) that the kernel requires in order to execute the correct routine. For instance, if a user mode application wants to open a file. It starts by calling the *Windows* API function *CreateFileW*, which is located in the *kernel32.dll* library. The latter function then calls another function, namely *CreateFileW*, in the *kernelbase.dll* library. Finally, *kernelbase.dll* calls a function called *CreateFileInternal* to actually execute the system call and open the file. In the *Windows* operating system, system calls are implemented inside system call stubs located in the Dynamic Link Libraries (DLL) *ntdll.dll* or *win32u.dll*. These DLLs are part of the *Windows* Application Programming Interface (API), so applications make system calls by calling functions in the *Windows* API. Analyzing system calls sequences represents

[1]www.statista.com/statistics/494947/ransomware-attempts-per-year-worldwide/

[2]netmarketshare.com/operating-system-market-share.aspx

[3]www.statista.com/statistics/1238996/top-malware-by-file-type/

[4]https://www.dailyhostnews.com/malware-threats-aimed-at-windows

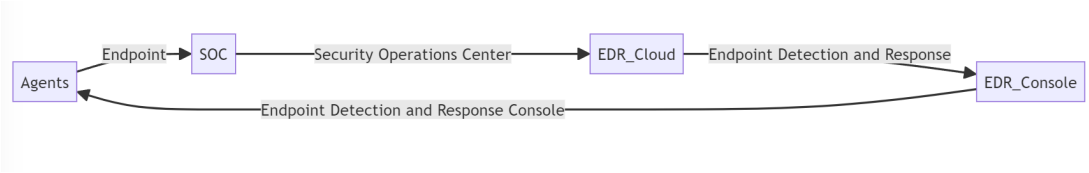

Fig. 1: EDR components

an interesting way to classify code behaviors between normal and abnormal. There are no malicious system calls by itself. However, they often are abused by malware in vulnerable context in order to gain profit. For example the system call "ProtectStreamAsync" could be used by someone to encrypt content before sending Data over the network or it could be used by a ransomware to encrypt victim files before asking for a ransom. Consequently, analyzing the sequence of system calls, their repetition, and their environment would allow classifying code as either normal or abnormal.

In this paper, we propose a malware detection approach based on analyzing the system calls in a *Windows* environment. First, we identify a series of system calls characteristic of ransomware. Then, we evaluate the effectiveness of a behavior-based detection method using a voting classifier.

This dynamic, behavior-based approach could be beneficial in an Endpoint Detection and Response (EDR) tool to detect novel malware variants that may evade traditional signature-based detection. EDR solutions have become an essential component of cybersecurity strategies within organizations. EDR (e.g., SentinelOne Singularity Platform, CrowdStrike Falcon, Microsoft Defender for Endpoint, WatchGuard EPDR and Cisco Secure Endpoint) encompasses a combination of technologies and techniques designed to detect and respond to threats targeting workstations and servers. The architecture of an EDR system is designed for effective monitoring and rapid response to threats on network endpoints. An EDR system generally comprises three primary components as Figure 1 shows:

1) Agents: deployed on each endpoint. These agents collect real-time data across various devices, including desktops, servers, and mobile phones. This includes files, IP addresses, URLs, registry keys, and processes, providing a comprehensive overview of endpoint activities.
2) Central server: collected data is transmitted to a central server, often housed in a cloud environment or the organization's data center. This server employs sophisticated heuristic analysis and machine learning algorithms to detect suspicious behaviors and potential threats.
3) Administration console: security teams use this console to access threat data, investigate incidents in detail, and implement appropriate response measures, such as isolating infected endpoints or removing malicious files.

