A Multidimensional Detection Model of Android Malicious Applications Based on Dynamic and Static Analysis

Hao Zhang[1,2(✉)], Donglan Liu[1], Xin Liu[1], Lei Ma[1], Rui Wang[1], Fangzhe Zhang[1], Lili Sun[1], and Fuhui Zhao[1]

[1] State Grid Shandong Electric Power Research Institute, Jinan, China
[2] Shandong Smart Grid Technology Innovation Center, Jinan, China
zhanghao_dky@163.com

Abstract. This paper presents an approach utilizing static and dynamic analysis techniques to identify malicious Android applications. We extract static features, such as certificate information, and monitor real-time behavior to capture application properties. Using machine learning, our approach accurately differentiate between benign and malicious applications. We introduce the concept of "Multi-dimensional features", combining static and dynamic features into unique application fingerprints. This enables us to infer application families and target groups of related malware. Tested on a dataset of 8000 applications, our approach demonstrates high detection rates, low false positive and false negative rates. The results highlight the effectiveness of our comprehensive analysis in accurately identifying and mitigating Android malware threats.

Keywords: Android malware · Dynamic and static analysis · Multi-dimensional features

1 Introduction

The rise of smartphones and their portable, feature-rich nature has made Android the operating system of choice on over 85% of global devices [2]. However, the openness of the Android platform has led to a surge in malicious applications, resulting in security and performance issues. As the Android platform becomes more popular, the number and complexity of these harmful apps grow exponentially [7].

The multifaceted nature of these applications, covering malware types like spyware, adware, ransomware, and banking Trojans, and the ability of some to evade standard detection measures, adds to the complexity of the problem. Current techniques, such as signature-based detection and behavioral analysis, present limitations [9].

To better combat Android malware, we propose a more advanced and comprehensive approach. Our method integrates static and dynamic analysis, offering a detailed view of app behavior, and employs machine learning to classify

Y. Zhang et al. (Eds.): CENet 2023, LNEE 1127, pp. 11–21, 2024.
https://doi.org/10.1007/978-981-99-9247-8_2

apps. This methodology greatly improves the detection of sophisticated Android malware.

By utilizing dynamic debugging, we gain real-time insights into the application's behavior during execution, capturing activities like network interactions, file operations, and system calls. Complementing this, we extract static features from the applications, including certificate information, strings, and permission requests, helping indicate potentially malicious behavior.

Our method's effectiveness has been proven by testing on a large dataset of over 8,000 malicious and benign apps, where it achieved a 96% detection rate, and false positive and false negative rates of 1%. Additionally, we identified "Multi-dimensional features" based on the patterns of permission requests made by malicious applications, improving the malware detection process.

Contribution. To summarize, we have made the following significant contributions:

- **Novel feature**. We innovate the malicious apps detection field by proposing "Multi-dimensional features" based on permission request patterns. This approach substantially boosts the efficiency in detecting and dealing with malicious apps.
- **Systematic Tool**. We present a ground-breaking method that merges dynamic debugging and static feature extraction. This approach enhances the precision and robustness of identifying malicious Android apps, which significantly refines current practices.
- **Experimental analysis**. We validate the efficacy of our novel method via comprehensive testing. The results affirm the method's high precision and consistency, marking a noteworthy advance in combating malicious apps.

2 Background

2.1 Android Platform and Malware

Android, an open-source operating system developed by Google, is broadly used for mobile devices like smartphones and tablets [8]. The flexible and open-source attributes of Android make it a popular choice among both legitimate developers and malicious actors.

Malware encompasses various harmful or intrusive software types, including viruses, worms, Trojan horses, ransomware, and others, specifically designed to target Android devices [5]. They exploit system vulnerabilities, permissions, or user behavior in Android to execute harmful actions, causing issues ranging from annoying disruptions to serious damage such as data loss and privacy violation.

2.2 Machine Learning in Malware Detection

The limitations of traditional detection techniques have prompted the integration of machine learning in malware detection. Machine learning provides an automated way to learn from and make decisions based on data. By training a

model with labeled benign and malicious apps, the system can learn to classify unseen apps effectively.

Machine learning-based approaches typically involve feature extraction and model training, using features derived from both static and dynamic analysis, such as permissions, API calls, network interactions, and system calls. These features are then used to train the model for further classification of apps into benign or malicious categories [10]. However, the model's performance heavily depends on the quality and variety of the features, which makes finding effective features crucial for malware detection.

3 Overview

This section elaborates on our systematic and iterative approach to combating Android malware. We discuss the challenges encountered during feature extraction, malicious application identification, and malware classification, which have simultaneously acted as catalysts for developing innovative solutions. Our strategies stem from an in-depth understanding of the Android platform, complex app behaviors, and the evolving landscape of Android malware.

