Asim Azhar, 2026, 14:2
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Malware Detection Using Machine Learning

Asim Azhar, Richa Gupta, Mudita Saxena, Dipanshu Singh, Rahul Anjana
School Of Computer Science & Engineering, IILM University - Greater Noida

Abstract- This project focuses on detecting Malicious Android applications using supervised machine learning techniques. A permission based dataset is used where each application is represented by behavioral features such as requested permission. After preprocessing the dataset, machine learning algorithms including Random Forest, Decision tree and Naive Bayes are implemented using the WEKA framework in Java. The models are evaluated using 10-fold cross validation and standard performance metrics. The objective of the project is to develop an automated, accurate and safe malware detection system without executing malicious code.

Keywords: Android Permissions, Machine Learning, Malware Detection, Mobile Security, Static Analysis.

I. INTRODUCTION

The exponential proliferation of digital technologies, ubiquitous internet connectivity, and the rapid expansion of mobile computing have fundamentally reshaped the global technological landscape [6]. Consequently, the cybersecurity domain has witnessed an unprecedented escalation in both the volume and sophistication of malicious software (malware) [2]. Threat actors continuously deploy diverse malware variants—including ransomware, trojans, worms, and spyware—to compromise systems, exfiltrate sensitive data, and disrupt critical network infrastructures. Traditional defensive mechanisms rely predominantly on signature-based detection paradigms, which depend on maintaining exhaustive databases of known malware hashes [7]. While computationally efficient for identifying historical threats, these deterministic systems exhibit critical shortcomings when confronted with zero-day vulnerabilities, polymorphic code, and sophisticated obfuscation techniques designed to evade standard detection [9]. Unlike static signature matching, ML algorithms possess the predictive capacity to discern complex, nonlinear patterns within vast datasets, enabling the identification of novel and anomalous behaviors indicative of malicious intent without relying on predefined rules [10],[13].

This research focuses on architecting an ML-driven malware detection framework specifically tailored for the Android operating system, which remains a primary target for cybercriminals due to its open ecosystem and immense global market share [2]. The proposed methodology leverages static analysis techniques, evaluating applications without executing their code. Specifically, the system scrutinizes the AndroidManifest.xml file to extract application permission requests [12]. Because malicious applications frequently exhibit anomalous or excessive permission matrices—such as unjustified access to SMS, contacts, or system storage—these features serve as powerful discriminators [5],[16]. Utilizing the Waikato Environment for Knowledge Analysis (Weka) [17] and IntelliJ IDEA, this study systematically implements and evaluates various machine learning classifiers to distinguish between benign and malicious software, ultimately contributing to more scalable and resilient mobile threat mitigation strategies [1].

II. LITERATURE REVIEW

The explosion of the Android mobile operating system, currently holding the dominant global market share, has inevitably established it as the primary vector for malicious software [2],[6]. Since traditional detection methods struggle against the accelerating volume and sophistication of modern malware, recent research has shifted extensively toward Machine Learning (ML) as a proactive defense mechanism [9]. Literature in this domain can be categorized by the analysis types employed—dynamic, static, or hybrid—with a significant body of work confirming the efficacy of static permission analysis for efficient classification [10].

© 2026 Asim Azhar, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

The Shift from Signature to Machine Learning : Early literature established that traditional signature-based systems, relying on specific file hashes, were computationally efficient but structurally inadequate against modern obfuscation techniques, polymorphic malware, and zero-day attacks [1],[7]. Researchers recognized that detection systems must evolve from reactive (identifying known threats) to predictive (classifying anomalous behavior). This catalyzed the application of machine learning algorithms. Studies such as those by Schmidt et al [8]. (2009) were pioneers, demonstrating that extracting features from the application's structure could yield robust detection models without executing the code.

