Bypassing Drebin: Exploring Techniques to Evade Android Malware Detection

1st Guy Azoulay
Bachelor of Computer Science
Ariel University
Rosh Hayin, Israel
Guy1azu@gmail.com

2nd Tal Malchi
Bachelor of Computer Science
Ariel University
Petah Tikva, Israel
malchital@gmail.com

***Abstract*—The Drebin classifier is a commonly used tool for identifying malicious Android applications. Despite its widespread use, it has been shown to be vulnerable to certain types of attacks. In this article, we present a variety of methods for attempting to compromise the Drebin classifier's classification abilities. Although we were not successful in significantly impacting its performance, the methods presented in this paper offer new perspectives on the limitations of the Drebin classifier and the need for more robust techniques for identifying malicious mobile applications.**

I. Introduction

In today's digital world, mobile devices and applications are increasingly vulnerable to cyber-attacks. One of the key challenges in mobile security is the ability to identify and classify malicious applications, which can steal personal information, perform unauthorized actions, or even cause damage to the device or network. However, attackers are constantly developing new techniques to evade detection by traditional classifiers, making it increasingly difficult to protect mobile devices and applications from malicious attacks. Static classifiers, in particular, have been shown to be particularly vulnerable to these types of attacks, as they rely on the analysis of the application's code and resources, which can be easily modified by an attacker.

In this work, we aim to evaluate the effectiveness of various methods for compromising the classification abilities of the Drebin classifier, a widely used static classifier for detecting malicious Android applications. Our study includes testing different methods on a dataset of 10,000 applications, with 80% of the apps being benign and 20% being malicious. This dataset serves as the training group. We also used a test group of 1,000 applications, with 30% being malicious and 70% being benign.

The Drebin classifier is a very stable and reliable classifier, and its functionality is extremely complex [1]. Despite these difficulties, we attempted to compromise its classification abilities through several methods. Although we succeeded in lowering the Entropy of malicious apps, the Drebin classifier was still able to detect them successfully. Our findings shed light on the limitations of current static classifiers and the need for more robust techniques for identifying malicious mobile applications.

II. Related Work

The detection and analysis of Android malware has been a popular area of research in recent years, due to the increasing number and sophistication of malicious applications. Many different concepts and techniques have been proposed to counter the threat of malware on the Android platform.

One approach for detecting Android malware is through the use of static analysis. This involves inspecting the code and resources of an application without executing it. This type of analysis is based on static program analysis, where several methods have been proposed that disassemble applications and check for indications of malicious activity. For example, the method Kirin checks the permission of applications, while Stowaway analyzes API calls to detect overprivileged applications.

Another approach is through the use of dynamic analysis, which involves monitoring the behavior of an application while it is running. This type of analysis is mainly applied for offline detection of malware, such as scanning and analyzing large collections of Android applications. However, dynamic analysis is technically too involved to be deployed on smartphones and detect malicious software directly.

Finally, machine learning has been proposed as a solution to the difficulty of manually crafting and updating detection patterns for Android malware. Several methods have been proposed that analyze applications automatically using learning methods. For example, the method of Peng et al. applies probabilistic learning methods to the permissions of applications for detecting malware.

In this research, we aimed to evaluate the robustness of the Drebin classifier against various types of attacks which we will explain later in this article. Our approach includes attempting to compromise the classification abilities of the Drebin classifier through several methods. We started by using obfuscation techniques using a tool called "Obfuscapk [2]", which is a well-known obfuscation tool for Android apps. We also experimented with different packers tools such as ProGuard [3] and DashO [4], which are used to shrink, optimize and obfuscate code. Additionally, we tried to

change the feature set of malicious apps by adding benign features to those apps, and we also tried to change the feature set and then obfuscate or pack the app. Our goal was to test the resilience of the Drebin classifier against a range of attacks by utilizing various methods.

III. METHODOLOGY

In this research, we used a machine learning classifier, the Drebin classifier, to evaluate its robustness against various types of attacks. The classifier is trained using a dataset of labeled applications, where the labels indicate whether an application is malicious or benign. The dataset was provided by our teacher Dr. Harel Berger and contains around 60GB of benign apps and 6GB of malicious apps in it. This dataset was used as the training group to "teach" the classifier how to classify Android files.

The features set of the Drebin classifier includes several types of information that are extracted from the applications, such as permissions, network addresses, and API calls. These features are used by the classifier to make its classification decisions. As an attacker, we used this classifier and tried to harm its classification abilities by using different methods such as Obfuscation, Packing, and Problem Attack.

In our methodology, we used a test group and applied the different methods on the it and evaluated the performance of the classifier on these modified apps and compared it to the original apps. The methods we used as we mentioned earlier are: Obfuscation, Packing, and Problem Attack.

