A Comparative Study of Android Malware Behavior in Different Contexts

Catherine Boileau[1], François Gagnon[2], Jérémie Poisson[2], Simon Frenette[2] and Mohamed Mejri[1]
[1]*Université Laval, Québec, Canada*
[2]*CybersecLab at Cégep de Sainte-Foy, Québec, Canada*

Keywords: Dynamic Malware Analysis, Android, Sandboxing.

Abstract: One of the numerous ways of addressing the Android malware threat is to run malicious applications in a sandbox environment while monitoring metrics. However, dynamic malware analysis is usually concerned with a one-time execution of an application, and information about behaviour in different environments is lacking in the literature. We fill this gap with a fuzzy-like approach to the problem: by running the same malware multiple times in different environments, we gain insight on the malware behaviour and his peculiarities. To implement this approach, we leverage a client-server sandbox to run experiments, based on a common suit of actions. Scenarios are executed multiple times on a malware sample, each time with a different parameter, and results are compared to determine variation in observed behaviour. In our current experiment, variation was introduced by different levels of simulation, allowing us to compare metrics such as failure rate, data leakages, sending of SMS, and the number of HTTP and DNS requests. We find the behaviour is different for data leakages, which require no simulation to leak information, while all results for other metrics were higher when simulation was used in experiments. We expect that a fuzzing approach with others parameters will further our understanding of malware behaviour, particularly for malware bound to such parameters.

1 INTRODUCTION

The Android operating system was the most popular platform for mobile devices in 2015. Since Android applications may be distributed via many unofficial markets and not only through the official Google Play Store[1], it contributes to put Android forward as the most targeted operating system by malware (PulseSecure, 2015). In order to study malware behaviour and protect Android against this rising threat, sandboxing has been used to analyze malware behaviour at multiple levels. A number of hybrid analysis tools for Android are available to investigate an application behaviour. However, given the rate at which malware hit the markets, time is of essence and in order to process all the new apps popping every day, these tools analyze the application once and subsequently present the saved report when a known app is resubmitted for processing.

However, studying the same malware in different contexts to compare its behaviour has not been done before. Such an approach would allow us to better understand what triggers malware characteristics, like malware depending on specific events or parameters to expose their malicious activity. We therefore explore how repeated experiments on the same sample, with a given parameter modified each time, could help unveil malware behaviour where a single experiment would not. We test our approach in a client-server sandbox, using virtual Android devices where a sample is installed following a sequence of actions called scenario, where a selected parameter is changed from one experiment to the other. In our present experiment, we modulate the level of user simulation to compare our metrics.

In this paper, we will first present, in Section 2, related work about publicly available sandboxes performing dynamic analysis. Then, a brief description will follow, in Section 3, of our sandbox environment, its possible configurations, our dataset and the data collection process. Afterwards, a description of the experiment will be presented in Section 4 as well as the scenarios used. Finally, our preliminary results will be discussed in Section 5, together with future work in Section 6, and concluding remarks in Section 7.

[1]https://play.google.com/store

Boileau, C., Gagnon, F., Poisson, J., Frenette, S. and Mejri, M.
A Comparative Study of Android Malware Behavior in Different Contexts.
DOI: 10.5220/0005997300470054
In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 1: DCNET*, pages 47-54
ISBN: 978-989-758-196-0

2 RELATED WORK

Before the first Android malware was discovered in August 2010 (Dunham et al., 2014), sandboxing techniques were already useful to fight PC-based digital threats (Bayer et al., 2009; Bayer et al., 2006; Willems et al., 2007). Malware then started to spread to the mobile world, and static (Arp et al., 2014; Arzt et al., 2014; Gonzalez et al., 2014; Zheng and Sun, 2013) and dynamic (Burguera et al., 2011; Enck et al., 2014; Rastogi et al., 2013) analysis tools were adapted to face the new challenge. Afterwards, hybrid systems using both static and dynamic analysis (Zhou et al., 2012; Eder et al., 2013) sprang to life and matured into the following public sandboxes analyzing Android malwares.

First among them is ANDRUBIS (Neugschwandtner et al., 2014), a system performing both static and dynamic analysis on a large scale. Using similar tools as our sandbox, and keeping tracks of numerous metrics, ANDRUBIS analyzes an app once, and if submitted again, the report from the first run is presented to the user. Mobile-sandbox (Spreitzenbarth et al., 2013) is another hybrid analysis system tracking native API calls. As their goal is to detect malware via a new metric, they process their apps once.