Static Analysis and Permission Scrutiny: Within the realm of Android malware detection, static analysis has emerged as a favored methodology because it provides a rapid, non-executional overview of an application's potential capabilities before it can inflict damage [12]. A significant sub-genre of this research focuses exclusively on application permissions requested within the AndroidManifest.xml file. Every Android application must declare the access it requires to sensitive user data or hardware components (e.g., GPS, Camera, SMS). Scholars have long validated that malicious applications frequently request excessive, unnecessary, or suspicious combinations of permissions when compared to their benign functional counterparts [16],[19]. For example, a simple "Flashlight" application requesting permissions to READ_SMS and ACCESS_FINE_LOCATION is immediately flagged as anomalous. This permission-based approach is often favored in static analysis due to its computational lightweight nature and the ease of extraction, making it highly suitable for real-time mobile security solutions (Sanz et al., 2012) [5].

Evolving Models and Performance: The progression of research has seen a shift from basic permission counting toward analyzing complex patterns [11]. Early studies focused on the mere presence of high-risk permissions. However, later contributions recognized that malware frequently combines benign permissions to execute malicious functionality (a "suspicious combination"), such as combining internet access with contacts access to facilitate data exfiltration [3]. Numerous studies have compared diverse ML algorithms to classify this permission data [15]. Research utilising ensemble methods, such as Random Forest, frequently reports superior accuracy rates on permission datasets, often exceeding 90% in controlled environments. Naive Bayes, Random Forest and Decision Trees also show significant promise, though their performance often fluctuates depending on data preprocessing and the size of the training corpus (Yerima et al., 2013) [4].

Current Challenges and Gaps: Despite the successes, literature acknowledges significant gaps. Static analysis is vulnerable to evasion tactics, particularly dynamic code loading, where malicious functionality is downloaded after the initial static check [14]. Furthermore, as the Android permission model evolves (shifting toward dynamic, runtime permissions), purely static checks must adapt. Recent scholarly work emphasizes the importance of robust feature engineering and understanding adversarial evasion techniques [20] to combat "model aging," ensuring classifiers remain relevant against continuously evolving malware variants [18].

III. METHODOLOGY

The methodology constitutes the foundational, procedural blueprint utilized to architect, develop, and rigorously validate the proposed Android malware detection system. In the highly dynamic and adversarial domain of mobile cybersecurity, where threat actors continuously evolve their obfuscation and evasion tactics, a loosely defined experimental design is fundamentally insufficient [9]. Therefore, establishing a highly systematic framework is paramount to the project's success. To guarantee absolute scientific rigor and ensure the complete reproducibility of our findings by the broader academic and security research community, the experimental workflow is deliberately constructed and logically partitioned into sequential stages. This

structured approach acts as a strategic safeguard, allowing for precise control, isolation, and optimization of variables at every juncture of the research pipeline—from the initial handling of data to the final algorithmic evaluation. Rather than treating the machine learning process as an opaque "black box," this step-by-step methodological philosophy ensures maximum transparency.

It meticulously maps the complex transformation of unstructured, raw application binaries—which are inherently noisy and difficult to analyze—into clean, structured mathematical features. By strictly adhering to this comprehensive framework, the project ensures that the resulting predictive intelligence is not merely a product of chance, but a highly accurate, reliable defense mechanism capable of discerning subtle malicious patterns within the Android ecosystem [8].

