

HEURISTIC-BASED MALWARE DETECTION FOR ANDROID USING MACHINE LEARNING

Muhammad Shahzaib Toor
University of Gujrat, Pakistan
shahzaibkhantoor@gmail.com

Received 04 July 2024 Received in Revised form 26 July 2024 Accepted 28 July 2024
Available online 30-July 2024

ABSTRACT

Data security is a serious issue for many organizations nowadays. Android-powered mobile devices are a significant entry point for cybercriminals. Open-source code and the freedom to install applications from third - parties without central monitoring make it easier for attackers to build malware and steal users' private data. Several machine-learning approaches are suggested for identifying malicious software. Here, we use the Botnet Detection dataset to determine whether our apps are safe to use. We use many methods to identify malware. In this research, we do a comprehensive literature analysis on Android malware detection and offer a novel, heuristic-based technique that uses machine learning. The findings from these methods are used to train security professionals to spot dangerous software by using machine learning techniques.

Keyword: Intrusion detection, static analysis, computer forensics, app traffic, machine learning based, heuristic method

1. INTRODUCTION

The number of people who use a smartphone has increased dramatically. That is why it is no surprise that malicious actors are targeting them. Defending against malware is crucial for smartphones since it is so widely used. It is a constant conflict between cyber criminals and security researchers to figure out how to spot malware. The two main characters are keen to embrace new technology. Indeed, this is most true of computer science, which is seeing more use in the anti-malware space. There are a wide variety of Android phones available, from budget options to premium brands. This means more people have access to Android than any other smartphone OS. In April 2020, it is predicted that 70.68 percent of cell phone users will be using Android[1].

Every day, millions of apps are downloaded from the main Android market by a large number of users. Android's free market philosophy means that no apps are reviewed by security professionals, making the platform vulnerable to malware. Targeted assaults on the data contained in and the infrastructures themselves have increased with the proliferation of information technology and Android. [2]. In recent years, Reinforcement Learning (RL) has been acknowledged for its ability in identifying and categorizing Android malware.

While there have been numerous suggestions for Machine Learning (ML) techniques, human ingenuity is still crucial to the development of many useful resources and processes. Particularly, conventional ML-based approaches to Android malware identification depend significantly on manually created characteristics. When it comes to finding malicious apps on Android, [3] classic Machine Learning has excelled. Most ML-based approaches, however, rely largely on the experience, degree of competence, and subject knowledge of security specialists to manually specify the attributes needed to classify Android malware. To stop malware from being distributed via the Android market or from being loaded and deployed, [4] several ML-based ways are widely employed as a security prevention solution.

According to previous studies, there are three distinct types of Android malware detection tools: Static detection,

dynamic detection, and hybrid detection are the three primary kinds of detection [5]. As the name implies, static detection does not need to run the Android to analyze potentially malicious code. It is possible to have a lot of code covered, but it is open to a lot of attacks thanks to things like code obfuscation, and dynamic code loading.

However, dynamic detection requires the code to be executed and the Android app to be analyzed. Dynamic detection may identify hazards that static analysis misses, but it requires a large investment of time and processing resources.

To strike a happy medium between detection efficacy and efficiency, researchers have developed a method called hybrid detection, which employs both stationary and non-stationary detection methods.[6] Approaches to identifying Android malware using machine learning have been described in the literature. That said, there are several notable outliers. The study's limited breadth, the lack of discussion on several sensitive subjects, and the reliance on out-of-date material in the prior evaluations.

This research proposes a novel method—a heuristic approach—to fix these problems. Decompiling suspicious software and inspecting its source code is at the heart of this method. We can improve our performance by mixing subsets of dynamic features. New dangerous programs that evade the letter and rule-based detection approaches are the primary focus of heuristic detection methods. To build heuristic malware detectors, we use Data Mining techniques. Both signature-based and behavioral tendencies are accounted for.

Data preparation, image segmentation, machine learning techniques, algorithms, and detector effectiveness assessment are all carried out in Section IV once the study's limitations and shortcomings have been outlined. In Section V, we propose some questions for and avenues of investigation into the future. Section VI is where we draw the initial findings.

II. LITERATURE REVIEW

Malware detection often uses signature-based approaches [7] first developed in the mid-1990s. A major flaw of these methods is that they cannot identify malware that is constantly evolving and is not being actively monitored. Instead of relying on static signatures, statistical and machine-learning approaches may dynamically retrieve malware patterns. One previous study [5] took advantage of data mining & features obtained from Windows binary API calls. Their performance was remarkable on a large dataset consisting of over 35,000 movable executable files.