The integration of artificial intelligence within EDR systems and especially in the central server represents a significant advancement in cybersecurity. Machine learning enables EDR solutions to learn user and application behavior patterns on endpoints. This ability allows for more accurate detection of anomalies, reducing false positives and enhancing overall effectiveness. By proposing a novel approach that focuses on the analysis of *Windows* system call sequences, this study introduces an innovative method to malware detection. Using a voting classifier, a machine learning model that synthesizes insights from multiple models, to identify malicious behaviors based on patterns rather than relying solely on known signatures. The use of large datasets of both benign and malicious system call traces for training adds robustness to our model enhancing its capacity to distinguish malicious activity accurately. Moreover, the potential to integrate this machine learning-based detection into Endpoint Detection and Response tools could mark a step forward in behavior-based detection capabilities.

II. Related Works

Maniriho et al. [2] classify malware detection and analysis systems into four classes: static, dynamic, memory, and hybrid analysis. Figure 2 presents a proposed taxonomy of malware detection techniques and the deployment approaches.

Static analysis is performed without executing the code. It is carried out using tools like disassemblers (e.g., IDA Pro[5]) to extract static features like strings from file headers, operational codes (opcodes), hashes, signatures, metadata, and so on. Static analysis based detection methods operate like signature-based methods. That is, features extracted are compared to a signature database for pattern matching. The primary limitation of static analysis methods is their vulnerability to obfuscation techniques [3]. Additionally, while static approaches that depend on pattern matching can identify known malware signatures, they fall short in detecting zero-day or polymorphic malware. Therefore, static analysis methods often prove to be unreliable.

To address the limitations of static-based malware detection, dynamic analysis examines malware behavior during execution. This analysis is conducted in isolated environments, commonly referred to as sandboxes (e.g., Cuckoo sandbox[6], CWSandbox[7]) to extract features like API calls, running processes, loaded DLLs, registry change, network behaviour, and so on. Dynamic analysis based detection methods. Numerous studies [4][2][5][6] state that dynamic analysis methods are considered the most effective and accurate in terms of malware

[5]https://hex-rays.com/ida-pro/
[6]https://github.com/cuckoosandbox
[7]https://cwsandbox.org/

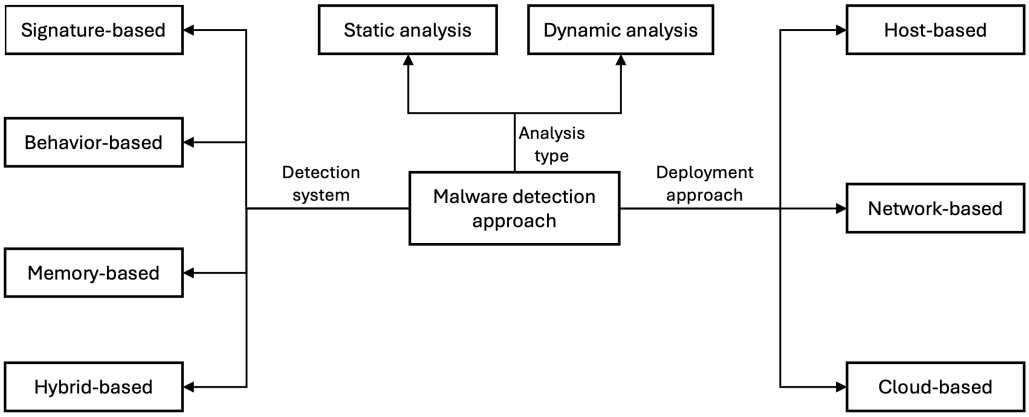

Fig. 2: Taxonomy of malware detection techniques

analysis. In our approach, we use API call sequences collected using a dynamic approach.

Memory analysis techniques focus on monitoring a program's behavior in memory during its execution. This approach offers several advantages, including the ability to reveal memory-based behavioral characteristics of running processes, network activities, and injected processes. It is particularly effective in detecting fileless and memory-resident malware, as it captures real-time interactions and modifications within memory. However, memory analysis is inherently limited to the data present in memory, which can restrict its overall effectiveness and coverage.

Finally, hybrid analysis techniques leverage multiple analysis methods, such as combining dynamic and static analysis, to extract features. By integrating the strengths of each technique, hybrid approaches provide a more comprehensive detection capability. However, implementing these techniques is often more complex due to the need to effectively integrate and manage different analytical processes.