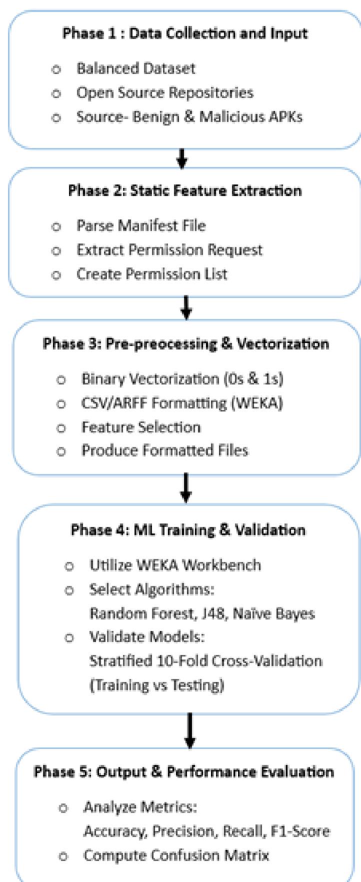

Fig (3.1): Methodology Workflow

1. Data Collection Methods

The foundational step of any robust machine learning pipeline is the acquisition of a high-quality, representative dataset. For this project, we will compile a strictly balanced dataset comprising an equal distribution of benign (safe) and malicious Android applications. A balanced dataset is critical to prevent class imbalance, which can severely skew the classifier's predictive capability and result in an artificially high accuracy driven by the majority class [10].

Benign applications will be sourced from reputable repositories such as the official Google Play Store, utilizing web scraping or open-source datasets like AndroZoo. To ensure these applications are genuinely benign, they will be cross-referenced against engines like VirusTotal [7]. Conversely, the malicious samples will be aggregated from established cybersecurity research repositories, such as the Drebin dataset, CICMalDroid, or the Android Malware Genome Project [1],[2]. These repositories provide a diverse array of malware families—including trojans, spyware, ransomware, and SMS botnets—ensuring our model learns a comprehensive spectrum of malicious behavior rather than overfitting to a single threat type [6].

2. Feature Extraction Pipeline

Because this project emphasizes static analysis, the feature extraction phase operates strictly without executing the applications [8]. Android applications are distributed as APK (Android Package) files, which are essentially compressed archives. Before any machine learning can occur, we must simulate a reverse-engineering pipeline to unpack these archives.

The focal point of this pipeline is the AndroidManifest.xml file [12]. In the Android operating system, every application must explicitly declare the permissions it requires to access system resources (e.g.,android. permission.CAMERA, android.permission .READ_CONTACTS, and roid.permission.SEND_SMS). Our automated pipeline will parse the manifest file of each APK in

the dataset to extract these declared permissions [5],[16].

Once extracted, this raw text data must be transformed into a machine-readable format. We will utilize a binary vectorization technique [15]. A universal list of all possible Android permissions will be created. For every application, a binary feature vector will be generated: if the application requests a specific permission, the corresponding feature value is marked as '1' (True); if not, it is marked as '0' (False). This transforms the applications into a structured mathematical matrix suitable for algorithmic processing.

3. Data Preprocessing and Dimensionality Reduction

Raw data extracted from thousands of APKs is inherently noisy and highly dimensional, as the Android framework contains hundreds of unique permissions. Feeding this raw matrix directly into a classifier is computationally expensive and prone to the "curse of dimensionality," which can degrade model performance.

To mitigate this, the dataset will undergo rigorous preprocessing. First, the data will be formatted into an ARFF (Attribute-Relation File Format) or a standard CSV file, which are the native ingestion formats required by the Weka machine learning workbench [17]. Next, we will apply Feature Selection techniques to identify and isolate the most discriminative permissions [11]. Algorithms such as Information Gain (Entropy) or Chi-Square attribute evaluation will be utilized to rank the permissions based on their predictive power [19]. Permissions that are universally requested by both benign and malicious apps (like INTERNET) offer little information and will be discarded. By reducing the feature space to only the most critical attributes—specifically, those suspicious combinations highlighted in our hypothesis—we significantly optimize training speed and enhance the classifier's accuracy.