A further technique of behavioral foot printing offers a dynamic strategy for identifying self-propagating malware.

More sophisticated malware assaults, such as repackaging, have been seen on iPhone mobile computing systems in recent years. The methods of installation, mechanisms of activation, and types of dangerous payloads carried by current Android malware are all carefully characterized in recent research by Zhou et al. [8]. Their tests expose the limitations of existing malware detection systems and show the need for other anti-mobile-malware solutions by comparing four typical mobile security products to more than 1200 gathered malware.

Some research [9] aims to identify malware by examining the metrics and/or dynamic behavior and features of apps, which is motivated by the increasing variety of Apps and the lack of efficient malware detection technologies. Zhou et al. [1] suggest heuristic filtering and the use of permission behavior to identify new Android malware. Droid Ranger is a hybrid approach that gets around the problem of undetectable spyware. Although the various approaches all improve Android malware identification. The SVM algorithm is not flexible enough to deal with new forms of Android malware and constantly requires regular signature changes.

III. ANDROID APPLICATION STRUCTURE

Here, we will go through the fundamentals of how an Android app is organized, with an emphasis on the most crucial files. Our system can extract important similarities for malware identification from this description of Android-based mobile platform Apps. The basic layout is as follows:

Android Package Kit abbreviates to "APK" The source code for an Android app (stored in .desk files), as well as all required resources, media, and the manifest file, are compiled and boxed into a single file during the packaging process. Any name up to and including. A PK is acceptable for the application package file. A crucial part of every Android project is the AndroidManifest.xml file.

It has all you need to know about the App in it. When an app is opened on an Android device, the operating system initially searches for the associated Manifest file. It is a guide for making sure all your apps are compatible with Android. If a resource, permission, or functionality is not explicitly included in an App's Android Manifest file, the Galaxy Nexus will not allow it. Thus, the AndroidManifest.xml file is a primary source for learning about the Apps' features and protections.

A.Android Security Approach

Android's security model is heavily reliant on permission-based mechanisms. There are about 130 permissions that govern access to various resources. To function, an Android application requires several permissions. As a result, granting all permissions requested by the application is an essential step in installing an Android application on a mobile device. Before installing an application, the system prompts the user for a list of permissions requested by the application and asks the user to confirm the installation settings. Although permission requests are useful for users to prevent Apps from misusing resources, users frequently lack the knowledge to determine whether permissions are harmful or not. For example, while requesting network access, including Wi-Fi and short message service (SMS), is fairly normal for generic Apps, some malware abuses the services to steal bandwidth or other useful information. As a result, it is extremely difficult for users to determine whether an App is malware based solely on the permission request. At the system level, Google announced that each application uploaded to their market is subject to a security checking mechanism. The Android operating system's open design allows users to install any application downloaded from an untrusted source. Nonetheless, the permission list remains the user's most basic line of defense in determining whether an application is potentially harmful. As a result, users can choose not to install an App if the App requests permission to the user's contacts unnecessarily (e.g., phone book).

Google divides Android permissions into four threat levels:

Normal permissions: These are less critical permissions that restrict access to less harmful API methods. Similar to RESTART PACKAGES, the system automatically provides this permission to a querying application during setup. Permissions that give access to sensitive user information should be granted only when necessary.

Permissions such as ACCESS NETWORK STATE and READ PHONE STATE, which might reveal a user's physical location, are considered to be very risky. Signing off on access to one of the most sensitive authorities is essential. The system will only provide access if the seeking app uses the same certificates as the app that first claimed the privilege.

Signature/System permission: A privilege only for Android system apps are given by the operating system. While checking whether or not an App asks that Android utilize an approved permission model to manage access to its features is a simple technique to identify malicious code, no clear data shows how successful it is at identifying malicious code.

Android Permission Setting:

An Android Criteria file may be found in the main folder of every APK. Manifest.xml provides the Android operating system and the user with crucial information about the application. Before the Android system can execute any of the application's code, it must acquire and process this data. The following are examples of what the manifest file accomplishes that are directly related to the App's behavior and security settings. Access to restricted APIs and inter-app communication are both outlined in the application's manifest file. It also specifies the privileges other users must possess to engage with the app's features. If you want to do experiments based on these rights, you will have to obtain an AndroidManifest.xml record and then reverse-engineer it using the APK tool [9]. Once you have the AndroidManifest.xml, you can use it to determine what rights each APK needs by reading the file and parsing it. Using a comparison of the most requested permissions by malicious and benign apps in the dataset, we show that permissions settings are important in shaping the behavior of both. We hope that by integrating Android application programming interface (API) calls and permissions, we can improve detection accuracy.