We detail our multi-stage workflow for detecting and classifying Android malware, from initial app behavior analysis to nuanced identification and classification of threats. We highlight our robust identification mechanism utilizing semantic and user interface recognition and an innovative classification system designed to improve detection efficiency and provide insights into malware behaviors and origins. This comprehensive overview reflects our commitment to a safer Android ecosystem.

3.1 Workflow Overview

In this section, we aim to provide a panoramic view of our comprehensive process, from initial feature extraction to final report generation. This sequential methodology encapsulates the exact steps we undertake to uncover and address the complex challenges Android malware poses. Each component of the workflow contributes to the robustness of our approach, working in synchrony to offer a superior malware detection and classification system, as shown in Fig. 1.

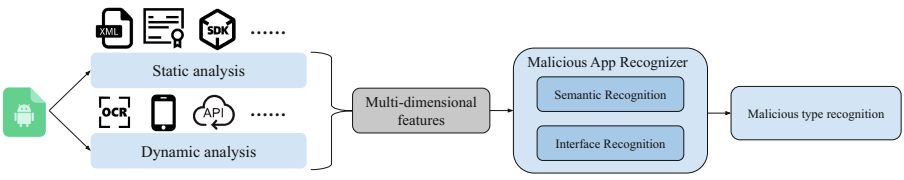

Fig. 1. Overview

Our methodology commences with Static Analysis, where we examine intrinsic properties of Android applications via the application package file (APK) contents, including Android manifest files, bytecode, and other embedded resources.

Permission Extraction. We start with permission extraction, where applications' requests for access to system resources or user data are scrutinized. We employ the AAPT tool to parse Android manifest files and accurately extract permission requests from the APK files, aiding in identifying potential malicious apps.

Certificate Extraction. Certificate extraction forms a key part of our approach, validating the application and its developer's authenticity. We use apktool to decompose APK files and inspect the application's structural details. The analysis of certificates, especially from untrusted sources, can serve as red flags indicating potential security risks.

String Extraction. The extraction of strings from the APK gives insight into the application's purpose and behavior. We use a customized program based on the MobSF framework for this task. Analyzing extracted strings, including URLs, IP addresses, file paths, and hard-coded sensitive information, helps understand the application's network communication patterns and potential malicious behavior.

3.2 Dynamic Analysis

In Dynamic Analysis, we observe applications' real-time behavior in a secure environment. This process reveals potentially harmful actions that might be hidden in the static code. By logging detailed runtime data and cross-referencing it with the static analysis findings, we detect possible malicious activities.

File Operations Monitoring. File Operations Monitoring is a crucial aspect of our dynamic analysis approach. It involves monitoring and analyzing file-related activities during the runtime of applications. By observing the interactions between an application and the file system, we can uncover potentially malicious actions that may not be evident through static code analysis alone.

Function Monitoring. To augment our Android malware detection system, we use the Frida framework for application function monitoring. Frida allows us to inject additional code into the app and monitor its operations, thereby contributing to our understanding of its functionality and possible malicious behavior.

3.3 Malicious App Recognition and Classification

Our system integrates static and dynamic analyses, semantic recognition, and interface identification, with a key role played by our LSTM neural network model. This comprehensive approach enhances malware detection accuracy.

Once a malicious app is detected, our Malicious App Classifier categorizes it based on distinctive features and behaviors, aiding malware detection and countermeasure development. The Malicious Type Recognition component then categorizes the malicious app based on unique patterns and characteristics using an LSTM model.

4 Evaluation

In this section, we provide a comprehensive evaluation of our Android malware detection system, reviewing the static and dynamic analysis stages and the recognition stages for their accuracy and effectiveness.

Dataset. Our evaluation is based on a diverse dataset of 8000 applications sourced from VirusTotal and Google Play Store. This dataset comprises 60% benign and 40% malicious applications from various categories. This diverse selection enables comprehensive testing of our system's performance across a broad range of application behaviors and malware types.

4.1 Static Analysis Evaluation

Static analysis forms the basis of our malware detection process, focusing on extracting key features such as permissions, certificate information, strings, SDKs, and shell fingerprints.

Permission Extraction Evaluation Permissions requested by an app provide insights into its functionalities and potential security risks. In our dataset, we observed a significant difference in the number and type of permissions requested by benign and malicious apps, as depicted in Table 1. Our permission extraction module successfully extracted permissions from 98.2% of the apps, signifying a robust and effective extraction process.

Table 1. Permission analysis

Permission type	Benign apps	Malicious apps
Sensitive permissions	5	11
Normal permissions	7	12
Signature permissions	3	6

SDKs and Shell Fingerprint Recognition Evaluation Our system effectively recognized SDKs in 95.6% of the apps and identified shell fingerprints in 94.3% of the apps, as illustrated in Table 2.