4. Algorithms Applied & Experimental Setup

With a refined, high-dimensional dataset prepared, the modeling phase will utilize the Weka workbench [17] to train and evaluate multiple classification algorithms. Given the binary nature of our problem (Benign vs. Malicious) [13], we will focus on algorithms known for handling boolean categorical data efficiently:

- Decision Trees (J48): This algorithm, an open-source implementation of C4.5, will be employed for its high interpretability. It creates a tree-like model of decisions, allowing us to visually trace the exact permission combinations (rules) that lead to a "malicious" verdict.
- Random Forest: To counter the potential overfitting of single decision trees, this ensemble learning method will be utilized. By constructing a multitude of decision trees during training and outputting the mode of the classes, Random Forest provides exceptional robustness and accuracy, particularly on complex, non-linear permission datasets [9].
- Naive Bayes: We will also establish a baseline using Naive Bayes. This probabilistic classifier applies Bayes' theorem with the "naive" assumption of conditional independence between every pair of features [4]. Comparing its performance against Random Forest will reveal whether the inter-dependencies between permissions (the "combinations") are truly the driving factor in malware detection.

5. Implementation Environment

The system was implemented in Java using the Weka machine learning API within the IntelliJ IDEA development environment. The dataset was loaded dynamically based on user input, allowing flexibility in evaluating different datasets. The program performs preprocessing, model training, and evaluation programmatically and generates structured output including accuracy metrics and confusion matrices. Additionally, the system provides functionality to save results as timestamped text files, ensuring reproducibility and proper record-keeping of experimental outcomes.

The architectural integration and custom scripting required to bridge these phases will be managed utilizing the IntelliJ IDEA integrated development environment (IDE). IntelliJ will serve as the central hub for our codebase. Custom scripts (likely in Java or Python) will be written to automate the batch unpacking of APKs (using tools like Apktool), the parsing of the XML manifests, the boolean vector generation, and the final CSV/ARFF formatting. Furthermore, the Weka API can be directly integrated into our IntelliJ project, allowing for programmatic execution of the machine learning pipeline rather than relying solely on Weka's graphical user interface [17].

6. Testing, Validation, and Evaluation Metrics

To guarantee that our trained models generalize effectively to novel, unseen threats, a rigorous validation protocol is mandatory [20]. A simple train/test split is highly susceptible to data variance. Therefore, we will employ a Stratified 10-Fold Cross-Validation methodology within Weka. The dataset will be randomly partitioned into ten equal subsets (folds). The model will be trained on nine folds and tested on the remaining one, iterating this process ten times so that every data point is used for both training and testing.

The ultimate efficacy of the models will be quantified using a confusion matrix to derive four primary metrics [1],[18]:

- Accuracy: The overall percentage of correctly classified applications.
- Precision: The ratio of true positive malicious detections to all predicted malicious detections. High precision is vital to minimize False Positives, ensuring benign apps are not wrongfully flagged.
- Recall (Sensitivity): The ratio of true positive detections to all actual malware samples. High recall is critical to minimize False Negatives, ensuring no malware slips through the system.
- F1-Score: The harmonic mean of Precision and Recall, providing a single, comprehensive metric to evaluate the model's balance and overall diagnostic performance.

IV. RESULT

In the experimental evaluation of our proposed framework we present a thorough evaluation of supervised machine learning techniques in a permission-driven static analysis setting to classify Android applications. The dataset consists of 10,686 instances with 535 attributes that are derived from the permissions an application requests. A balanced dataset between benign and malicious samples was preprocessed in an appropriate manner, and evaluated using the stratified 10-fold cross-validation method available in Weka data mining tool to measure the model’s ability to generalize and to prevent overfitting. A more thorough evaluation of a model is achieved by computing its accuracy as well as precision, recall (sensitivity) and F1-score, all available in the confusion matrix.

Experimental results obtained with all implemented classifiers show good classification performance and it varies on the basis of the datasets under test. The primary dataset achieved an accuracy of approximately 86.84% with the Random Forest, approximately 86.40% with J48 Decision Tree, and approximately 75.40% with the Naive Bayes classifier.

```
=== Accuracy ===

Random Forest Accuracy: 86.843%
J48 Accuracy: 86.403%
Naive Bayes Accuracy: 75.398%
```

Fig(4.1) : Android Permission-Based Dataset

Experimental verification was also performed on other datasets, and the methods, especially Random Forest, achieved high accuracy rates up to almost 100% classification accuracy for some of them. The performance of models on novel datasets depends highly on the characteristics of the dataset, including feature distribution and class separability. Extremely high accuracy also implies high feature separability

or even overfitting, and therefore the results must be viewed with reserve.

