Challenges for Reliable and Large Scale Evaluation of Android Malware Analysis

Jean-François Lalande, Valérie Viet Triem Tong, Mourad Leslous and Pierre Graux
CentraleSupélec, Inria, Université de Rennes 1, CNRS, IRISA
F-35065 Rennes, France
jean-francois.lalande@inria.fr

INVITED TALK EXTENDED ABSTRACT

Since Android became the first smartphone operating system, malware developers have put large efforts to craft new threats uploaded to the Google Play store and other third market places. Companies and researchers now include in their activities the analysis of malware targeting smartphones. Most of the time, the problem that is addressed consists in deciding if an application should be considered as a malware or not. Nevertheless, once a malware is tagged as a malicious application, users that have been infected ask for more technical explanations about the threat they have been exposed to. Dissecting a malware requires a lot of efforts for a security analyst to be conducted and companies are in demand of new tools for automatizing the analysis.

From a research perspective, testing new ideas about malware analysis requires performing experiments on malware datasets. Compared to other operating systems, Android has fast development cycles with a new major release each year. A lot of malware samples do not run anymore when executed on new versions of Android. Experiments of the literature becomes quickly out of date and non reproducible when studying few samples. Thus, working on larger datasets, built at the time of writing, may give more consistent experimental results. New challenges come from using such datasets. First, as the behavior of the samples are unknown, the obtained results from the experiments are difficult to evaluate. Second, the experiment itself may require a large amount of time, depending of the quality of the automatization and the complexity of the analysis. Third, the protections that are put by developers in the malware decrease the quality of the results. This paper discusses these challenges and describes our efforts to build reliable and large scale experiments.

A. Lessons Learned from Manual Investigation

We started our research efforts in malware analysis with the help of several students that were asked to reverse known or unknown malware samples. Well known samples, such as SimpleLocker [1] or DroidKungFu [2], help to discover the basics of reverse engineering and to study complex attacks in a reasonable amount of time. Most of the time, picking randomly a sample identified as malicious by online platforms such as VirusTotal gives malware of low interest that send premium rated SMS or display unwanted ads.

Studying malware samples helped us to understand that malware developers now implement countermeasures to defeat both static and dynamic analysis tools. To illustrate and discuss these countermeasures, we built a small representative dataset, called kharon dataset [3], that technically describes the way some samples try to escape analysis tools. These countermeasures can take different forms.

For defeating static analysis, malware developers use: malformed files raising an exception when parsed but keeping intact the application behavior; obfuscated payloads mixed with benign code which complexifies manual investigations; native code that hide parts of the payload; dynamic loading or packers that make the code unavailable without analyzing the runtime. These countermeasures make unreliable using solely static analysis techniques.

For defeating dynamic tools, malicious applications contain countermeasures for interacting and testing their environment: emulators can be detected because of the lack of hardware components or because emulation side-effects [14]; reconnaissance techniques can recognize known analysis tools; logic bombs may encapsulate the payload and prevent its execution [4]; malicious code may require that the user uses the graphical interface of the application. Thus, any fully automated experiment on a dataset may suffer from these countermeasures and there are little chances to observe the dynamic behavior of a malware sample at runtime. For example, Tam et al. reported a successful stimulation of 2.78% to 3.08% of the malicious code related to SMS sending on their dataset [5]. When targeting more general parts of the code (telephony abuse, introspection, contact access), we observed a covering of 20% at runtime, without any particular stimulation [6]. These variations suggest that the chosen dataset may influence the experimental results and research efforts should be encouraged for elaborating evaluation datasets.

B. Existing Malware Datasets

Several papers have defined working datasets of malware in order to improve the repeatability of experiments. As pointed out by Wei et al. [7], the malware Genome project [8] is one of the first dataset to be largely used by the community which is now outdated and no longer supported.

In parallel of this academic dataset, a lot of researchers use randomly chosen malware from public or fee paying repositories such as Contagio Mobile, VirusTotal, Koodous. This approach has several advantages. It avoids building manually any contextual information for each sample, for example the family of the samples, the location of the malicious code, the triggering conditions, the required environment. Nevertheless, manipulating a large dataset without any knowledge about the samples makes difficult a precise evaluation of the obtained results. At least, conducting large scale experiments on thousands of malware may improve the confidence in the results, compared to an evaluation conducted on a small number of samples manually reversed.

Recently, new efforts have been done for building up-to-date datasets with additional contextual information, as an improvement compared to random datasets. For example, AndroZoo [9] is a large dataset of three millions Android applications. In addition to the samples themselves, AndroZoo contains an analysis of pairs of repackaged applications [10] which helps to distinguish the original application and the repackaged one with the introduced payload. Another dataset, called AMD [7], has been built from 2010 to 2016 and contains 24,650 malware spread among 71 families. Each representative sample has been manually reversed and a graphical representation of the malicious parts of the code is given. Nevertheless, a structured and digital version of these results is not available yet. The AMD dataset is, by far, the first to structure the nature of the malicious code but we believe that more information is needed when performing experiments when the malware is run.