CopperDroid (Reina et al., 2013) is also part of the dynamic analysis sandbox family and their approach is closer to our own, as they compare applications behaviour with and without user simulation. They are thus able to demonstrate the usefulness of simulation during an experiment, whereas we pushed this logic to multiple scenarios and are interested in putting results into perspective. Tracedroid (van der Veen et al., 2013) is also a hybrid analysis service, based on method traces as an extension on the virtual machine. As the other tools, they aim at processing a quantity of apps, to then label and sort out malware. To our knowledge, Android sandboxing is mostly used to detect malware from good applications and not to observe behaviour in different contexts. Moreover, once a malware is processed, it is not analyzed again.

As our approach implies multiple runs with variations in parameters, environments and network configurations, we find that fuzzing is somehow related to our work. Per definition, fuzzing is an automated technique that provides boundary test cases and massive inputs of data in order to find vulnerabilities (Au et al., 2012). In the Android world, the framework AndroidFuzzer was developed in the cloud to that intent. Some tools focused on a particular area of testing, such as permissions (Au et al., 2012), activities or intents (Sasnauskas and Regehr, 2014; Ye et al., 2013). However, to the best of our knowledge, fuzzing has not been extended to the analysis or detection of malwares.

3 METHODOLOGY

In order to observe variations in application behaviour, we use a client-server sandbox (Gagnon et al., 2014a; Gagnon et al., 2014b) that executes an experiment in a particular context. An experiment is defined as the execution of a scenario (a series of actions to configure, install, test, and collect data on apps) applied to all samples of our dataset. The server side of the sandbox is responsible for the management of the experiment (i.e., running all scenarios against all available apps), whereas clients manage runs (i.e., the execution of the selected scenario for a particular app of the pool). The client-server architecture was selected for its scalability, for it is as simple as adding a client to process more applications in the same time. A server-controlled experiment also ensures that all runs in an experiment will execute the same scenario with the same parameters (network configuration, android version, etc), since the configuration of the experiment is done on the server and applied to each client available for the experiment. Thus, the sandbox architecture lowers the variability in an experiment to allow a sound comparison between runs of a same experiments and between experiments on the same malware.

3.1 Client-server Architecture

First, our sandbox server works as a controller over the whole experiment. To begin with, all parameters are defined in the configuration file, to prepare the environment (network configuration and android version) and the parameters (scenario to use, dataset to analyze, etc.) needed for the experiment. Once configured, the server will start and monitor the pool of clients. Each new client will register itself to the server pool of clients, thus notifying the server of its availability to perform a part of the experiment. As soon as the server detects that a client is ready to execute a run, it will give that client the scenario to execute and the application to analyze.

At that moment, the client starts its run by loading the scenario and the application to install and analyze. The scenario is parsed by the client, to extract its parameters and actions to perform (see following Section 3.2 for more details). Following that, the analysis phase starts with the sequence of actions to execute. Once the client concludes the action phase, data collected (see following Section 3.4) by multiple tools is bundled into a result file and the server is notified that results are ready to be stored. Finally, the client changes its status back to available, thus letting the server knows it is ready to execute another run.

Thus, the server pushes runs to available clients until all the applications in the dataset are processed, completing the experiment. Gathered data is then stored into our results database, where it is available for post-processing and analysis.

3.2 Experiment Scenario

The scenario, an XML document, contains two parts: the environment configuration and the action sequence to execute. The environment configuration indicates which Android virtual device (AVD) must be used, thus specifying the version of Android to use, and what network services should be provided (e.g. DNS and HTTP proxies). The action sequence is basically an ordered list of commands that will be launched by the client to successfully perform a credible simulation of the application. Each scenario starts with the same set of instructions, to prepare the environment for the experiment. For example, starting the AVD, waiting for the boot sequence to complete and starting the monitoring of metrics are parts of the initialization sequence.

Once this preparation phase is over, the second phase, the installation of the app to test and user simulation, starts. When a scenario includes user simulation steps, the Application Exerciser Monkey[2] tool, is used. Its capacity to generate pseudo-random and system-level events to navigate the application under scrutiny makes for a basic simulation. Finally, once all actions have been executed, the data collection phase is launched and a report with all the metrics is sent back to the sandbox server.

Also, an important element of a virtual sandbox is the level of network simulation, specified in the scenario. For the experiments mentioned in this paper, the sandbox was equipped with a DNS proxy (relaying DNS queries/reponses to a real server) as well as a gateway to redirect all outgoing network traffic generated by the mobile applications to our own fake server. The fake server only allows the establishment of TCP handshake to take place for any connections. By properly completing TCP handshake, we expect some apps to perform requests (e.g., HTTP GET requests) and want to observe their characteristics.

[2]http://developer.android.com/tools/help/monkey.html

3.3 Applications Dataset

For these experiments, a set of 5519 apps was used. A mix of legitimate applications and malware was combined from various sources, with a bias towards malware applications: 3519 apps were labelled as malware while the remaining 2000 were deemed legitimate applications. A thousand apps were selected on both the Google Play Market[1] (labelled the GooglePlay dataset) and AppsAPK[3] (labelled AppsAPK dataset), in a random fashion. For malware, The Malware Genome Project[4] (Zhou and Jiang, 2012) (labelled MalGenome) dataset contains 1260 apps while the DroidAnalytics (Zheng and Sun, 2013) (labelled DroidAnalytics) dataset was the source for the remaining 2259 apps constituing our bank.

