When the Guard failed the Droid: A case study of Android malware

Rawan Shareef computer science student ,Andrey Fridman computer science student

***Abstract*—Android malware analysis is the process of identifying and studying malicious software that targets the Android operating system. The goal of this analysis is to understand the functionality and behavior of the malware, as well as to identify any potential vulnerabilities or weaknesses that could be exploited by the malware. To study malwares behavior we used Kirin classifier and permissions extraction. Using the results we got we improved Kirins classification.**

I. Introduction

Android malware is a common problem that speaks of an ongoing threat to billions of users around the world and there have been many cases where security measures such as "Guardians" or antivirus software have failed to protect devices from malware, so Android malware detection systems are often vulnerable to evasion attacks, in which an adversary manipulates malicious cases so that they are mistakenly identified as benign.

In this paper, various novel evasion attacks are launched against several Android malware detection systems in these detection systems inherent vulnerability is that they are part of Androguard [1]. Some of the detection systems drop to a 0

The paper presents a new assessment scheme for creating evasive attacks that exploit vulnerabilities in known Android malware detection systems. When evaluating the functionality and damage of the executed instances generated by our evasion attacks, there are differences in both the malicious and functional tests of our compromised applications, thus showing that non-functional applications do not threaten users and are therefore useless from the attacker's point of view. Thus, we conclude that evasion attacks must be evaluated both in terms of function and in terms of damage, in order to assess their impact.

II. Related Works

The article reviews well-known ML-based detection systems for Android malware so that there are three main approaches, and also describes some popular evasion attacks targeting the detection systems. We discuss three forms of attack, then describe several permission-based detection systems for Android malware. Since this article presents several vectors for evasion attacks targeting Android privileges, we discuss recent work dealing with these features. Finally, we examine the functionality and maliciousness tests of Android malware. To the best of our knowledge, these parameters have not been fully explored in previous works in the field of Android malware. Android Malware ML-Based Detection Systems Like Drebin collects syntactic features, and collects eight types of features from the APKs also have the DroidAPIMiner similar to Drebin, in that API calls and permissions are checked. In addition there is DroidMat where other features from the AndroidManifest XML file were tested, there is also MalPat which analyzes the use of API calls in relation to app permissions. Evasion attacks against ML-based detection systems can take multiple courses,a typical approach to evasion attacks on ML-based detection systems includes adding noise to the app, thus misleading the classifier's assignment of benign and malicious app. An example of the use of this approach can be found in Android HIV where the authors implemented non-invoked dangerous functions against rebin and a function injection against MaMaDroid another option is to use Generative Adversarial Networks (GAN) to add noises to the classification process,stub function4 can also be used to break the app flow,In addition to this we are improving our attacks with a special focus on the Manifest file, thus implementing new attack methods such as Manifest Pockets and other Manifest based attacks.

Android integrates system permission requests from the user to obtain system resources as a permission tag used in a Manifest file that contains a fixed string name of permission. The permission requests are considered simple features for Android malware detection systems and we discuss Android's well-known permission-based malware detection system called Kirin, this rules-based security system for Android checks an app's Android permission usage against a list of rules at installation time. So the authors Evaluated their system against 311 apps from the Android market, where 10 apps raised alerts. Five of the suspect apps claimed a dangerous configuration of permissions, but were used within reasonable functional needs based on application descriptions. The other 5 apps were classified by the authors as malicious and need to be addressed when installed. Therefore, they concluded that their technique requires user involvement in about 1.6

In our article the data set we used has a total of 9000 benign apps and 1000 malicious apps, with a ratio of 9:1 as it exists in the real world. The 1000 malicious apps were divided into 5 groups.

DREBIN is an Android malware detection method that allows you to detect malicious applications directly on the smartphone, and it was created because the amount of malicious applications poses a threat to the security of the Android platform because the increasing number of these applications renders conventional defenses largely ineffective, making it difficult for Android smartphones to protect themselves against new malware.

MaMaDroid is a static analysis-based system that simplifies the API calls made by an application to its class, package, or family, and builds a model of their sequences obtained from an application's call graph as a Markov network. This ensures that the model is more resilient to API changes and that the feature set is a manageable size.

Andromaly is a framework for detecting malware on Android mobile devices by implementing a host-based malware detection system that continuously monitors various features and events received from the mobile device, and then applies a machine learning anomaly detector to classify the collected data as benign or malicious And because Android malware apps are not available then we developed 4 malware apps, and evaluated Andromaly's ability to detect new malware based on known malware samples.

III. Methodology

Kirin is a rule based detection system that uses app permissions requests as its only feature. So to improve kirins classifications we first of all need to take a data set of benign and malicious apps, extract their permissions and look what we can find there. Kirin uses a set of groups of specific permissions so if a program asks for this group of permissions from android, that means it is probably a malicious app.