Numerous studies have been conducted on malware detection [7][8][9][2][6]. Our research leverage the *Windows* OS API's calls to identify malicious behaviors and hence malwares. Therefore, in this section, we review previous works that employ system calls for malware detection within *Windows* environments.

Or-Meir et al. [10] proposed a Long Short Term Memory (LSTM) based method for malware classification that relies on system calls extracted from malware while running in the isolated dynamic analysis environment. With this method, long sequences of system calls invoked by malware can be processed in a short time, *Pektaş et al.* [11] analyzed n-grams in malware API call sequences to uncover malicious patterns. They found that frequently occurring patterns could be considered as behavioral representation features of malware. *Lee et al.* [12] introduced a methodology to categorize various types of malware mutants by capturing and clustering n-grams from these mutants. *Tran et al.* [13] applied natural language processing techniques to analyze API call sequences. They divided the sequences using n-grams and assigned weights using Term Frequency-Inverse Document Frequency (TF-IDF). The purpose of TF-IDF was to transform n-grams into numerical features suitable for machine learning algorithms. However, TF-IDF methods do not preserve the contextual relationships between words. *Amer et al.* [4] employed word embedding to understand the contextual relationships between API functions in malware call sequences. They used this method to cluster individual functions with similar contextual traits. Based on these clusters, they developed a method to detect and predict malware using a Markov chain model. *Kim et al.* [14] utilized multiple sequence alignment (MSA) to create "feature-chain" patterns representing malware behavior. These generated patterns were then used to classify malware similar to the training patterns. *Tungjitviboonkun et al.* [15] proposed an approach similar to [14]. They used the longest common API sequence to break down malware API call sequences into simpler sequences. The extracted longest common subsequences were then used as signatures to identify similar malware.

III. Background on Windows API

As mentioned earlier, in the *Windows* OS, user applications cannot directly access hardware or system resources. Instead, they rely on interfaces provided by dynamic-link libraries (DLLs). These libraries offer functionalities to access hardware and system resources. Below is a brief description of the main DLLs:

- Ntoskrnl.dll: Also known as the kernel image, this kernel-mode DLL is a fundamental part of the operating system, providing various system services such as hardware abstraction, memory management, and process management.
- Bootvid.dll: A kernel-mode DLL that provides functions for video graphics adapters (VGA), making it heavily utilized in the *Windows* operating system.
- Hal.dll: This kernel-mode DLL acts as middleware between the kernel and the hardware, providing functions for interfacing with unique chipsets associated with specific motherboards.

- User32.dll: A user-mode DLL that provides components for the user interface, such as buttons and scroll bars, and functionalities for controlling and responding to user actions.
- Kernel32.dll: A user-mode DLL that indexes core functions such as accessing and manipulating memory, files, and hardware.
- Advapi32.dll: A user-mode DLL that provides functionalities for core *Windows* components like service management and the registry.
- Rpcrt4.dll: A user-mode DLL that offers essential functions for remote procedure calls (RPC), used by various *Windows* applications for networking and internet communication.
- Ntdll.dll: A user-mode DLL that serves as the interface to the *Windows* kernel. When a process imports this DLL, it indicates the use of functions to create tasks or new functions not available to standard *Windows* programs, such as process manipulation or hiding functions.
- Gdi32.dll: A user-mode DLL that contains functions for graphical display and manipulation.

Windows API call sequences are widely recognized as a key feature in behavior-based malware detection [4]. While the API call mechanism itself does not distinguish between malicious and benign programs—since both can use the same APIs—the sequence and pattern of these calls can reveal the contextual behavior of the calling process [16][4]. By analyzing these sequences, it becomes possible to infer the intent and nature of the process, aiding in the identification of malicious activity.