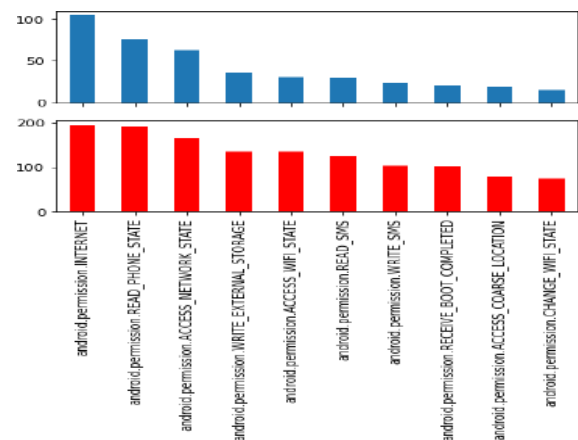


Fig 1: topmost requested permission by malware and benign application

Android API Calls

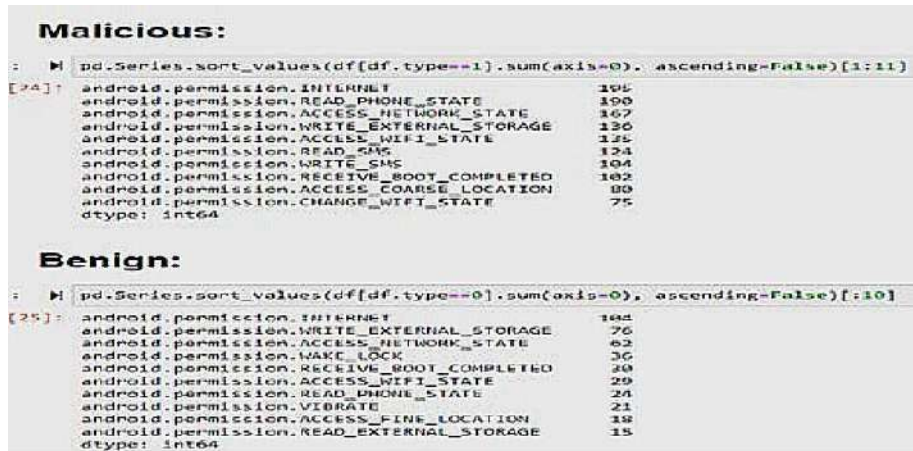


Fig 2: top 20 API calls permissions

For Apps to communicate with Android itself, the Android platform offers a framework API. There is a minimum set of packages and classes that make up the framework's API. Given the prevalence of API use across most apps, we are driven to utilize API calls distinguishing between malicious and safe software. As a means of doing this, it is possible to build a framework for deciphering APKs and obtaining API calls for each program. Once that is done, the same method as the authorized representation up top may be used. We detail the top 20 API calls made by malicious and benign apps to show how useful they are for identifying malware. These are common requests made by malicious and safe apps alike. The first two usually require the embedded libraries to operate correctly. However, dangerous applications commonly seek access to your phone's SMS settings. Furthermore, we see that harmful apps often ask for greater access than good ones.

The mean number of permissions both dangerous and benign apps in our sample ask for. These findings corroborate previous research showing a striking distinction between the permission use patterns of malware and those of innocuous programs.

IV.METHODOLOGY

For Android malware detection, we present a Heuristic-based learning system here. The Android framework takes into account permissions and API requests to describe app activities.

Figure 3 depicts the four main components of the proposed framework. The first part is an application analyzer, which takes an application's compressed APK file and unpacks it, revealing the Accommodate and class files required for app classification.

The second part of the plan is to categorize apps by the permissions and APIs they use. Features such as permission features taken from Android Manifest as API call features retrieved from class files are generated by the completed system, the feature generator. A class label designates whether an App is safe or malicious, and the process represents each App as a unique instance with binary permission and API call characteristics (Detailed information about benchmark data is reported in Section V). Lastly, categorization models are trained using the amassed data. The framework's transformation of Apps into a general instance feature format makes it possible to create classification algorithms from the training data using almost any learning method. In our studies, we take a close look at several different categorization strategies, including Designing Trees, K-mean, Naive Bayes, and Random Forests.