The good results obtained with Random Forest are possibly due to the ensemble learning technique used in this algorithm (a combination of multiple decision trees), which often leads to good generalization ability and a decrease of overfitting. In fact, also J48, which is based on a single decision tree, gave very good results, because it is easily interpretable and provides a set of rules easily explained for users. Naive Bayes achieved the worst results because it is based on a naive assumption of independence of features, that is not verified in permission-based datasets, where features are normally dependent.

Decision Tree (J48) with around 99.95% accuracy also showed promising results. The output is easy to understand due to its tree-like structure and provides the possibility to explicitly specify decision rules at each node by successively splitting features in a hierarchical way. Even complex classification decisions can be followed by others. The deep trees, however, are still prone to overfitting and do not generalize as well as combinations of multiple models for hard classification do.

Results indicate that while the Naive Bayes classifier, a probabilistic classifier based on Bayes' theorem and assumptions of feature independence, achieves roughly 66.35% accuracy on the primary dataset, it is still insufficient, arguably due to the interdependent nature of Android permissions and their joint contributions to malware behavior. Although the Naive Bayes classifier is inefficient from a computational perspective, it acts as a good baseline for comparison between the approaches developed within this work.

```
=== Accuracy ===

Random Forest Accuracy: 100.000%
J48 Accuracy: 99.995%
Naive Bayes Accuracy: 66.349%
```

Fig(4.2) : Android Behavior-Based Dataset

The performance statistics extracted from the classification confusion output further detailed the accuracy of each of the models by quantifying the actual true positives and false negatives for each classifier. The results for the Random Forest ensemble method were particularly striking with an excellent true positive rate with very low false negatives. In other words Random Forest was an extremely accurate detector of malware. The performance of the J48 decision tree and Naive Bayes classifiers also remained relatively balanced with error rates

smaller than classification error 0.250. Interestingly Naive Bayes had many more false negatives than the other methods.

Our results suggest that ensemble methods like Random Forest are more effective for dealing with feature interactions, in contrast to the lower performance of probabilistic methods like Naive Bayes for cases involving feature dependencies.

In summary, our results demonstrate that permission-based static analysis combined with machine learning algorithms results in a scalable, efficient, and accurate solution to the Android malware detection problem. In particular, the results of the Random Forest model support the use of ensemble learning approaches to handle the complexities of Android malware classification and suggest the potential of our approach for practical implementations and further enhancements.

To increase the adaptability of the system, the most appropriate classifier is determined dynamically. Running tests showed that the best performing model of classifiers included in the system is the most accurate one, and it is selected automatically for classification of new records, which means there is no need for hardcoded conclusions.

V. CONCLUSION

In this paper, we present an automated machine learning-based approach for the detection and prevention of malicious Android applications using permission-based static analysis. Using supervised machine learning classifiers and a robust feature

extraction methodology, we developed a static analysis-based framework to detect malicious and normal Android applications without executing them, and achieved high accuracy rates for safe and efficient malicious application detection.

Comparative experiments show that various methods of ensemble learning considerably outperform single learning approaches. In this study the highest accuracy was obtained by using a Random Forest classifier based on 200 decision trees. Such a method can process large feature spaces efficiently, it can model non-linear relationships and has a low level of overfitting through the bootstrapping of training instances. Other ensemble learning approaches like Decision Tree (for their interpretability, provided as a set of rules, and as a basis for comparison with Random Forest) and Naive Bayes (for its high computation speed) were less accurate. Notably the Random Forest classifier's performance did not improve by enlarging the number of decision trees.