- Firstly, for the Obfuscation method, we used a tool called "Obfuscapk" to obfuscate the applications and make it difficult for the classifier to extract the features.
- Secondly, for the Packing method, we used different packers tools such as ProGuard and DashO. These tools are used to shrink, optimize and obfuscate code, which makes it difficult for the classifier to extract the features.
- Lastly, for the Problem Attack method, we tried to change the feature set of malicious apps by adding benign features to those apps and also tried to change the feature set and then obfuscate or pack the app. For each of these methods, we will provide a detailed explanation of the techniques used, the results obtained, and the implications of these results on the robustness of the Drebin classifier.

A. OBFUSCATION

Obfuscation is a technique used to make the code of an application difficult to understand and reverse-engineer. Obfuscation can be used to protect intellectual property, hide the workings of a program, or make it more difficult for attackers to discover vulnerabilities in the code. In this research, we used the "Obfuscapk" tool to apply obfuscation to the applications in our dataset. The "Obfuscapk" tool is a well-known tool for obfuscating Android applications. It can be used to apply various obfuscation techniques such as renaming classes, methods, and fields, and removing debugging information. The "Obfuscapk" tool also supports the use of custom obfuscation plugins, which allows for a wide range of obfuscation techniques to be applied.

One way of achieving Obfuscation is by changing the data strings in the application. This is done by encrypting, renaming, or removing certain data strings in the app. By doing this, the attacker is trying to make it harder for the classifier to detect the malicious application and classify it as a benign application.

During the Obfuscation process, the entropy change. Entropy is a measure of randomness or disorder in a data set. In the context of mobile app classification, entropy can be used as a feature to differentiate between benign and malicious apps. This is why showing the difference between the entropy of benign and malicious applications is important. It helps us understand the robustness of the classifier against various types of attacks, including obfuscation.

In our study, we used the "Obfuscapk" tool to apply obfuscation to the applications in our dataset. We used both the common features of the tool as well as advanced features. By applying advanced features, it makes it more difficult for the classifier to extract the features of the applications. After applying obfuscation, we evaluated the performance of the Drebin classifier on the obfuscated applications. In Fig 1 we can notice the process this tool made on our data.

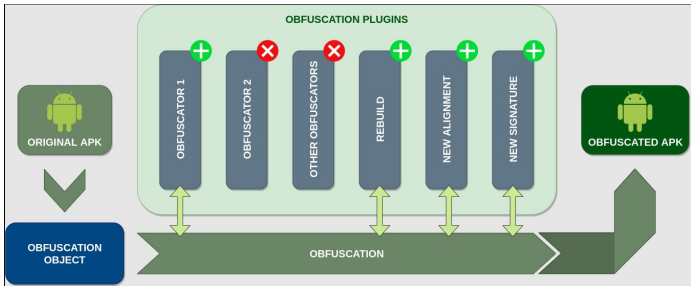

Fig. 1. How the "Obfuscapk" works in general.

B. PACKERS

Packing is another technique used to make the code of an application difficult to understand and reverse-engineer. In this research, we used two popular packers tools: DashO and ProGuard. DashO is a commercial tool that provides advanced obfuscation and code shrinking features for Android and Java applications. It also provides protection against reverse engineering and tampering. ProGuard is a free, open-source tool that also provides obfuscation and code shrinking features for Java applications. It is commonly used with the Android SDK to shrink, optimize and obfuscate code.

In our study, we used both DashO and ProGuard to pack the applications in our dataset. By using these tools, we aimed to make it difficult for the classifier to extract the features of the applications and hence compromise its classification abilities.

C. PROBLEM ATTACK

Lastly, we employed a two-step approach. First, we used the apktool, an open-source tool that allows for the decompiling and compilation of Android applications, to extract the features of the malicious apps. Second, we used a script we wrote to manipulate the extracted features in order to change the feature set of the malicious applications. The goal of this feature-set attack was to add benign features to the malicious apps in order to make it difficult for the classifier to identify them as malicious. This approach simulates the scenario where an attacker tries to evade detection by adding benign functionalities to the malware. In addition, we also tried to combine this method with the obfuscation method (3.1) using obfuscapk and hoped for better results in terms of compromising the classification abilities. In Fig 2 you can see the features-set attack scheme, and in Fig 3 you can see and features we injected into every malicious application.