Using an existing dataset must be clearly linked with the reasons authors created it. For example, a dataset without enough precise attributes would not be usable for machine learning algorithms. Similarly, for an experiment that intends to execute the payload at runtime, the name of all methods that are malicious are required, especially when the code is repackaged in a benign application. The conditions to be met for executing the payload, i.e. the Android version, the events to trigger, the data expected by the malware, needs to be given for repeating experiments. Without such information, we believe that it is hazardous to compare experimental results on the same dataset as illustrated later in Section D and E.

C. *Building Malware Datasets*

Building a dataset from scratch is thus a difficult task. During our work on a specific dataset focused on Android malware that use native code, we had to deal with the problem of determining if a sample can be confirmed as malicious. We collected 2000 malware samples by downloading each day 20 recent samples from the Koodous repository and 30 random samples from the AndroZoo repository [9]. We identified 683 samples that use native calls in order to build our specialized dataset. Submitting these samples to VirusTotal is the usual method that enables to decide if these samples are real malware (TP for True Positive) and not regular applications submitted to the repositories (False Positive) [11-13, 20]. As shown in Figure 1, 48% of samples are not recognized by any antiviruses used by VirusTotal. Then, for a detection of at least 5 antivirus and more the curve is almost linear: there is no obvious threshold to decide that a sample has been recognized by enough antiviruses to classify it as a TP. We were expecting a drop of detection for a certain number of antiviruses, as represented by the clear curve. Additionally, these results may change with time, as the used antiviruses update frequently there data. We conclude from this experiment that using VirusTotal as an Oracle for confirming that a sample is malicious is not reliable, especially for recent samples.

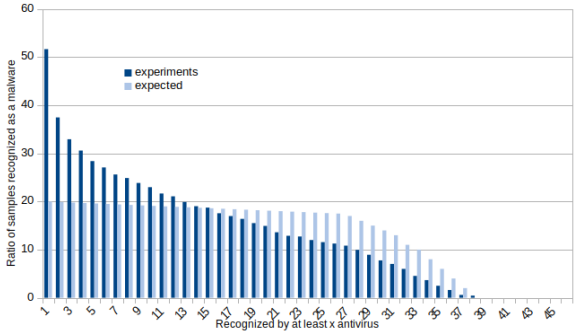

Figure 1. Ratio of recognized malware by at least x antivirus

D. *Automatizing Dynamic Analysis*

Large scale experiments require an automatization of the analysis process. For a static analysis, a distributed infrastructure can be employed, but for executing samples to observe its actions, the problem is more complex.

First of all, the setup of the execution environment is of primary importance. We believe that executing a sample in an emulator would lead to inconsistent results because of their ability to detect emulation artifacts [14]. Thus, a reliable environment requires a real smartphone connected to the internet and with plausible traces of usage. The version of the operating system may also influence the results and thus, repeating experiments may avoid compatibility issues and help to get a working malware execution. Additionally, because experimenting with a malware may damage the operating system, a fresh install should be restored between two experiments. Such a setup cannot be easily deployed in a cloud infrastructure which limits the scalability of experiments.

Second, proper inputs should feed the application that is analyzed [15]. In particular, the graphical interface should be manipulated carefully [6, 16], external events such as SMS receiving or internal messages should be generated if needed [5, 17]. Fuzzing techniques [18], symbolic or concolic executions [19] are now actively investigated for reducing the number of inputs necessary to trigger the payload.

Finally, an oracle is required to evaluate the success of an experiment where a malware is executed. When using a pre-built dataset where the malicious code is clearly identified, the oracle should monitor the successful execution of this part of the code. For a dataset that is built manually or if the malicious code is not clearly pointed out, the implementation of the oracle becomes a scientific challenge.

With these remarks in mind, we designed an execution platform, called the Kharon platform located in the High Security Laboratory of Inria Rennes. It is composed of two components: a static and dynamic analysis that are orchestrated

by GroddDroid [5, 20], and a kernel component, Blare [21], that helps to monitor the actions of the malware[1]. We briefly discuss the performance of the platform in the next section.

E. Performances of Dynamic Analysis

Even if the performances are linked to the difficulties of the scalability of the experimental setup, some other figures from our experiments should be mentioned.

Executing unknown applications, even if in a polished environment that is as close as possible to a real smartphone, may result in an unexpected crash. When launching the application (or implemented component such as a service) without any further stimulation, we observed 5% of crash for the AMD dataset [7] and 20% of crash for our specialized native dataset. Such results are of good quality, compared to the 76% of crashed apps reported by Yang et al. [22]. Malware that have crashed are usually removed from experiments [22] and never investigated, hiding potential dangerous samples.