The data set we used has a total of 9000 benign apps and 1000 malicious apps, with a ratio of 9:1 as it exists in the real world. The 1000 malicious apps were divided into 5 groups. Each of these 5 groups was attacked with 6 different attacks: MB1, MB2, MB3, MB4, SB and a combination of these attacks. After kirins classification we got a csv file that had a table with all the groups we had, each one was divided into 6 different rows, each row represented a group and the attacks that were used on this group. In each row we also got a recall that was calculated with the confusion matrix that we also got from kirin classification.

To improve kirins classification we decided to add a combination of new permissions to the list of rules that kirins use. To know what to add we first of all extracted the permissions from our database. After that we checked what is the most popular permission that was used by the malicious apps that wasn't in the original kirins list of rules. We found that some of the most popular (maybe not the most) are WAKE LOCK and ACCESS NETWORK STATE.

We decided to combine ACCESS NETWORK STATE with INTERNET because a malicious program can access the network state and read personal data from there and send it via the internet. Also we decided to combine WAKE LOCK with INTERNET because a malicious program can keep the phone's screen from dimming and make a screenshot or screen record, which doesn't require permission, and send it via the internet. For example, a person can put his phone into his pocket but it wont turn off after some time and the malicious program will make screenshots and send it.

IV. Results

First of all we ran the Kirin classifier without any changes to the list of rules. The average recall that we got is 0.43. The accuracy here is 301/680 = 0.44. The precision is 1. The F1 score is 2*((1 * 0.43) / (1 + 0.43)) = 0.6. The F1 score is a measure of a model's accuracy that combines precision and recall. (Table 1).

The second run of the kirin classifier wasn't default. We added to the list of rules 2 permissions, that are WAKE LOCK and INTERNET, and we got the average recall that was 0.63. The accuracy here is 397/680 = 0.58. The precision is 1. The F1 score is 2*((1 * 0.58) / (1 + 0.58)) = 0.73. (Table 2).

On the third run we added ACCESS NETWORK STATE and INTERNET permissions instead of the previous 2. The average recall was 0.79. The accuracy here is 539/680 = 0.8. The precision is 1. The F1 score is 2*((1 * 0.79) / (1 + 0.79)) = 0.88. (Table 3).

We tried to run the Kirin classifier with READ EXTERNAL STORAGE and INTERNET but the average recall was the same as the original one so we can understand that this permission does not help us. The accuracy here is the same as in the default kirin run 0.44. The precision is 1. The F1 score is 2*((1 * 0.43) / (1 + 0.43)) = 0.6. (Table 4).

group	attack	test_recall	test_confusion_matri
0	1	0.38	[[0 0] [13 8]]
0	2	0.36	[[0 0] [14 8]]
0	3	0.36	[[0 0] [14 8]]
0	4	0.36	[[0 0] [14 8]]
0	5	0.38	[[0 0] [13 8]]
0	6	0.33	[[0 0] [14 7]]
1	1	0.43	[[0 0] [13 10]]
1	2	0.43	[[0 0] [13 10]]
1	3	0.43	[[0 0] [13 10]]
1	4	0.43	[[0 0] [13 10]]
1	5	0.43	[[0 0] [13 10]]
1	6	0.43	[[0 0] [13 10]]
2	1	0.5	[[0 0] [12 12]]
2	2	0.5	[[0 0] [12 12]]
2	3	0.5	[[0 0] [12 12]]
2	4	0.5	[[0 0] [12 12]]
2	5	0.5	[[0 0] [12 12]]
2	6	0.5	[[0 0] [12 12]]
3	1	0.5	[[0 0] [11 11]]
3	2	0.48	[[0 0] [12 11]]
3	3	0.48	[[0 0] [12 11]]
3	4	0.5	[[0 0] [11 11]]
3	5	0.5	[[0 0] [11 11]]
3	6	0.48	[[0 0] [12 11]]
4	1	0.38	[[0 0] [13 8]]
4	2	0.43	[[0 0] [13 10]]
4	3	0.43	[[0 0] [13 10]]
4	4	0.43	[[0 0] [13 10]]

Fig. 1. Table 1.

V. Conclusion

In this work, we proposed some extensions and improvements that can be added to the Kirin classifier to improve its

group	attack	test_recall	test_confusion_matri
0	1	0.67	[[0 0] [7 14]]
0	2	0.64	[[0 0] [8 14]]
0	3	0.64	[[0 0] [8 14]]
0	4	0.64	[[0 0] [8 14]]
0	5	0.67	[[0 0] [7 14]]
0	6	0.62	[[0 0] [8 13]]
1	1	0.52	[[0 0] [11 12]]
1	2	0.52	[[0 0] [11 12]]
1	3	0.52	[[0 0] [11 12]]
1	4	0.52	[[0 0] [11 12]]
1	5	0.52	[[0 0] [11 12]]
1	6	0.52	[[0 0] [11 12]]
2	1	0.62	[[0 0] [9 15]]
2	2	0.62	[[0 0] [9 15]]
2	3	0.62	[[0 0] [9 15]]
2	4	0.62	[[0 0] [9 15]]
2	5	0.62	[[0 0] [9 15]]
2	6	0.62	[[0 0] [9 15]]
3	1	0.73	[[0 0] [6 16]]
3	2	0.7	[[0 0] [7 16]]
3	3	0.7	[[0 0] [7 16]]
3	4	0.73	[[0 0] [6 16]]
3	5	0.73	[[0 0] [6 16]]
3	6	0.7	[[0 0] [7 16]]
4	1	0.62	[[0 0] [8 13]]
4	2	0.65	[[0 0] [8 15]]
4	3	0.65	[[0 0] [8 15]]
4	4	0.65	[[0 0] [8 15]]
4	5	0.62	[[0 0] [8 13]]
4	6	0.65	[[0 0] [8 15]]