IV. Toward a voting classifier for malware detection

A. Ensemble learning

Ensemble techniques such as generative method, boosting, and clustering are commonly employed in machine learning to enhance model performance by combining predictions from several models. The underlying principle is that a group of models (an ensemble) can collectively make more accurate predictions than any individual model alone. This approach can reduce errors, increase accuracy, and make the model more robust. Voting is a fundamental approach in ensemble learning. It involves merging the predictions from various models to determine a final outcome. The two main types of voting in ensemble methods are hard voting and soft voting. In this work we implement both approaches.

1) Hard Voting involves counting the predictions from each base model and selecting the class that receives the most votes as the final prediction. This method is best suited for classification tasks where classes are distinct and exclusive.
2) Soft Voting considers the probability scores that each base model assigns to each class and computes the weighted average of these probabilities to reach the final prediction. Soft voting calculates the average probability for each class and selects the class with the highest weighted probability. It is applicable for both classification and regression tasks.

In the remainder of this section we present the classifiers used by our voting classification approach.

B. Decision tree

Decision Trees (DTs) are a type of supervised learning technique used for classification and regression. It is a simple tree-structured technique that employs decision rules representing tree branches derived from dataset features. The goal of DT is to develop a model that can predict the value of a response value (leaf node) based on learning decision rules [17] [18]. To classify a data point, we begin at the root of the tree and compare the value of the root feature to the value of the feature point. We proceed following the branch corresponding to that value, and the same process is repeated at each internal node and branches of the tree, until we reach the leaf decision node. We summarize the DT algorithm as follows:

- It begins with the entire dataset as the root node.
- It chooses the attribute with the greatest Information Gain (IG) value. IG criterion is computed based on the entropy metric, which measures the uncertainty in a group of data, and it can be calculated as follows:

$$Entropy = -\sum_{i=1}^{k} p_i \log_2 p_i \tag{1}$$

where p_i is the probability of a data point to be selected from class i and k is the number of classes. The IG can be calculated as follows:

$$IG = E_{parent} - E_{children} \tag{2}$$

where E_{parent} is the entropy of parent node and $E_{children}$ is the average entropy of the child nodes (the left and right branches). The Gini Index (GI), which is preferable in the case of categorical variables, is another well-known criterion for splitting the dataset. It can be calculated as follows:

$$GI = 1 - \sum_{i=1}^{k} (p_i)^2 \tag{3}$$

where p_i is the same probability used in IG. The attribute with a lower Gini index should be chosen.

- It repeats the process for each subset, considering the attributes that were not previously selected.

The DT algorithm is simple to understand and interpret because of its hierarchical structure, which can easily show the most important attributes. It can also handle a variety of data types, including continuous, discrete, and categorical data, depending on the criteria used to manipulate the various data types. A complex decision tree, on the other hand, can result in an overfitting problem. Furthermore, minor changes in the data can result in a different decision tree. In terms of computational complexity, it is considered to be more expensive than other algorithms (such as KNN, random forest, XGBoost, naive Bayes, etc).

C. Random forest

Random Forest (RF) is an ensemble technique for improving the model performance by combining multiple decision tree models. This ensemble technique is also known as Bagging or Bootstrap aggregation. Each decision tree model is based on a subset of the entire dataset that is chosen at random. As a result, it works in a divide-and-conquer fashion, and the generated DTs are based on criteria such as the Gini index, information gain, or gain ratio. It can be used for regression as well as classification, and it can handle datasets with categorical or continuous features [19] [20]. In the classification problem, the most-voted class is chosen as the output class for a given data point. The algorithm of RF operates as follows:

- Select randomly several samples from the original dataset with replacement. This step is called bootstrapping.
- Generate a decision tree independently for each sample.
- Apply a majority voting approach to predict the class for a new set of given data.

When compared to another algorithm like DT, the random forest technique has several advantages. It considers the results of multiple decision trees rather than just one. As a result, it does not suffer from the overfitting problem and can produce highly accurate results due to a large number of DTs involved in the process. However, by traversing the tree from the root to the leaf node, it is not easier to interpret the results compared with a DT tree. It also has a computational complexity due to the fact that the process requires several DTs to work in parallel to generate the final results.