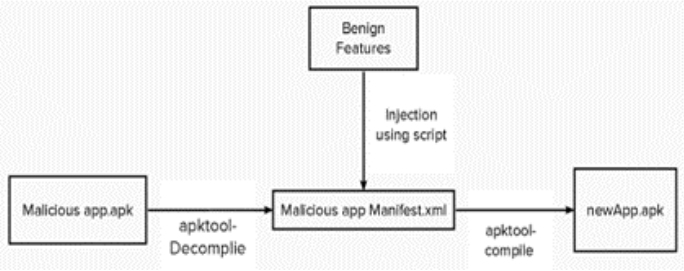

Fig. 2. Problem attack process.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<app_permissions android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<app_permissions android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<app_permissions android:name="android.permission.READ_PHONE_STATE" />
<app_permissions android:name="android.permission.ACCESS_FINE_LOCATION" />
<app_permissions android:name="android.permission.DISABLE_KEYGUARD" />
<app_permissions android:name="android.permission.WAKE_LOCK" />
<app_permissions android:name="android.permission.INTERNET" />
<app_permissions android:name="android.permission.VIBRATE" />
<api_permissions android:name="android.permission.VIBRATE" />
<api_permissions android:name="android.permission.INTERNET" />
<api_permissions android:name="android.permission.READ_CONTACTS" />
<api_permissions android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Fig. 3. Benign features we injected to the malicious application.

IV. RESULTS

In this research, we aimed to evaluate the robustness of the Drebin classifier against various types of attacks. We employed a variety of methods such as Obfuscation, Packing, and Problem Attack, and used a combination of tools and techniques to compromise the classification abilities of the Drebin classifier. Despite our efforts, we were unable to significantly harm the classification abilities of the Drebin classifier.

In Figure 4, we can see a representation of the results of our experiment for the obfuscation. Where we manually added 18 benign applications, represented by green dots on the graph, and 18 malicious applications, represented by red dots. These malicious applications have undergone an obfuscation process. The Y-axis of the graph represents the Entropy of each application. It's important to note that, the yellow bullets on the graph represent malicious applications that after the obfuscation process, their entropy level has remained unchanged or only experienced minor changes.

The following table illustrates the outcome of the amalgamation between problem attack and obfuscation techniques. We conducted an experiment utilizing a smaller dataset consisting of 30 applications, with 21 being benign and 9 being malicious. As depicted in the table, with a small dataset, we were able to effectively impede the classifier's capability to accurately classify malicious applications.

Malware	Accuracy	recall	f1-score
Before	0.923	0.83	0.86
After	0.893	0.75	0.82

For the bigger dataset, our results indicate that the Drebin classifier might be stable against the types of attacks that we evaluated. The classifier was able to accurately classify most of the applications in our dataset, even when they were obfuscated, packed, or had their feature sets changed.

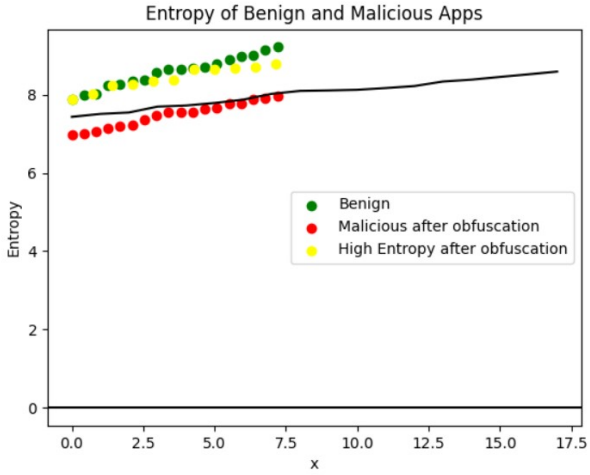

Fig. 4. Application Entropy comparison

V. CONCLUSION

Drebin is a well-known static Android malware classifier that uses machine-learning techniques to identify malicious apps. It has been widely used in the Android security research community as a benchmark for evaluating the performance of different malware detection methods. Our goal in this research was to thoroughly examine the Drebin classifier and identify its weak spots. Our objective was to exploit these weaknesses in order to understand the limitations of the classifier.

In this research, we discovered the weak spots of the Drebin classifier - its reliance on static analysis only, its limitation to

known malware, and its tendency to generate false positives. We attempted to exploit these weaknesses by attacking the classifier, as detailed in this research. Though we may not have fully succeeded in our mission, we were motivated to continue trying and are hopeful that in the future, we will achieve our goal. Through this research, we gained valuable insights and knowledge, by experimenting with a variety of tools, thinking creatively and out of the box, and ultimately, learning a lot in the process.

REFERENCES

[1] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[2] Obfuscap. Tool employees from france called us and explain step by step about this tool.

[3] ProGuard. a free tool that we use as another way for packing files.

[4] DashO tool. Tool employees from france called us and explain step by step about this tool.