(a) Table 1

group	attack	test_recall	test_confusion_matri
0	1	0.67	[[0 0] [7 14]]
0	2	0.68	[[0 0] [7 15]]
0	3	0.68	[[0 0] [7 15]]
0	4	0.68	[[0 0] [7 15]]
0	5	0.67	[[0 0] [7 14]]
0	6	0.67	[[0 0] [7 14]]
1	1	0.78	[[0 0] [5 18]]
1	2	0.78	[[0 0] [5 18]]
1	3	0.78	[[0 0] [5 18]]
1	4	0.78	[[0 0] [5 18]]
1	5	0.78	[[0 0] [5 18]]
1	6	0.78	[[0 0] [5 18]]
2	1	0.88	[[0 0] [3 21]]
2	2	0.88	[[0 0] [3 21]]
2	3	0.88	[[0 0] [3 21]]
2	4	0.88	[[0 0] [3 21]]
2	5	0.88	[[0 0] [3 21]]
2	6	0.88	[[0 0] [3 21]]
3	1	0.91	[[0 0] [2 20]]
3	2	0.87	[[0 0] [3 20]]
3	3	0.87	[[0 0] [3 20]]
3	4	0.91	[[0 0] [2 20]]
3	5	0.91	[[0 0] [2 20]]
3	6	0.87	[[0 0] [3 20]]
4	1	0.71	[[0 0] [6 15]]
4	2	0.74	[[0 0] [6 17]]
4	3	0.74	[[0 0] [6 17]]
4	4	0.74	[[0 0] [6 17]]
4	5	0.71	[[0 0] [6 15]]
4	6	0.74	[[0 0] [6 17]]

(b) Table 2

group	attack	test_recall	test_confusion_matri
0	1	0.38	[[0 0] [13 8]]
0	2	0.36	[[0 0] [14 8]]
0	3	0.36	[[0 0] [14 8]]
0	4	0.36	[[0 0] [14 8]]
0	5	0.38	[[0 0] [13 8]]
0	6	0.33	[[0 0] [14 7]]
1	1	0.48	[[0 0] [12 11]]
1	2	0.48	[[0 0] [12 11]]
1	3	0.48	[[0 0] [12 11]]
1	4	0.48	[[0 0] [12 11]]
1	5	0.48	[[0 0] [12 11]]
1	6	0.48	[[0 0] [12 11]]
2	1	0.54	[[0 0] [11 13]]
2	2	0.54	[[0 0] [11 13]]
2	3	0.54	[[0 0] [11 13]]
2	4	0.54	[[0 0] [11 13]]
2	5	0.54	[[0 0] [11 13]]
2	6	0.54	[[0 0] [11 13]]
3	1	0.59	[[0 0] [9 13]]
3	2	0.57	[[0 0] [10 13]]
3	3	0.57	[[0 0] [10 13]]
3	4	0.59	[[0 0] [9 13]]
3	5	0.59	[[0 0] [9 13]]
3	6	0.57	[[0 0] [10 13]]
4	1	0.43	[[0 0] [12 9]]
4	2	0.48	[[0 0] [12 11]]
4	3	0.48	[[0 0] [12 11]]
4	4	0.48	[[0 0] [12 11]]
4	5	0.43	[[0 0] [12 9]]
4	6	0.48	[[0 0] [12 11]]

(c) Table 3

Fig. 2.

classification of malicious programs as such. By adding new permissions that will be checked by the classifier. It's important to note that the permissions that were added wasn't chosen randomly but after checking, from a big dataset of permissions, which permissions were the most popular among the malicious programs. After adding these permissions we let our classifier do his job and we discovered that the number of classified malicious programs was much greater than the default one.

As feature work, we aim to add even more permissions to the classifier so it can classify the malicious programs even better. But to find suitable permissions it will be better not to check the most popular among the malicious programs, but the most popular groups of connected permissions.

REFERENCES

[1] Harel Berger, Chen Hajaj, Amit Dvir *When the Guard failed the Droid:*, A case study of Android malware.

[2] William Enck, Machigar Ongtang, and Patrick McDaniel , On Lightweight Mobile Phone Application Certification.

[3] KENNETH OLMSTEAD and MICHELLE ATKINSON *Apps Permissions in the Google Play Store:*, .