D. Naive Bayes

Naive Bayes classifiers are a family of probabilistic classification algorithms that uses Bayes' theorem to predict data point class values, while assuming independence between features. Bayes' theorem is used in Naive Bayes techniques to compute the conditional probabilities (also called posterior probability), and it is given by:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \tag{4}$$

where y is the class label and X represents the values of independent variables (features) for a given data. By considering the independence among features, the Bayes formula can be written as:

$$P(y|x_1, ..., x_n) = P(y) \prod_{i=1}^{n} P(x_i|y) \tag{5}$$

To obtain the class variable (y), we have to maximize the probability in equation (5) as:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y) \tag{6}$$

The class with the highest posterior probability is chosen as the predicted class value.

It is easy to understand and can quickly predict the category of testing data. Furthermore, if the assumption of independence is validated, it can produce produces good results. It can also be used with categorical and numerical variables and for multi-classification. It has several types: multinomial naive Bayes, Bernoulli naive Bayes (for binary variables), and Gaussian naive Bayes (used in case of continuous variables but assumed to be normally distributed). However, logistic regression requires features' independence, which is highly unlikely in most real-world applications.

E. K-nearest neighbors

K-Nearest-Neighbors (KNN) is a popular supervised machine learning algorithm. It can be used for both classification and regression. The main idea behind KNN is to classify data points based on their k nearest neighbors. The main steps of the KNN algorithm are summarized below:

- Specify k, the number of neighbors.
- Compute the distance (Manhattan, Minkowski, ..) among all data points.
- For each data point, keep only the k closest neighbors based on the calculated distance.
- Assign a new data point to the class, in which the majority of its k neighbors belong to that class (majority vote).

It is easy to understand this algorithm that is robust to outliers. In most cases, it also has a relatively high accuracy. However, its precision is directly proportional to the value of k. Several techniques, such as the square root of the dataset or the elbow method have been proposed to select the best value of k. Furthermore, it has a high computational cost, particularly for large data sets, due to the use of the distance that must be calculated between all data points.

V. Performance evaluation

In this section, we present a comparative analysis of the performance of our detection approach against several well-established classification and detection methods. Specifically, we evaluate our approach in relation to Logistic Regression, Random Forest, Naive Bayes, and K-Nearest Neighbors (KNN) classifiers. These methods were selected due to their widespread use and proven effectiveness in malware detection.

To ensure a fair and consistent comparison, we implemented all approaches, including our own, using Python, which offers robust support for machine learning and data analysis through various libraries. For the experimental evaluation, we used a dataset[8] provided by *De Oliveira et al.* [21], a comprehensive resource for analyzing malware behavior. This dataset includes 42,797 API call sequences from malware samples and 1,079 sequences from benign software (goodware). Each sequence comprises the first 100 non-repeated consecutive API calls associated with the parent process, carefully extracted from the 'calls' elements of *Cuckoo Sandbox* reports. This meticulous data extraction process ensures the integrity and relevance of the sequences used in our classification tasks, providing a

