Towards an automated malware detection approach on Android

Louison Gitzinger,
Université Rennes 1, France
louison.gitzinger@inria.fr

As the number of distinct malware increases, it becomes more and more difficult to characterize and define precisely what a malicious application is and thus, harder to design efficient malware detection tools. Most of current malware detection solutions which rely either on static or dynamic analysis requires human efforts. In this work, we propose a novel approach to dynamically detect malware at runtime without requiring human intervention.

I. Introduction

As of today, the Android system reaches 88% of smartphones market share [1]. Following this trend, the Android application ecosystem gets bigger everyday, thanks to an active developer community that distribute their applications through major [2] [3] and alternative [4] online application market places. For example, Google Play Store holds 3.3 millions applications with a rate of more than 50 000 submissions a month.

However as for any operating system, third party applications remain a major attack vector. This is why a certain quantity of applications, known as malwares, are deliberately malicious softwares built for damaging phones and are dangerous for end users. Malwares can, for example, gain root access on the phone, steal or encrypt user private data or send premium SMSs.

To fight this problem and strengthen user safety, Android actors provide many efforts. On one side, the Android's open source development approach allows to collaboratively locate vulnerabilities and develop mitigations. Latest Android system updates have been shipped with major security improvements and restrictions to reduce the system attack surface. As examples, Android 5 was released with a runtime permission system, SE Linux, and a restricted access to the `/proc` directory has been implemented into Android 8. On the other hand, Android related companies set up solutions to reduce the number of malwares available on their online application markets. To do so, companies mix state of the art software analysis techniques to verify each submitted application. First, they check the application signature against a database containing malware signatures. Then, they use static analysis to detect potential malicious behaviors such as unwanted information flow.

Despite the efforts to reduce the attack surface on the Android system, various security issues are discovered every day. The number of Android malwares explodes in 2018 [5], which confirms the recent years trend. Moreover, studies [6] [7] show an increasing number of distinct malwares families. This is partly because smartphones are continually improved on all levels to provide users better services, ranging from a wide range of connectivity options (GSM, WI-FI, GPS, Bluetooth, NFC) to a high availability of personal information such as contacts, messages or browsing history. Depending their intentions, malwares adapt their behavior to take advantage and exploit vulnerabilities of these new services.

Moreover, we recall that Android applications run on a particular Linux system (Android OS) and have particulriaties that differentiate them from standard linux softwares. In particular, the Android's application model allows multiple applications to easily communicate with each other. To enable this particular feature, Android applications are built of components able to communicate with each other, inside and outside the application context. This global communication system can be used to exchange information between two application as well as to access the phone's data storage system. These specificities make an Android application to behave differently than known Linux malwares, and thus bringing new challenges in malware detection.

As the number of distinct malware increases, it becomes more and more difficult to characterize and define precisely what a malicious application is. This blurred vision of a malware is preventing us from developing generalized analysis techniques. The fact is that even in a given family, malwares are using totally different behaviors to reach their goal. The result is a particularly heterogeneous ecosystem which prevents today's tools from categorizing them as malicious applications from the same kind.

To detect them, previous studies used a techniques based on either static or dynamic analysis for detecting data leakage and apk signatures to recognize a known malicious application. The current diversity in malware applications could make these solutions less effective.

In the industry, anti-viruses detect malwares by checking each application against an updated collection of known malicious applications signatures. Typically, an usual anti-virus will scan a given application looking for portions of code that already exists in its database. If this portion of code is flagged as suspicious, the application will have more chance to be categorized as malicious. However, the malware ecosystem becomes so complex that new malwares implement known malicious behaviors in a totally different manner making impossible for an anti-virus to reason on the application code.

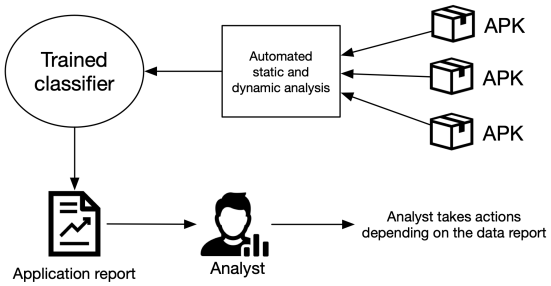
As an alternative to conventional anti-viruses, we observe a trend in latest studies to use learning techniques with data collected by both static and dynamic analysis [8]. Particularly, they train a learning model by choosing a small set of features

such as dangerous permissions, protected api calls or data leaks at runtime supposed to characterize any malware. These approaches have their limitations for three reasons. First, it is sensitive to false positives because a non negligible number of benign applications could use a combination of these features for a non malicious purpose. Secondly, these learning models are made to analyze and classify an application prior to their execution. This implies that if the model fails to detect a malware at installation time, no mechanism will prevent it from doing more damages at runtime. Finally, they do not take into account the fact that malwares try to escape these detection techniques by using a combination of more elaborated techniques. For example, a malware that wants to steal user contacts without asking the `READ_CONTACTS` permission could ask another application that already have this permission granted to send it the contacts via *Intent* communication. In this way, any technique that does not check both Android permissions and inter app communication will never notice the malicious behavior. Such malware practices must be monitored at runtime in a real environment to be detected. To tackle this, studies such as *Andrubis* [9] runs applications in a monitored virtual machine. However, The inherent problem is that malwares become capable of detecting that they are running within a virtual environment [10]. Moreover, these automated virtual machines fake user actions with probabilistic inputs that may never trigger the malicious behavior.