Fig 3: The proposed permission and API call feature-based malware detection framework.

V. DATA COLLECTION:

We have collected a total of 2510 APK files for use as a benchmark, 1260 of which are malicious Apps verified in prior research [10], and the leftover 1250 are clean Android APKs. We discovered this by comparing feature values and discovering that a large percentage of malware shares the same ones (this often happens for malware belonging to the same family, such as a different version of one malware application). In this way, we may reduce the size of the dataset by keeping just one instance of malware that has the same vector properties. This procedure yields 610 malware samples from 49 malware families, showing a comprehensive coverage of the current Android threat landscape. A total of 720 samples of malicious software are produced.

VI. CLASSIFICATION METHODS

Through the use of several classifiers (decision trees, K-means, Naive Bayes, and Random Forest), we evaluate the efficacy of the approach. The goal is to find out whether it is possible to get more insight into describing App behavior by blending permission and API requests. Moreover, we want to identify the top classification approach that is particularly well-suited to malware detection. Training samples are selected at random from the initial training set and then replaced with new ones to create the training set for each of the basic classifiers. The vote total of all classification models determines the ultimate verdict for a given instance.

Table 1: The results of precision, recall, accuracy, and AUC between different classifiers.

Data Set	Classifier	Accuracy	Precision	Recall	AUC
Perm	Random Forest	92.9	0.92	0.94	0.91
API	k-nearest neighbors	98.8	0.98	0.90	0.98
Com+	Decision Tree	95.75	91.7	95.7	0.957
Perm	naive Bayes	94.9	94.1	92.8	0.986

VII. CONCLUSION

In this work, we suggested a method for identifying malware on Android by analyzing the permissions of the API calls of installed programs. The proposed framework mines Android apps for permissions and then uses a combination of API calls to create a comprehensive profile of each app in the form of a fast feature vector. Classification models for determining if an App is safe or malicious may be created by applying learning techniques to the amassed datasets. The framework's efficacy in detecting malware has been shown experimentally using real-world data. The suggested framework adds three things above the current methods: First, we show that combining authorization and API calls improves malware detection. Second, our framework does not require a dynamic tracing of the Android applications. Finally, our framework is generalizable to all mobile applications.

REFERENCES

- [1].Statcounter. (Mar. 2020). Mobile Operating System Market Share Worldwide.', IEEE online Accessed: April. 16, 2021. [Online]. Available: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
- [2].Chang, YS., Kuo, SP., Huang, CH. (2013). Design and Implementation of a Software Development Kit for LLRP Readers. In: Chang, RS., Jain, L., Peng, SL. (eds) Advances in Intelligent Systems and Applications -Volume 1. Smart Innovation, Systems and Technologies, vol 20. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35452-6_68
- [3].R. Daniel Arp, "Effective and explainable detection of Android malware in your pocket.," In Proceedings of the 21st Network and Distributed System, 2014.

[4].Ziyun Zhu, Tudor Dumitras, " Automatically engineering features for malware detection by mining the security literature," Conference, 2016. CCS'16, October 24 - 28, 2016, Vienna,Austria,[http:// dx.doi.org/10.1145/2976749.2978304](http://dx.doi.org/10.1145/2976749.2978304)

[5]. Sihan Qing, "Research Progress on Android Security," Journal of Software,27(2016)45-71.

[6].Milad Taleby Ahvanooy, Qianmu Li, Mahdi Rabbani, Ahmed Raza Rajput, "A survey on smartphones security: Software vulnerabilities, malware, and attacks," (IJACSA) International Journal of Advanced Computer Science and Applications,8(2017)30-45

[7].Salzberg, S.L. C4.5: Programs for Machine Learning by Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993. *Mach Learn* **16**, (1994)235–240,[https://doi.org/10.1007/ BF00993309](https://doi.org/10.1007/BF00993309)

[8].Zhou Y, Jiang X (2012) Dissecting android malware: Characterization and evolution. In: Proc. of IEEE Symposium on Security and Privacy,-pp95–109

[9].J. Wang, P. Deng, Y. Fan, L. Jaw, and Y. Liu,Virus detection using data mining techniques. In Proceedings of IEEE International Conference on Data Mining,2019.

[10] Min Zhaos, Tao Zhang, Fangbin Ge, Zhijian Yuan,'RobotDroid: A Lightweight Malware Detection Framework On Smartphones', Journal of Networks, 7(2021)