This paper presents an advanced approach for network behavior analysis and automated detection of known malware. Furthermore, we demonstrate that certain permission combinations serve as strong features that enable discrimination between benign and malicious behavior. The model achieves strong recall and demonstrates robustness through stratified 10-fold cross-validation. The model was fine-tuned in order to achieve low false positive and false negative rates, suitable for real-world deployment.

In the present work we successfully applied a machine learning method to detect Android malware using permission-based static analysis of Android applications. Experimental results show that the best results were obtained by the Random Forest model. Another approach that achieved competitive results is the J48 Decision Tree, which has the advantage of being very understandable for a human analyst. We also implemented the simple Naive Bayes model as a baseline approach, but it reached the worst results. The proposed system is versatile and adaptive for processing different dataset types and automatically comparing multiple detection models for evaluation.

This system can cater to the dynamic nature of real-world datasets. Future directions would include incorporating functionality such as API call and dynamic behavior analysis, in order to enhance the detection accuracy with the introduction of deep learning techniques in tackling evolving malware threats.

VI. REFERENCES

1. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, 2014, pp. 23-36.
2. Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in 2012 IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 2012, pp. 95-109.
3. Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in android," in International Conference on Security and Privacy in Communication Systems (SecureComm), Sydney, Australia, 2013, pp. 86-103.
4. S. Y. Yerima, S. Sezer, I. Muttik, and M. A. Mukhtar, "A new Android malware detection approach using Bayesian classification," in 2014 28th International Conference on Advanced Information Networking and Applications (AINA), Victoria, BC, Canada, 2014, pp. 121-128.
5. B. Sanz, A. Iglesias, C. Zamorano, J. Carlos, and G. Maciá-Fernández, "PUMA: Permission usage to detect malware in Android," in International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), Natal, Brazil, 2012, pp. 288-295.
6. A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), Chicago, IL, USA, 2011, pp. 3-14.

7. W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS), Chicago, IL, USA, 2009, pp. 235-245.
8. A. D. Schmidt, R. Bye, H. G. Schmidt, J. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Sahin, "Static analysis of executables for collaborative malware detection on android," in 2009 IEEE International Conference on Communications (ICC), Dresden, Germany, 2009, pp. 1-5.
9. A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, "A review of machine learning based malware detection," Computers & Security, vol. 53, pp. 15-26, 2015.
10. A. Sharma and S. K. Dash, "Effectiveness of machine learning techniques in Android malware detection," International Journal of Computer Science Issues (IJCSI), vol. 11, no. 1, pp. 171-177, 2014.
11. D. J. Wu, C. H. Mao, T. E. Wei, H. M. Lee, and K. P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS), Seoul, South Korea, 2012, pp. 82-83.
12. R. Sato, D. Chiba, and S. Goto, "Detecting Android malware by analyzing manifest files," in Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Kaohsiung, Taiwan, 2013, pp. 1-5.
13. J. Sahs and L. Kraemer, "A machine learning approach to android malware detection," in 2012 European Intelligence and Security Informatics Conference (EISIC), Odense, Denmark, 2012, pp. 141-147.
14. A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': a behavioral malware detection framework for android devices," Journal of Intelligent Information Systems (JIIS), vol. 38, no. 1, pp. 161-190, 2012.
15. N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI), Herndon, VA, USA, 2013, pp. 300-305.
16. W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," IEEE Transactions on Information Forensics and Security, vol. 9, no. 11, pp. 1869-1882, 2014.
17. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," SIGKDD Explorations Newsletter, vol. 11, no. 1, pp. 10-18, 2009.
18. A. Altaher, "An improved Android malware detection scheme based on an evolving hybrid neuro-fuzzy classifier (EHNFC) and API calls," Applied Soft Computing, vol. 54, pp. 433-440, 2017.
19. J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, 2018.
20. K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in European Symposium on Research in Computer Security (ESORICS), Oslo, Norway, 2017, pp. 62-79.