# Detection and Identification of Android Malware Based on Information Flow Monitoring

Radoniaina Andriatsimandefitra, Valérie Viet Triem Tong

# Detection and Identification of Android Malware Based on Information Flow Monitoring

Radoniaina Andriatsimandefitra
CIDRE Research group
Centrale Supélec / INRIA
FRANCE
radoniaina.andriatsimandefitra@supelec.fr

Valérie Viet Triem Tong
CIDRE research group
Centrale Supélec / INRIA
FRANCE
valerie.viettriemtong@centralesupelec.fr

***Abstract*—Information flow monitoring has been mostly used to detect privacy leaks. In a previous work, we showed that they can also be used to characterize Android malware behaviours and in the current one we show that these flows can also be used to detect and identify Android malware. The characterization consists in computing automatically System Flow Graphs that describe how a malware disseminates its data in the system. In the current work, we propose a method that uses these SFG-based malware profile to detect the execution of Android malware by monitoring the information flows they cause in the system. We evaluated our method by monitoring the execution of 39 malware samples and 70 non malicious applications. Our results show that our approach detected the execution of all the malware samples and did not raise any false alerts for the 70 non malicious applications.**

***Keywords*—*Malware detection, Malware classification, Android, Information Flow***

## I. Introduction

Android is an operating system mainly made for mobile devices. Because of its large adoption and the value of data it may process, it became the target of malicious applications that try to steal or corrupt data on the device, misuse the system, etc. The most common technique to detect malware consists in analysing statically applications and comparing them with a signature database. A signature can be a simple hash, bytes in a file, etc. However, such techniques can be easily defeated with code obfuscation, metamorphism and polymorphism [1]. One way to avoid that shortcoming is to monitor an application during its execution thanks to Dynamic Information Flow Tracking (DIFT).

DIFT consists in monitoring where sensitive data goes within a given environment at execution time. On Android, it has often been used to find out if an application is leaking sensitive data such as contact list, pictures, etc. Such approach has shown itself helpful but we believe that DIFT has also other uses. In a previous work [2], we proposed to represent Android malware through the information flows they induce in the system. This way a malware is characterized by the damages it causes in the system independently from its effective code. This method permits avoiding all possibilities that malware have to escape static analysis. In this article we push further this previous work. We propose to use these profiles as behavioural signature for detecting their execution.

In the rest of the document we present some related works about information flow monitoring (section II). We detail how malware profiles are obtained (section III). Last, we assess the relevance of these profile as behavioral signatures and experiment an intrusion detection system based on these signatures (sections IV and V).

## II. Related works

In [3], Enck et al. present TaintDroid, a modified version of Android, that dynamically tracks information flows to detect privacy leaks. Android applications are written in Java and their code is interpreted by the Dalvik virtual machine. Thanks to modifications done to the Dalvik virtual machine, TaintDroid is able to track information flows within Java applications, across them and between Java applications and files. Enck et al. evaluated their approach by analysing 30 Android applications and showed that 2 out of 3 were leaking sensitive data to remote entities. Since its publication, TaintDroid has been used in other tools such as Andrubis [4] or App Fence [5]. Another information flow monitor for Android is DroidScope [6]. DroidScope is an Android emulated device that tracks information flows at hardware level. It is not limited by the type of applications to be monitored like TaintDroid but cannot be used on real devices since the information flow tracking is done by the emulator.

In TaintDroid, DroidScope and other similar works, information flow analysis has been used to detect privacy leaks. Such approaches have helped to detect security threats on Android but we believe that information flow analysis have other uses too. Wüchner et al. have for instance use quantitative data flow graphs to detect attacks on Windows in [7]. A quantitative data flow graph describes any information flow between system objects and the quantity of information exchanged between these objects during the execution of the system. Based on a manual analysis of Windows malware samples, they manually designed QDFGs that represent classes of attacks such as malware replicating itself, malware probing the system state by querying the registries, rootkits etc. To detect an attack on the system, they then compare the QDFG of the system with the ones they built to describe classes of attacks. They evaluated their approach by analysing malicious and non-malicious programs. Their results showed that they were able to detect $95.65\%$ of the malicious applications that were active during their analysis (22) and got a false positive rate of $1.65\%$ when analysing non-malicious applications. According to the authors, this FPR was higher for non-malicious installers ($6.90\%$).

Similarly, we proposed a method to characterize automatically Android malware based on information flows they cause at system level [2]. Instead of measuring the quantity of information exchanged between system objects, we used DIFT to observe where data belonging to the application under analysis go in the system. From the observed information flows, we then constructed a system-wide information flow graph that we used as a profile of the application. We built the profiles of 19 malicious applications from 4 malware families (DroidKungFu1, DroidKungFu2, jSMSHider and BadNews) and automatically computed from these profiles 4 malware profiles. A malware profile is also system-wide information flow graph and is obtained by extracting the common parts of malicious application profiles. Each malware profile corresponds to a different malware and describes the attack it carried out. In this current work, we push further the work we did in [2] and propose to use these malware profiles to detect the execution of Android malware. The next section explains the environment we use to analyse and the system-wide information flow graph.

