ANDROfa Under Attack: Assessing the Limitations and Vulnerabilities of a Dynamic Android Malware Detection System

Elad Sezanayev
Ariel University
eladsez36@gmail.com

***Abstract*—This paper presents an evaluation of the robustness of the ANDROfa classifier [1], a dynamic android malware detection system based on resource consumption. Through an experimental approach, the study finds that by exploiting a specific vulnerability in the classifier, it is possible to evade detection by running malware outside of the app context. The results of this research provide insight into the limitations and vulnerabilities of the ANDROfa classifier and highlights the need for robust countermeasures to improve the robustness of dynamic classifiers against evasion techniques in the field of android malware detection.**

I. Introduction

Malware classification is an important task in the field of cybersecurity as it helps to identify and isolate malicious software from legitimate software. With the increasing popularity of mobile devices and the widespread use of Android operating systems, android malware classification has become a crucial issue for protecting mobile users from various types of cyber attacks.

Dynamic classifiers, which are based on the behavior of the malware rather than its static characteristics, have been shown to be effective in detecting unknown and evolving malware. These classifiers, particularly those based on resource consumption, have the advantage of being able to detect malware that is designed to evade detection by traditional static analysis methods.

The ANDROfa classifier is a dynamic classifier that is based on resource consumption. It is designed to detect android malware by analyzing the resource consumption patterns of the malware during its execution. The ANDROfa classifier is a promising approach for android malware classification, but it is important to understand its limitations and vulnerabilities. This paper aims to evaluate the robustness of the ANDROfa classifier and identify potential vulnerabilities that can be exploited by attackers to evade detection. In other words, The main objective of this paper is to attack the ANDROfa classifier and create a way to create malwares in a way that would go unnoticed.

II. Related Work

Previous research on evading machine learning-based malware detection has focused on various techniques such as feature manipulation, model inversion, and adversarial examples. Feature manipulation attacks involve modifying the features of a malware sample in order to evade detection by the classifier. Model inversion attacks aim to reconstruct the classifier's decision boundary by inferring the model's parameters from a set of labeled samples. Adversarial examples are maliciously crafted samples that are designed to cause the classifier to make a wrong decision.

In the context of android malware detection, several studies have proposed dynamic analysis-based methods to detect malicious behaviors at runtime. These methods use various features such as system calls, network traffic, and API calls to classify an app as malicious or benign. Some of the well-known android malware detection systems are Drebin [2], MaMaDroid [3] and Andromaly [4]. These systems have been shown to be effective in detecting android malware, but they have also been shown to be vulnerable to evasion attacks.

Previous studies have highlighted the vulnerability of android malware detection systems to evasion attacks. For instance, in [5], a technique was proposed for evading detection by injecting malicious code into benign apps and disguising it behind legitimate API calls. Similarly, [6]proposed a method for evading detection by manipulating system call sequences of the app. Additionally, [7] proposed a technique for evading detection by encrypting the payload and concealing it within the benign part of the app. Furthermore, [8] proposed a method for evading detection by employing code obfuscation techniques such as code scrambling and control-flow obfuscation.

In contrast to these previous studies, this research focuses on exploiting a specific vulnerability in the ANDROfa classifier that relies on the /proc file system of an android device.

III. Methodology

In order to evaluate the performance of the ANDROfa classifier, this study utilizes the publicly available Drebin dataset, which comprises of a large number of benign and malicious Android applications. The dataset, which is sourced from various platforms such as Google Play, third-party app stores, and malware repositories, offers a

diverse representation of malware families and types, making it an ideal dataset for evaluating the generalization capabilities of the classifier. However, due to the time-consuming nature of the classifier's data collection stage, the study was only able to train the classifier on a minimal subset of the dataset.

In order to classify android applications as benign or malware, the ANDROfa classifier extracts a set of features from the proc filesystem of an android emulator at runtime of the application. The feature set is detailed below by categories:
From /proc/stat: user cpu, nice cpu, system cpu, idle cpu, iowait cpu, irq cpu, softirq cpu, steal cpu.
From /proc/[pid]/statm: total program size (on ram, in pages), resident set size, shared pages.
From /proc/[pid]/stat: user space time, kernel space time, wait for children running on user space time, wait for children running on kernel space time, Virtual memory size in bytes, Number of threads, CPU number last executed on, number of minor faults, number of minor faults with childs, number of major faults, number of major faults with childs.
From /proc/net/dev: number of bytes received, number of packets received, number of bytes transmitted, number of packets transmitted. The chosen feature set allows the classifier to capture the behavior of the application during its execution. These features represent a set of characteristics that are specific to malware and can be used to distinguish them from benign applications.

In order to evaluate the performance of the ANDROfa classifier, several evaluation metrics are used, including accuracy, precision, recall, F1-score, and the confusion matrix. These metrics are commonly used in the field of machine learning and provide a comprehensive evaluation of the classifier's performance. The accuracy measures the proportion of correctly classified instances in the dataset, precision measures the proportion of true positive cases among the positive predictions, recall measures the proportion of true positive cases among all actual positive cases, F1-score is the harmonic mean of precision and recall, and the confusion matrix is a table that is used to define the performance of a classification algorithm.

The attack that was conducted in this research specifically targeted the ANDROfa classifier's reliance on the /proc file system of an android emulator at runtime of the application. It is important to note that the ANDROfa classifier used in this research employs a Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel for its classification model. Additionally, the attack presented in this research is considered a whitebox attack as the attacker has knowledge of the features, ML model, and other details of the classifier being targeted. This type of attack can be more challenging to defend against as compared to a blackbox attack where the attacker has limited knowledge of the classifier's inner workings.