[8]https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences

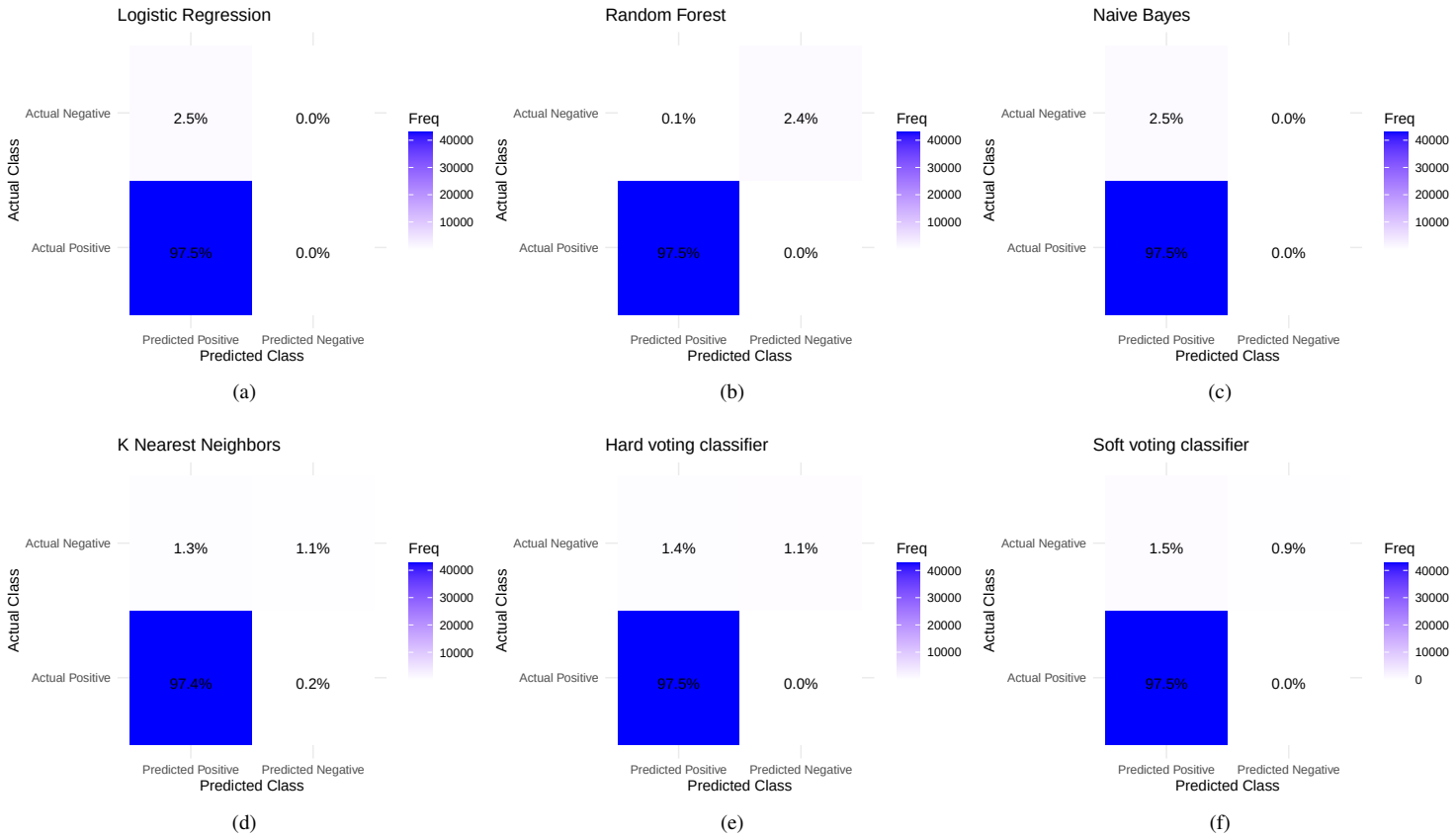

Fig. 3: Confusion matrices for the classification methods tested: (a) Logistic regression; (b) Random forest; (c) Naive Bayes; (d) K nearest neighbors; (e) Hard voting classifier; (f) Soft voting classifier