## III. Background

### A. AndroBlare: an information flow monitor

In this work, we use AndroBlare[1] to track information flows on Android. AndroBlare is aware of information flows occurring between files, processes and sockets. We use AndroBlare because it has a real system-wide view of information flows compared to other solutions like TaintDroid which is limited to applications written in Java. Furthermore, we want to be able to perform both the analysis and detection on a real device so monitoring at hardware level was neither a viable solution. Using a real device instead of an emulator avoids any change of behaviour based on the environment analysis. Measurements done in [8] shows that AndroBlare causes an overhead of $8\%$ on execution time which means that it should not badly impact the user experience.

Technically speaking, AndroBlare is a kernel module that uses the hooks provided by the Linux Security Module framework [9] to track information flows. Thanks to these hooks, a module can intercept and control the syscalls made by user processes. A syscall is a request sent by a process to the kernel to interact with the system. Some of these syscalls, such as `read`, cause information flows and are intercepted by AndroBlare to track information flows. Each time an information flow is observed, AndroBlare updates the taint associated to the object of which content has been changed by the observed flow to take into account its new content. In addition, AndroBlare also logs the observed information flows in the kernel log. An entry describes an information flow and contains 5 types of information as shown in figure 1: a timestamp, a description of the source container, a description of the destination container and the taint of the information involved in the observed flow. In our case, there is only one taint as we only track one type of data. An object is tainted if it contains data from the application we are analysing. An information container is described with its type, its name and its system identifier.

[1]https://www.blare-ids.org

```
[1][HONKDROID] FILE MALICIOUS.APK 1 > PROCESS APP 2 > {1}
[3][HONKDROID] PROCESS APP 2 > FILE PAYLOAD 3 > {1}
```

Fig. 1: Example of AndroBlare log that describes information flowing from an apk file to a file named payload

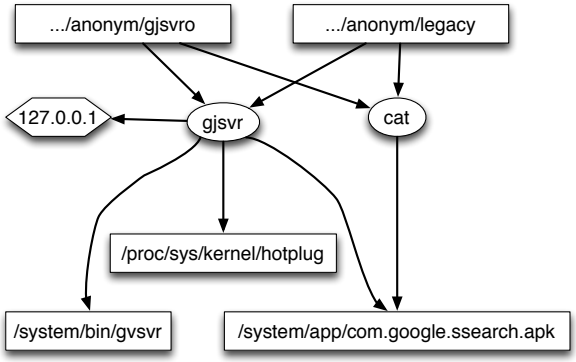


Fig. 2: Profile of DroidKungFu1

### B. Malware profile: system information flow graph

A system information flow graph is a representation of the information flows observed by AndroBlare and describes how some sensitive data are propagated within the system. Formally, it is a labelled directed graph $G = (V, E)$ of which nodes $v \in V$ represent information containers and edges $e \in E$ from a node $v_1$ to a node $v_2$ represent an information flow from $v_1$ to $v_2$. Each node has three attributes: its system id, its name and its type (process, file or sockets). These three attributes are respectively denoted $v.id$, $v.name$ and $v.type$. An edge has two attributes identified as $e.flow$ and $e.timestamp$. $e.flow$ is a collection of information identifiers involved in the flow corresponding to $e$. The attribute $e.timestamp$ is a list of timestamps at which AndroBlare observed the flow corresponding to $e$. A graph is automatically built from an AndroBlare log.

In our previous work [2], we used these graphs as application profiles and malware profiles. Such profiles describe how an application or a malware propagates its data within the system. An application profile is built from an AndroBlare log whereas a malware profile is extracted from application profiles. Figure 2 shows the profile of DroidKungFu1 that was automatically extracted from the profiles of 5 applications infected with this malware. It describes the installation of two applications on the device and the flow corresponding to a root exploit. We believe that applications belonging to the same malware family have a common behaviour. We showed that by extracting malware profiles from the malicious application profiles. In this work, we extend this point of view by using these malware profiles to detect any applications injected with a malware for which we have a profile.

## IV. Detection of Android malware execution

To detect an application infected with a known malware, we look for a match between the information flows involving its data and the edges of the malware profiles. A known malware

is a malware for which we have computed a profile. The two following conditions must be fulfilled to consider that a flow matches an edge. First, the information flow and the edge must involve the same information. Second, the source and destination nodes of the edge must describe the source and destination containers of the flow. If these conditions are fulfilled, we consider that the analysed application is infected with the malware of which profile enables us to find the match. We therefore consider the application as infected with the malware of which profile enables us to find the match.