Table 2. Performance of SDKs and shell fingerprint recognition

Feature	Recognition success rate (%)
SDKs	95.6
Shell fingerprints	94.3

4.2 Dynamic Analysis Evaluation

File Operations Monitoring Monitoring file operations during dynamic analysis allows us to track the creation, modification, and deletion of files by the analyzed apps. Our system successfully monitored file operations in 95.8% of the apps, providing comprehensive visibility into their file-related activities.

Upon analyzing the dataset, we observed distinct differences between malicious and benign apps regarding file operations. Malicious apps exhibited a higher frequency of creating and modifying files than benign apps. This behavior raises concerns about the potential for unauthorized data manipulation, stealthy file-based attacks, or attempts to hide malicious payloads within the file system.

To illustrate the findings, consider the following sample comparison of file operations between malicious and benign apps:

App type	Average files created	Average files modified
Malicious	28	17
Benign	9	7

The data indicate that malicious apps tend to create and modify a significantly larger number of files compared to benign apps, as indicated by the higher averages. These findings suggest a higher likelihood of malicious intent or hidden activities within the file system.

System Calls Monitoring System calls monitoring enables us to gain insights into the low-level interactions between the analyzed apps and the underlying operating system. Our dynamic analysis module effectively monitored system calls, achieving a success rate of 97.2%, which allowed us to capture crucial runtime behavior details.

To illustrate the findings, consider the following sample comparison of system call frequencies between malicious and benign apps:

System call	Call frequency (per minute)
Malicious	-
open()	43.2
execve()	12.6
ioctl()	7.9
Benign	-
open()	8.4
execve()	2.1
ioctl()	1.3

Overall, our dynamic analysis module demonstrates its effectiveness in providing valuable insights into network interactions, file operations, and system calls of the analyzed apps. By monitoring these runtime behaviors, we can uncover potential malicious activities and enhance the security of app evaluation and detection processes.

4.3 Malicious App Recognition Evaluation

After the feature extraction phase, the extracted features are fed into our Semantic Recognizer and Interface. The results of the classification process are summarized in Table 3.

Table 3. Performance of semantic and interface recognizers

Recognizer	Accuracy (%)	False positive rate (%)	False negative rate (%)
Semantic recognizer	95.7	4.2	4.3
Interface recognizer	94.8	5.1	5.2

The results in Table 3 confirm the robustness and reliability of our recognizers in classifying the apps. While both recognizers exhibited high accuracy, we observed that the Semantic Recognizer had a slightly higher accuracy than the Interface Recognizer.

4.4 Comparison with Previous AI Method

In the ever-evolving field of Android malware detection, it is crucial to continually evaluate and benchmark new methods against existing ones to ensure their effectiveness and superiority. In line with this, we comprehensively compared our proposed method with a previously established AI method. This comparison aimed to provide a transparent and objective evaluation of our method's performance and its improvements over previous approaches. Figure 2 shows that our

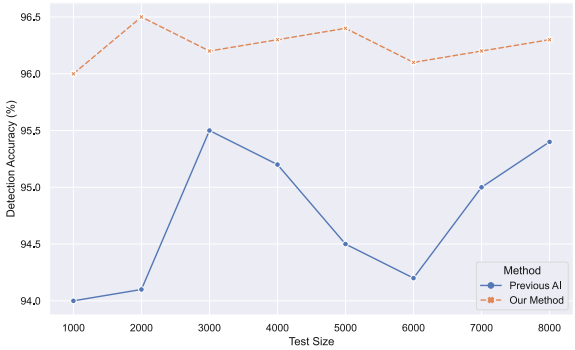

Fig. 2. Comparison of detection accuracy over different test sizes

method consistently outperformed the previous AI method across all test sizes. Despite the increasing complexity and diversity of the test set, our method maintained a high detection accuracy of around 96%, significantly higher than the previous AI method. This result demonstrates the robustness and adaptability of our method, which can effectively handle a wide range of apps and maintain high performance.

4.5 Malicious App Families

Based on our analysis using Multi-dimensional featuress, we have identified six distinct malicious app families. Understanding these families is crucial for developing targeted detection and prevention strategies. It is important to note that these families are determined based on the unique characteristics and behaviors observed in the Multi-dimensional featuress we have gathered. The following table summarizes the identified families and the number of apps associated with each family:

Malicious app family	Number of apps
com.xxx.sty	1204
com.xxx.mhfy	925
com.xxx.didi	280
com.xxx.pronha	175
com.xxx.Wose	365
com.xxx.ransom	155

These families represent different malicious behaviors found with the same developer in mobile apps. We can gain insights into their patterns and techniques by categorizing them, leading to more effective detection and prevention mechanisms.