As previously discussed, the classifier extracts its classification data from the /proc file system and takes data for the features from the following files: /proc/stat, /proc/net/dev, /proc/[pid]/statm, and /proc/[pid]/stat.
The weak spot identified in the classifier is that it relies on the application process and only the application process. This vulnerability can be exploited by running malicious code from another context, such as another process, that is not related to the process that runs the application and does not use too many general things that are checked in the first two files.

To exploit this vulnerability, a malware app for android was developed that writes an init.rc service file to the filesystem of the android. The service file runs malicious code in the next boot of the emulator. This malware app was then injected into 100 benign apps that the classifier had already classified as benign and that have permissions to write to external storage (to write the service file).
The functionality of the injected apps was verified using Droidbot to ensure that the injection did not destroy the functionality of the apps. The apps were then re-classified with the ANDROfa classifier.

IV. Result

The results of the attack on the ANDROfa classifier show that the classifier's performance was significantly impacted by the attack. Before the attack, the classifier had an accuracy of 73% and a precision of 96% when classifying the benign apps that were used in the experiment. The recall at this point was 73.51%. After the attack, the classifier's accuracy dropped to 12.87%, its precision dropped to 12.87%. Now we are not interested in the recall because now we are not classifying un-malicious apps. This indicates that the classifier was unable to detect the presence of the malware in the benign apps that were injected with the malware app. The high drop in precision indicates that the attack was able to generate a high number of false positive detections, which would lead to a high number of malicious apps being incorrectly classified as benign. It is worth mentioning that the data was split into a training set (80%) and a test set (20%) for the purpose of evaluating the performance of the model.

The results also showed that the functionality of the injected apps was not affected by the injection of the malware app. The apps were tested using Droidbot and no difference in functionality was observed between the original apps and the injected apps. This indicates that the attack was able to propagate malware without affecting the functionality of the apps.

It is worth noting that the attack was conducted on a dataset of 100 benign apps that the classifier had already classified as benign and that have permissions to write to external storage. This limitation of the dataset can be considered as a weakness of the research since it is not

known how the attack will perform against other datasets. However, this attack shows that the ANDROfa classifier has a vulnerability that can be exploited by attackers, and it highlights the need for robust countermeasures to improve the robustness of dynamic classifiers against evasion techniques.

V. Conclusion

In conclusion, this research has presented an evaluation of the robustness of the ANDROfa classifier, a dynamic android malware detection system based on resource consumption. By exploiting a specific vulnerability in the classifier, it was possible to evade detection by running malware outside of the app context. The results of this research provide insight into the limitations and vulnerabilities of the ANDROfa classifier, and highlight the need for robust countermeasures to improve the robustness of dynamic classifiers against evasion techniques in the field of android malware detection.

One key insight from the attack is the importance of considering the entire runtime environment of an application, rather than just the app itself, when designing a dynamic classifier. The ANDROfa classifier's reliance on the /proc file system of an android emulator at runtime proved to be a weak spot that could be exploited by attackers. This highlights the need for dynamic classifiers to take a more holistic approach to runtime analysis, rather than focusing solely on the app or the /proc file system.
It is important to note that further research should be conducted to determine the longevity of the malware's presence on the device, as the attack may not show immediate effects, but rather infect the device with something that would come up after a longer period of time. This highlights the need for ongoing monitoring and detection efforts to ensure the security of mobile devices.

Another insight from the attack is the need for dynamic classifiers to be able to adapt to evolving threats. As malware becomes more sophisticated, it will likely continue to evolve and adapt to evade detection by existing classifiers. Therefore, it is crucial for dynamic classifiers to be able to adapt and improve their detection capabilities in response to these evolving threats.

In addition, this research also highlights the need for better evaluation methods for dynamic classifiers. While the ANDROfa classifier performed well on the Drebin dataset, the research was limited by the use of a pre-existing dataset, which may not have fully represented the range of malware families and types that the classifier would encounter in the real world. Therefore, better evaluation methods are needed to more accurately assess the performance and robustness of dynamic classifiers in the face of real-world threats.

Overall, this research has provided valuable insights into the limitations and vulnerabilities of the ANDROfa classifier and the importance of robust countermeasures for dynamic classifiers in the field of android malware detection. It is crucial for future research to continue to address these issues in order to improve the effectiveness of dynamic classifiers in detecting and preventing android malware.

References

[1] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 31–38, IEEE, 2017.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, pp. 23–26, 2014.

[3] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," *arXiv preprint arXiv:1612.04433*, 2016.

[4] 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*, vol. 38, no. 1, pp. 161–190, 2012.

[5] S. Bhandari, R. Panihar, S. Naval, V. Laxmi, A. Zemmari, and M. S. Gaur, "Sword: semantic aware android malware detector," *Journal of information security and applications*, vol. 42, pp. 46–56, 2018.

[6] L. Onwuzurike, *Measuring and Mitigating Security and Privacy Issues on Android Applications*. PhD thesis, UCL (University College London), 2019.

[7] S. Mirza, H. Abbas, W. B. Shahid, N. Shafqat, M. Fugini, Z. Iqbal, and Z. Muhammad, "A malware evasion technique for auditing android anti-malware solutions," in *2021 IEEE 30th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp. 125–130, IEEE, 2021.

[8] W. F. Elsersy, A. Feizollah, and N. B. Anuar, "The rise of obfuscated android malware and impacts on detection methods," *PeerJ Computer Science*, vol. 8, p. e907, 2022.