Our analysis environment is a Nexus S device running a modified version of Android 4.0. This modified version contains a kernel to which we added AndroBlare and tools related to the manipulation of taints. To analyse an application, we install it on the device, mark its `apk` and use it as a normal user would do. Meanwhile, AndroBlare tracks the tainted data and logs information flows involving them. We mark the `apk` because it contains all the information belonging to the application when it is installed on the device. After the analysis, we then feed our detection engine with the AndroBlare log to detect the execution of known malware.

## V. Detection evaluation

The quality of a detection engine is generally evaluated by measuring its True Positive Rate and its True Negative Rate. The TPR corresponds to the percentage of malware samples detected using our approach and the TNR corresponds to the percentage of non malicious applications that were not detected as a malware sample. To compute the TPR, we analysed 39 malicious applications belonging to one of the following malware families: DroidKungFu1, DroidKungFu2, jSMSHider and BadNews. Most of these applications come from two public datasets of Android malware: Contagio[10] and Malware Genome Project [11]. The remaining ones, some samples of BadNews, were manually collected.

DroidKungFu1 [12] and DroidKungFu2 [13] are Android malware that were discovered in 2011. They exploit vulnerabilities of the Android system to get root access and install other applications on the system. JSMSHider [14] is also a malware that installs additional applications on the system. Instead of exploiting vulnerabilities of the system, it exploited the fact that some system keys used to sign Android custom ROM were publicily available. It was thus possible to create "legitimate" applications with system privileges. BadNews [15] is an Android malware discovered in 2013. Its behaviour is controlled by a Command and Control server which was still active during the time of our experiment. Based on a manual analysis we did on some samples of the malware, we discovered that the C&C server can ask the malware to dowload files, install applications, add shortcuts on the main screen, display notifications and update the address of the C&C server. In our previous work [2], we automatically computed a profile for each one of these families and we use in this work these profiles to detect the execution of applications belonging to one of these 4 malware families.

To evaluate the TNR of our approach, we analysed 70 of the most downloaded applications from Google Play. We consider these 70 applications as non malicious because they raised no alarm when we submitted them to Virus Total [16]. VirusTotal provides a web service that analyses any type files to find out if it is a malware. Any file submitted on Virus Total is scanned by 49 anti-virus products. None of the 70 applications we submitted was detected as a malware so it is safe to say that they are not malicious.

### *Experiment*

To evaluate our approach, we analyse each application as described in section IV. In addition to that, we also introduce key events in the system when executing malicious applications. These events are events that trigger the attack carried out by the malware. By doing that, we make sure that the malicious code is executed during our analysis and it enables us to decide whether our approach really works or not since our detection approach is based on the fact that we detect the execution of the malicious code.

The developers of DroidKungFu1 and DroidKungFu2 added a time bomb in their malicious applications. The applications wait that a predefined amount of time has elapsed before executing their malicious code. The elapsed time is computed from a value stored in the shared preferences of the application. We manually change this value to lure the applications in executing their malicious code. BadNews samples wait for a special intent before contacting the C&C server. We manually send this special intent to the component the applications that initiates the contact with the server.

### *Evaluation results*

Tables I and II present the results we got using our dataset of 109 applications. The first column lists the category to which the applications belong, the second one lists the number of applications per category, the third one lists the profiles causing the alerts and the last one lists the number of information flows observed during the analysis of each category. In the case of the malicious applications, the categories are the malware families and the number of samples are decomposed in two. The first number is the number of known samples and the second one is the number of unknown samples. Known samples are malicious applications that were analysed when we computed our malware profiles in [2]. Unknown samples are malicious applications that we did not analyse before.

Our results show that all the malicious applications we analysed were detected and non-malicious applications did not raise any alarms. More precisely, almost all malicious applications were detected with the profile of the malware family to which they belong except two samples of DroidKungFu1. All BadNews samples were for instance detected with the profile of BadNews and 14 samples of DroidKungFu1 out of 16 were detected with the profile of DroidKungFu1. The two remaining samples were detected with the profile of DroidKungFu2. The reason these two samples were detected as DroidKungFu2 is that we observed during their analysis the same behaviour as the one described by the profile of DroidKungFu2. We thus believe that either these samples have been misclassified in their origin database or classified using criteria different from ours.