5 Limitations and Future Work

Limitations. Despite its effectiveness, our system has certain limitations. The ever-evolving Android malware landscape presents new malicious techniques and obfuscation methods that could challenge our analysis methodologies. Dynamic analysis could be resource-intensive, potentially limiting scalability. Also, our classifier's performance relies heavily on the quality and diversity of our training dataset. Insufficient representation of some malware types could reduce the detection rate for those categories.
Future Work. Despite these challenges, there are ample opportunities for future research and development. Developing a real-time system, implementing defenses against adversarial attacks, exploring privacy-preserving data analysis techniques, and creating an automated mechanism for classifier updates to accommodate emerging malware types are among the exciting prospects. By continuously evolving and improving, we aim to remain at the forefront of Android malware detection, contributing to a safer app ecosystem.

6 Related Work

Detection of malicious mobile app. A wealth of methodologies and tools for mobile software analysis have been proposed in previous works, encompassing both static and dynamic analysis approaches [1,12]. These research endeavors have focused on a range of areas such as mobile app analysis, detection of malicious mobile software [6]. Specifically, the studies focusing on the analysis of malicious mobile adware [3,4] have been instrumental in shaping our approach towards analyzing malicious apps of this nature. Our work continues in this trajectory, aiming to further expand the understanding of such harmful applications.
Automation application test. Our investigation draws from these technologies to explore the relatively uncharted territory of hacked mobile software analysis. Key elements of these previous works, such as workflow analysis of mobile software, identification of malicious mobile software, software vulnerability analysis, page layout recognition in software, and the use of automated testing tools, software vulnerability exploration [11]. all contribute to the foundation of our study.

7 Conclusion

In this study, we tackled the significant challenge of Android malware detection. We proposed and implemented an advanced Android malware detection system, combining static and dynamic analysis methods with semantic and interface identification. In addition, we introduced an advanced malicious type identification process, and we proposed the innovative concept of "Multi-dimensional features" based on permission request patterns. Our evaluation results, validated through widely accepted statistical measures such as Recognition Success Rate and False Positive and Negative Rates, demonstrate the robust performance of

our system in distinguishing between benign and malicious applications with high precision and accuracy. Furthermore, our classifier, trained on a diverse and up-to-date dataset, was able to categorize malicious applications into specific classes, enabling targeted response strategies. Finally, we successfully differentiated six malicious app families using the extracted "Multi-dimensional features", demonstrating the practical utility of our proposed approach. This research represents a substantial stride forward in combating Android malware, contributing to a more secure Android app ecosystem and laying a robust foundation for future research and development in this domain.

Acknowledgment. This research is sponsored by the project of State Grid Shandong Electric Power Company Science and Technology Program, Project Name: Research on Key Technologies for Security Analysis of Mobile Applications for eIoT, ERP Number: 520626220019.

References

1. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. Acm Sigplan Notices **49**(6), 259–269 (2014)
2. Chau, M., Reith, R.: Smartphone market share. IDC Corporate USA **444** (2020)
3. Crussell, J., Stevens, R., Chen, H.: Madfraud: Investigating ad fraud in android applications. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, pp. 123–134 (2014)
4. Dong, F., Wang, H., Li, L., Guo, Y., Xu, G., Zhang, S.: How do mobile apps violate the behavioral policy of advertisement libraries? In: Proceedings of the 19th International Workshop on Mobile Computing Systems and Applications, pp. 75–80 (2018)
5. Dunham, K., Hartman, S., Quintans, M., Morales, J.A., Strazzere, T.: Android Malware and Analysis. CRC Press (2014)
6. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android permissions: User attention, comprehension, and behavior. In: Proceedings of the Eighth Symposium on Usable Privacy and Security, pp. 1–14 (2012)
7. freebuf: 2023 global threat report. https://www.freebuf.com/articles/paper/360177.html (2023)
8. Google: Android. https://www.android.com/ (2023)
9. Martinelli, F., Mercaldo, F., Saracino, A., Visaggio, C.A.: I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware. In: 2016 14th Annual Conference on Privacy, Security and Trust (PST), pp. 129–136 (2016)
10. Souri, A., Rahmani, A.M., Jafari Navimipour, N.: Formal verification approaches in the web service composition: A comprehensive analysis of the current challenges for future research. Int. J. Commun. Syst. **31**(17), e3808 (2018)
11. Wang, L., He, R., Wang, H., Xia, P., Li, Y., Wu, L., Zhou, Y., Luo, X., Sui, Y., Guo, Y., et al.: Beyond the virus: A first look at coronavirus-themed mobile malware. arXiv preprint arXiv:2005.14619 (2020)

12. Wei, F., Roy, S., Ou, X., et al.: Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1329–1341. ACM (2014)