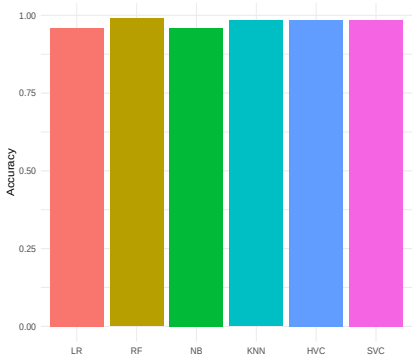

Fig. 4: Accuracy of the tested classifiers

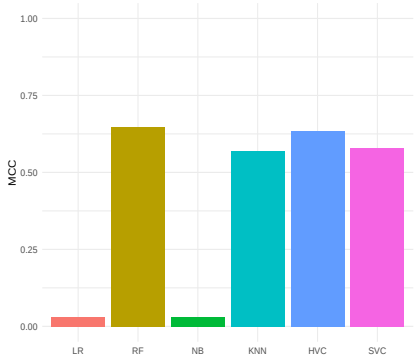

Fig. 5: MCC of the tested classifiers

solid foundation for assessing the effectiveness of different detection methods. Figure 3 depicts the confusion matrices obtained through the different detection techniques.

For performance evaluation, we use both accuracy and Matthews Correlation Coefficient (MCC) as comparison metrics. While accuracy is a widely used metric, representing the ratio of correctly predicted instances (true positives and true negatives) to the total number of instances, it has limitations, particularly in imbalanced datasets. Relying solely on accuracy can be misleading, as it does not provide a comprehensive view of a model's performance across all classes. In contrast, the MCC offers a more balanced evaluation by considering all four categories of the confusion matrix: true positives, true negatives, false positives, and false negatives. This makes MCC a more reliable indicator of performance. Figures 4 and 5 illustrate the accuracy and MCC values obtained from the different classifiers, respectively.

The performance evaluation of the various classifiers for detecting malicious activity on the dataset reveals significant differences in their accuracy and Matthews Correlation Co-

efficient (MCC). Logistic Regression and Naive Bayes both achieve an accuracy of 96%, but their MCC values are notably low at 0.03, indicating that these models may not effectively differentiate between true positive and negative classifications. In contrast, the Random Forest classifier demonstrates a higher accuracy of 99% with a substantially improved MCC of 0.647, suggesting a better balance in its predictions. The k-Nearest Neighbors (KNN) classifier also performs well, with an accuracy of 98.3% and an MCC of 0.57, indicating a solid performance in distinguishing between malicious and benign activities. The Voting Classifiers, both hard and soft, exhibit strong results as well. The Hard Voting setup achieves an accuracy of 98.4% and an MCC of 0.635, while the Soft Voting setup has a slightly lower accuracy of 98.1% and an MCC of 0.58. These results demonstrate that our approach of Voting Classifiers can effectively leverage the strengths of individual models, resulting in improved overall performance. Overall, the Random Forest and Voting Classifiers stand out for their higher MCC values, reflecting their superior capability in making reliable predictions compared to the simpler models like Logistic Regression and Naive Bayes.

While both the voting classifier and the Random Forest classifier demonstrated similar performance levels in terms of detection accuracy and MCC, the voting classifier offers distinct advantages. The key strength of the voting classifier lies in its ability to leverage the diversity of multiple models, which can lead to greater robustness and resilience against overfitting. Unlike the Random Forest, which relies on an ensemble of decision trees with inherent similarities, the voting classifier combines a heterogeneous set of models, each contributing unique perspectives to the final decision. This diversity can be particularly beneficial in scenarios where the underlying data distributions are complex or where certain models may excel in capturing specific patterns. Additionally, the flexibility of the voting classifier allows for incorporating models with different strengths, potentially enhancing generalization to unseen data.

VI. Conclusion

This paper addresses the challenge of detecting malware, which is an ongoing issue as malware authors continuously develop new techniques to evade traditional signature-based. This paper proposes a novel approach to malware detection that analyzes patterns in *Windows* system call sequences to identify malicious behaviors. In this work, we use a voting classifier which combines the predictions of multiple sub-models to arrive at a more accurate and robust output. We train the voting classifier on large datasets of benign and malicious system call traces to detect anomalies indicative of malware. Focusing on system call behavior, rather than static code characteristics allows the approach to identify novel malware variants without requiring prior knowledge of their signatures. The high detection rates and low false positives rate observed in the experiments highlight the potential of this approach to contribute to the development of more effective and adaptable malware detection systems.