II. Approach

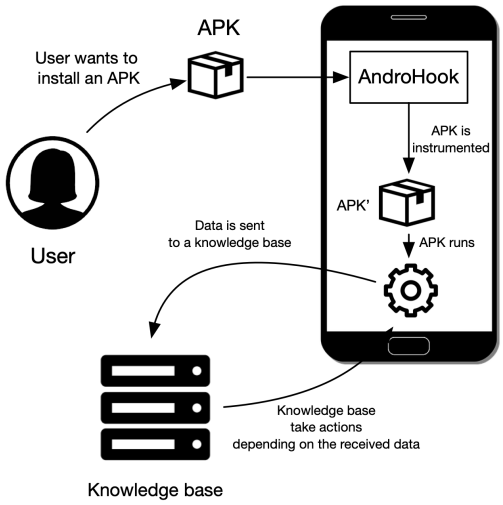
Our work offers an alternative to current studies by proposing an approach which allows to detect a malicious behavior during the application execution. To do so, we instrument the application at installation time to tailor a dynamic analysis that will be triggered at execution time. When the application runs, we use a fuzzy logic to pro-actively learn from the data in real time by the dynamic analysis.

Fig. 1: State of the art process such as *Drebin*



The Figure 1 schematically describes how current state of the art solutions works in terms of malware detection. First, an automated tool using both static and dynamic analysis is built to extract meaningful features from a given application. Secondly, a classifier model is built using learnings techniques from the extracted features of a data set of malicious and benign applications. Once trained, the model can takes the features of any new application as input and produce a report which gives the percentage of *malwareness* of the given application. It is then up to the analyst that reads the report to take actions about the analyzed application such as choosing whether to keep or reject the application. In all cases, the whole process is performed before execution time. On the contrary, Figure 2 shows our alternative approach. When the user wants to install a new application, our tool, which is embed on the phone, takes as input the legacy application to generate an instrumented one that is thereafter installed on the phone. As a result, as soon as the instrumented application is run, meaningful data is collected to be analyzed on the fly, and actions can be taken directly on the phone like stopping the application process for example, in case of dynamic malware detection.

Fig. 2: Approach based on pro-active learning



A. Building the model

To be capable of detecting malicious behavior at runtime, we provide a model shaped by application data collected in real time. A naive technique would be to build this model exhaustively by collecting all existing method call during execution. Unfortunately, it leads to a state explosion problem, and deteriorates application performances and thus user experience. To circumvent this problem we leverage on preivous studies [8] and [11] to build a custom abstract model containing the most representative patterns of each malware family. We develop a process that allows us to identify an entire malware family from diagnoses made on a few of them. We state the following hypothesis: from the data collected of a few malware family samples, it is possible to generate an abstract model characterizing all malwares within this family. To this aim, we narrowed our exploration space by focusing on a smaller part of the malware ecosytem. This approach allows the design of tools able to analyze the distinct behaviors of malwares within the same family.

B. Conclusion

To overcome challenges mentioned above, we provide *AndroHook*, a tool that leverage both on static and dynamic

analysis that allows us to collect data characterizing the intrinsics behaviors of a given application. *AndroHook* performs a static analysis on a target application to collect the data needed to customize the dynamic analysis according to a malware family and instrument it accordingly. When run, the instrumented application sends data to an interpreter which perform dynamic analysis diagnose the application execution.

The *AndroHook* tool aims to resolve two open issues. First, we need to build a model capable of detecting malwares from real time application features. To reach this aim, our strategy is to develop the toolchain that generates an abstract model, using fuzzy-logic or deep learning inference, capable of covering at once a set of similar malwares. Secondly, *AndroHook* needs to perform an effective dynamic analysis that maintain application performances and preserve application code integrity. To solve this issue, we make our tool scale by hooking native system calls in an efficient manner in order to reduce the overhead cost to hook system primitives at runtime.

REFERENCES

[1] S. 2018. (2018) Global mobile os market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018. Test. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[2] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 1. ACM, 2014, pp. 221–233.

[3] "Amazon appstore for android launches," https://www.marketplace.org/2011/03/22/tech/codebreaker/amazon-appstore-android-launches, (Accessed on 09/27/2018).

[4] "Getjar - download free apps, games and themes apk," https://www.getjar.com/, (Accessed on 09/27/2018).

[5] S. malware forecast. (2018) Sophoslabs 2018 malware forecast. Test. [Online]. Available: https://www.sophos.com/en-us/en-us/medialibrary/PDFs/technical-papers/malware-forecast-2018.pdf?la=en

[6] D. J. Tan, T.-W. Chua, V. L. Thing *et al.*, "Securing android: a survey, taxonomy, and challenges," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 58, 2015.

[7] X. Jiang and Y. Zhou, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy(SP)*, vol. 00, 05 2012, pp. 95–109. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2012.16

[8] D. Arp, M. Spreitzenbarth, M. HÃŒbner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket ." Internet Society, 2014. [Online]. Available: https://www.ndss-symposium.org/ndss2014/programme/drebin-effective-and-explainable-detection-android-malware-your-pocket/

[9] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. Van Der Veen, and C. Platzer, "Andrubis–1,000,000 apps later: A view on current android malware behaviors," in *Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014 Third International Workshop on*. IEEE, 2014, pp. 3–17.

[10] T. Vidas and N. Christin, "Evading android runtime analysis via sandbox detection." ACM Press, 2014, pp. 447–458. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2590296.2590325

[11] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques," *International Journal of Information Security*, vol. 14, no. 2, pp. 141–153, Apr. 2015. [Online]. Available: https://link.springer.com/article/10.1007/s10207-014-0250-0