Furthermore, a complete match with the malware profile was observed for most of the malware samples. A complete match means that all information flows described by a malware

| Malware | # Samples | Profile |
|---|---|---|
| BadNews | 5 + 5 | BadNews |
| DroidKungFu1 | 7 + 7 | DroidKungFu1 |
| | 0 + 2 | DroidKungFu2 |
| DroidKungFu2 | 3 + 3 | DroidKungFu2 |
| jSMSHider | 4 + 3 | jSMSHider |

TABLE I: Detection results for the analysis of 39 malicious applications. TPR: 100%

| Category | # Samples | Profile |
|---|---|---|
| Games | 48 | None |
| Tools | 8 | None |
| Social networks | 5 | None |
| Multimedia | 5 | None |
| Magazine | 3 | None |
| Antivirus | 1 | None |

TABLE II: Detection results for the analysis of 70 non malicious applications. TNR: 100%

profile were observed during the analysis. An incomplete match means that only some of them were observed. That was the case of some samples of DroidKungFu1. The reason is that only one application was installed instead of two. Therefore the part of the malware profile describing the installation of the second application was never observed.

## VI. Conclusion

To sum up, we presented in this work a method to detect the execution of Android malware based on the information flows they cause in the system. Our method is based on the idea that samples of the same Android malware should have a common behaviour. We started exploring this idea in a previous work by characterizing Android malware with the information flows they cause in the system. We analysed samples of 4 malware families and automatically built 4 malware profiles such that each profile corresponds to a different malware family and describes the attack carried out by the malware. In this work, we pushed further these findings and proposed a method that uses these profiles as behavioural signatures to detect the execution of Android malware. We evaluated our approach by analysing 39 malicious applications and 70 non malicious applications and detected the execution of all malicious applications without raising any alarms for the non malicious applications. In other words, using our dataset of 109 applications, we obtained a TPR of 100% and a TNR of 100%, which are better than the results obtained by Wüchner et al. in [7] using a similar approach on Windows.

Such results may appear astonishing and in the following we give the reasons why we got them. First, an Android malware is generally distributed as different repackaged applications. It means that samples of the same malware have a common behaviour due to the malicious code they share. As our malware profiles describe the attacks carried out by malware, then a sample of a malware should always be detected as long as its malicious code is executed. Second, our detection capability is based on the execution of the malicious code. As our profiles describe the attack, no alarm will be raised unless the malicious code is executed. In our experiments, we made sure that it was executed as we introduced some key events. Therefore, the attack was carried out each time we analysed a malicious application and our detection engine successfully detected it.

As written above, the detection capability of our approach is based on the execution of the malicious code in the malware samples. In this work, we manually introduced events that trigger the malicious code to make sure that it was executed but in a future work we plan on using an automatic approach to do the triggering.

## References


[1] Rastogi, V., Chen, Y., Jiang, X.: Droidchameleon: Evaluating android anti-malware against transformation attacks. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security. ASIA CCS '13, ACM (2013)

[2] Andriatsimandefitra, R., Viet Triem Tong, V.: Capturing Android Malware Behaviour using System Flow Graph. In: NSS 2014 - The 8th International Conference on Network and System Security, Xi'an, China (2014)

[3] Enck, W., Gilbert, P., gon Chun, B., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: An information-flow tracking system for real-time privacy monitoring on smartphones. In: In Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI). (2010)

[4] Lab, I.S.S.: Anubis - malware analysis for unknown binaries. http://anubis.iseclab.org/

[5] Hornyack, P., Han, S., Jung, J., Schechter, S., Wetherall, D.: "these aren't the droids you're looking for": Retrofitting android to protect data from imperious applications. In: CCS'11. (2011)

[6] Yan, L.K., Yin, H.: Droidscope seamlessly reconstructing os and dalvik semantic views for dynamic android malware analysis. In: Proceedings of the 21st USENIX Security Symposium. (August 2012)

[7] Wüchner, T., Ochoa, M., Pretschner, A.: Malware detection with quantitative data flow graphs. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. ASIA CCS '14, New York, NY, USA, ACM (2014) 271–282

[8] Andriatsimandefitra, R., Geller, S., Tong, V.V.T.: Designing information flow policies for android's operating system. In: IEEE International Conference on Communications. (2012)

[9] Wright, C., Cowan, C., Smalley, S., Morris, J., Kroah-Hartman, G.: Linux security module framework. In: OLS2002 Proceedings. (2002)

[10] Parkour, M.: Contagio mobile. contagiominidump.blogspot.com

[11] Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy. SP '12, Washington, DC, USA, IEEE Computer Society (2012) 95–109

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

[13] Jiang, X.: Security alert: New droidkungfu variants found in alternative chinese android markets. http://www.csc.ncsu.edu/faculty/jiang/DroidKungFu2/

[14] Strazzere, T.: June 15, 2011 security alert: Malware found targeting custom roms (jsmshider). https://blog.lookout.com/blog/2011/06/15/security-alert-malware-found-targeting\-custom-roms-jsmshider/

[15] Rogers, M.: The bearer of badnews. https://blog.lookout.com/blog/2013/04/19/the-bearer-of-badnews-malware-google-play/ (2013)

[16] : Virustotal. https://www.virustotal.com/