References

[1] Badis Hammi and Sherali Zeadally. Software supply-chain security: Issues and countermeasures. *Computer*, 56(7):54–66, 2023.

[2] Pascal Maniriho, Abdun Naser Mahmood, and Mohammad Jabed Morshed Chowdhury. A systematic literature review on windows malware detection: Techniques, research issues, and future directions. *Journal of Systems and Software*, page 111921, 2023.

[3] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[4] Eslam Amer and Ivan Zelinka. A dynamic windows malware detection and prediction method based on contextual understanding of api call sequence. *Computers & Security*, 92:101760, 2020.

[5] Ömer Aslan Aslan and Refik Samet. A comprehensive review on malware detection approaches. *IEEE access*, 8:6249–6271, 2020.

[6] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. Dynamic malware analysis in the modern era—a state of the art survey. *ACM Computing Surveys (CSUR)*, 52(5):1–48, 2019.

[7] Mohana Gopinath and Sibi Chakkaravarthy Sethuraman. A comprehensive survey on deep learning based malware detection techniques. *Computer Science Review*, 47:100529, 2023.

[8] Faitouri A Aboaoja, Anazida Zainal, Fuad A Ghaleb, Bander Ali Saleh Al-Rimy, Taiseer Abdalla Elfadil Eisa, and Asma Abbas Hassan Elnour. Malware detection issues, challenges, and future directions: A survey. *Applied Sciences*, 12(17):8482, 2022.

[9] Jagsir Singh and Jaswinder Singh. A survey on machine learning-based malware detection in executable files. *Journal of Systems Architecture*, 112:101861, 2021.

[10] Ori Or-Meir, Aviad Cohen, Yuval Elovici, Lior Rokach, and Nir Nissim. Pay attention: Improving classification of pe malware using attention mechanisms based on system call analysis. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[11] Abdurrahman Pektaş and Tankut Acarman. Malware classification based on api calls and behaviour analysis. *IET Information Security*, 12(2):107–117, 2018.

[12] Taejin Lee, Bomin Choi, Youngsang Shin, and Jin Kwak. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. *The Journal of Supercomputing*, 74:3489–3503, 2018.

[13] Trung Kien Tran and Hiroshi Sato. Nlp-based approaches for malware classification from api sequences. In *2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES)*, pages 101–105. IEEE, 2017.

[14] Hyun-Joo Kim, Jong-Hyun Kim, Jung-Tai Kim, Ik-Kyun Kim, and Tai-Myung Chung. Feature-chain based malware detection using multiple sequence alignment of api call. *IEICE TRANSACTIONS on Information and Systems*, 99(4):1071–1080, 2016.

[15] Thotsaphon Tungjitviboonkun and Vasin Suttichaya. Complexity reduction on API call sequence alignment using unique API word sequence. In *2017 21st international computer science and engineering conference (ICSEC)*, pages 1–5. IEEE, 2017.

[16] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. A novel approach to detect malware based on api call sequence analysis. *International Journal of Distributed Sensor Networks*, 11(6):659101, 2015.

[17] Sherali Zeadally and Michail Tsikerdekis. Securing internet of things (iot) with machine learning. *International Journal of Communication Systems*, 33(1):e4169, 2020.

[18] Abir Mchergui, Tarek Moulahi, and Sherali Zeadally. Survey on artificial intelligence (ai) techniques for vehicular ad-hoc networks (vanets). *Vehicular Communications*, page 100403, 2021.

[19] V Rodriguez-Galiano, M Sanchez-Castillo, M Chica-Olmo, and MJOGR Chica-Rivas. Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines. *Ore Geology Reviews*, 71:804–818, 2015.

[20] Iftikhar Ahmad, Mohammad Basheri, Muhammad Javed Iqbal, and Aneel Rahim. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE access*, 6:33789–33795, 2018.

[21] Angelo Schranko de Oliveira and Renato José Sassi. Behavioral malware detection using deep graph convolutional neural networks. *Authorea Preprints*, 2023.