3.4 Data Collection

This client-server sandbox relies on different tools to collect static and dynamic information during the experiments. A processing of the Android manifest for permissions used, broadcast receivers and intents is performed. Of the dynamic analysis tools used, Taintdroid (Enck et al., 2014) monitors the sensitive data leakages to public channels. Equally, outgoing SMS and phone calls are recorded, either through the use of a monitoring agent installed on the AVD or an instrumented version of the Google Emulator. Finally, the rest of the network traffic is recorded for post-experiment analysis of protocols and requests used.

4 EXPERIMENT

As a preliminary study, three experiments were performed, based on three different scenarios. Our first scenario was the installation of the application only (labelled InstallOnly). Our second scenario was installing the application and then launching it upon a successful install (labelled InstallStart). Finally, our last scenario included the installation of the application, its launching and a simulation of random user actions (labelled InstallSim).

These three scenarios were selected to constitute a proof-of-concept that variations of parameters in scenarios would lead to a means of comparing malware behaviour. Since most of the aforementioned dynamic analysis tools are researching ways to expand their simulation engine, a difference in the level of simulation appeared as a natural starting point for our first experiments.

[3]http://www.appsapk.com/
[4]http://www.malgenomeproject.org/

Our objective with using these scenarios is to better understand at what point an application is susceptible to perform an action that will be caught by one of our data collection processes. We are also interested is learning whether all metrics show similar variations from different contexts. In other words, is it mandatory to simulate user interaction for every metric? Our three scenarios represent an increasing degree of sophistication. An application exhibiting malicious behaviour during InstallOnly would be more vicious for the end user (although easier to observe in a sandbox) than one exhibiting malicious behaviour only during InstallStart (or InstallSim).

5 RESULTS

Results for each run were collected and compared using the metrics presented in this section. Failure rate of runs, apps sending SMS, sensitive data leaks and network traffic (DNS and HTTP requests) were analyzed to picture malware behaviour in different contexts.

5.1 Failure Rate

The first metric we compared was the failure rate of runs. To qualify as a success, all the steps of the scenario in a particular run had to be completed. On the contrary, if a scenario was unable to complete, stopped or failed to execute a scenario step in the allotted time, it was considered a failure.

For our experiment InstallOnly, 420 runs failed, representing 8.3% of the applications submitted to this experiment. The numbers climbed at 667 runs (13.9%) for experiment InstallStart and 713 runs, for 18.4% of the runs in experiment InstallSim. Those increasing rates may be explained by the complexity of the scenario, as a simpler scenario, like scenario InstallStart, has fewer steps and less chances of running into a problem, than a complex one.

Also, failed runs were divided by dataset, as illustrated in Figure 1. Runs performed on applications from the MalGenome dataset for experiment InstallSim failed 4.2% of the time, while applications from DroidAnalytics, the GooglePlay and AppsAPK dataset showed a higher failure rate, as respectively 14.5%, 14.7% and 18.4%. Interestingly, when comparing scenarios InstallStart and InstallSim for each dataset, the number of runs failing to complete were very similar, for 3 out of 4 sets. Indeed, the number of apps from the AppsAPK dataset failing to install with the InstallOnly scenario was 139, but climbed at 184 apps with the InstallStart scenario.

Therefore, simulation of user actions does not significantly increase failure rate when compared to starting an app, but launching an application does when compared to installing it only.

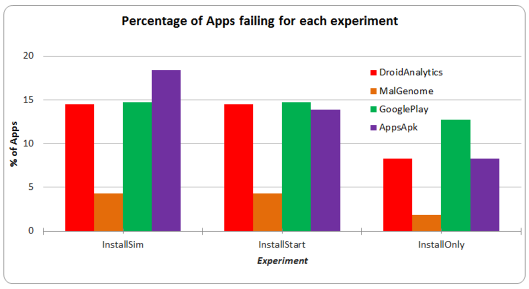

Figure 1: Percentage of applications failing, by experiment and dataset.

5.2 Sending SMS

As Figure 2 shows, no SMS were automatically sent by apps labelled as legitimate in either of the three experiments. On the other hand, malware were caught sending SMS in two of the three experiments. Moreover, our results show that starting an application is required for SMS to be sent, as no SMS were sent under scenario InstallOnly. Indeed, SMS were first recorded with scenario InstallStart applied to dataset MalGenome and DroidAnalytics, in respectively 1.11% and 1.19% of the runs. SMS were also recorded with scenario InstallSim, where the rate climbed at 8.8% of the runs on dataset DroidAnalytics and 4.4% of the runs for MalGenome dataset. Therefore, user actions simulation is in order to properly conclude if an application is automatically sending SMS.