We measured the analysis time when a sample is operated by the Kharon platform. The preparation of the smartphone requires 60s, the static analysis phase is of 7s on average and the dynamic analysis lasts 245s on average. In total, we need 312s for executing one suspicious method. Thus, for the AMD dataset where our 135 samples contain 99 suspicious locations on average, we need in the worst case 48 days of experiment when using one smartphone. Repeating experiments on Android versions from KitKat to Oreo, which covers 72% of devices according to Google [2], would require 8 repeated experiments, requiring in total approximatively one year. Such a simple estimation illustrates the limits of dynamic analysis. In particular, locating the malicious code is of primary importance, before executing this code.

F. Conclusion

This paper has presented some of the difficulties related to experiments with Android malware. Currently, no dataset is enough mature for being used for comparison purpose and for different research objectives. Challenges remain unaddressed for building datasets with meta-data and execution environments that help to reproduce large scale experiments. Finally, processing samples with multiple smartphones, in a scalable way, remains a difficult challenge that requires rethinking the architecture of both the experiments and the internal components of Android.

REFERENCES

[1] R. Lipovsky, RESET analyzes first android fileencrypting, TOR-enabled ransomware, June 2014.
http://www.welivesecurity.com/2014/06/04/simplocker/

[2] X. Jiang, Security alert: New sophisticated android malware droidkungfu found in alternative chinese app markets, May 2011. http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu.html

[3] N. Kiss, J.-F. Lalande, M. Leslous, and V. Viet Triem Tong, "Kharon dataset: Android malware under a microscope," in The LASER Workshop: Learning from Authoritative Security Experiment Results, 2016, pp. 1–12.

[4] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "TriggerScope: Towards Detecting Logic Bombs in Android Applications," in 37th IEEE Symposium on Security and Privacy, 2016, pp. 1–33.

[5] K. Tam, S. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: Automatic Reconstruction of Android Malware Behaviors," in 22nd Annual Network and Distributed System Security Symposium, 2015.

[6] A. Abraham, R. Andriatsimandefitra, A. Brunelat, J. F. Lalande, and V. Viet Triem Tong, "GroddDroid: A gorilla for triggering malicious behaviors," in 10th International Conference on Malicious and Unwanted Software, MALWARE 2015, 2015, pp. 119–127.

[7] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep Ground Truth Analysis of Current Android Malware," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2017, pp. 252–276.

[8] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in IEEE Symposium on Security and Privacy, 2012, no. 4, pp. 95–109.

[9] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: collecting millions of Android apps for the research community," in 13th International Workshop on Mining Software Repositories, 2016, pp. 468–471.

[10] L. Li et al., "Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting," IEEE Trans. Inf. Forensics Secur., vol. 12, no. 6, pp. 1269–1284, Jun. 2017.

[11] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: semantics-based detection of Android malware through static analysis," in 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 576–587.

[12] Y. Duan et al., "Things You May Not Know About Android (Un)Packers: A Systematic Study based on Whole-System Emulation," in Annual Network and Distributed System Security Symposium, 2018.

[13] M. Zheng, M. Sun, J. C. S. Lui, and C. Science, "DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware," in 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013, pp. 163–171.

[14] T. Vidas and N. Christin, "Evading Android Runtime Analysis via Sandbox Detection," in 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 447–458.

[15] M. Y. Wong and D. Lie, "IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware," in The Network and Distributed System Security Symposium, 2016, no. February, pp. 21–24.

[16] Gianazza, F. Maggi, A. Fattori, L. Cavallaro, and S. Zanero, "PuppetDroid: A User-Centric UI Exerciser for Automatic Dynamic Analysis of Similar Android Applications," 19-Feb-2014.

[17] M. Y. Wong and D. Lie, "IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware," in The Network and Distributed System Security Symposium, 2016, no. February, pp. 21–24.

[18] S. Rasthofer, S. Arzt, S. Triller, and M. Pradel, "Making Malory Behave Maliciously: Targeted Fuzzing of Android Execution Environments," 2017 IEEE/ACM 39th Int. Conf. Softw. Eng., pp. 300–311, 2017.

[19] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "TriggerScope: Towards Detecting Logic Bombs in Android Applications," in 37th IEEE Symposium on Security and Privacy, 2016, pp. 1–33.

[20] M. Leslous, V. Viet Triem Tong, J.-F. Lalande, and T. Genet, "GPFinder: Tracking the Invisible in Android Malware," in 12th International Conference on Malicious and Unwanted Software, 2017, pp. 39–46.

[21] R. Andriatsimandefitra, S. Geller, and V. Viet Triem Tong, "Designing information flow policies for Android's operating system," in IEEE International Conference on Communications, 2012, pp. 976–981.

[22] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps," 2017, pp. 288–302.

[1] Platform: http://kharon.irisa.fr and resources: http://kharon.gforge.inria.fr

[2] Distribution dashboard : https://developer.android.com/about/dashboard

This work has received a French government support granted to the COMIN Labs excellence laboratory (ANR-10-LABX-07-01