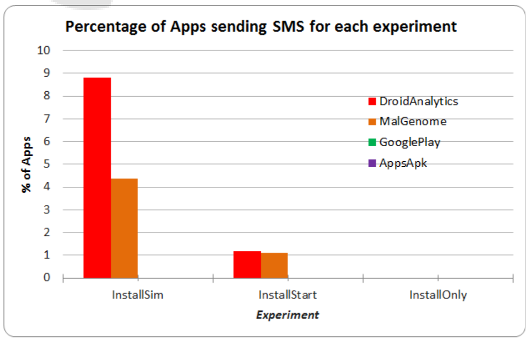

Figure 2: Percentage of applications sending SMS, by experiment and dataset.

Also, other metrics related to SMS were checked, to extend the information on malware. When malware sent SMS in scenario InstallSim (in 254 runs), an average of 2.5 SMS were sent per run. A convergence about SMS numbers and texts was also observed, as only 23 numbers and 50 texts were found. A pattern was visible in SMS sent to the same number or by the same app, as shown in Table 1. No comparison has been done on these metrics so far, as few apps in our sample turned out to be sending SMS. However, on a larger sample, this information may further help to compare and sort SMS-sending applications. It may also prove valuable when comparing metrics for families of malware, as is intended in our future work.

5.3 Data Leakages

Another metric taken during experiments is the leaking of sensitive data to public channels. Results are shown in Figure 3. As soon as sensitive data was detected, the app was considered flawed. On the whole, we found that 21,8% of applications (1205 out of 5519) had data leaks, in scenario InstallSim. Of these, 91.1% were malware (1098 apps out of 1205). Applications labelled as good also leaked sensitive data, in a much lesser way than malware: only 8.9% of the legitimate apps (107 out of 2000) were faulty. Therefore, as the SMS metric, the data leaks metric is also a good indicator of malware.

However, while comparing the results by scenario, we found that the difference between the scenario InstallSim and InstallStart running on malware was not really significant: while 1098 apps leaked data when submitted to a thorough simulation, 1011 apps leaked the data when launched. Our simulation scenario allowed us to discover only 87 more applications than our InstallStart scenario. This means, interestingly, that for the data leak metric, user action simulation is not necessary to gather a representative picture of malware; starting the application is enough to find unreliable apps.

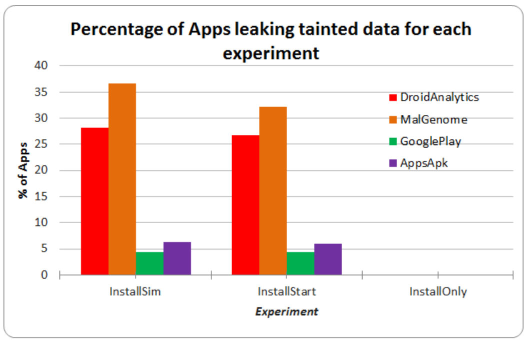

Figure 3: Percentage of applications leaking sensitive data, by experiment and dataset.

5.4 Network Traffic

During a run, network traffic is also closely monitored, to gather information about malware servers, addresses they may contact, etc. In our present experiment, DNS requests have been recorded as have been the HTTP requests.

5.4.1 DNS Requests

The metric considered for DNS requests was the average number of requests made by an app, presented in Figure 4. Results show that malware queried their DNS servers 6.5 times, while good applications averaged 3.9 requests while paired to the scenario InstallSim, once again the scenario showing the highest numbers. The scenario InstallStart also show significant, but slightly lower, averages for DNS queries; 3.3 requests for malware against 2.4 requests by good apps. However, looking at averages by dataset, we observe that the DroidAnalytics dataset shows 7.6 requests per malware, while the MalGenome dataset average (4.6 requests per malware) is similar to averages of the GooglePlay and AppsAPK dataset, (respectively 3.8 and 4.1).

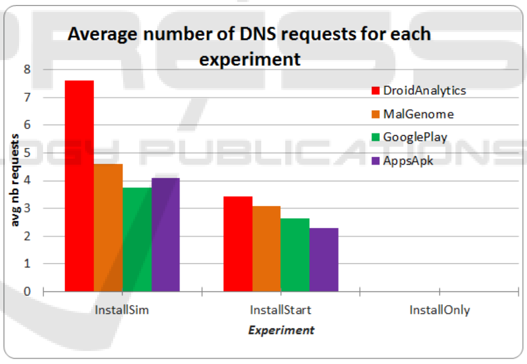

Figure 4: Average number of DNS requests sent, by experiment and dataset.

When comparing numbers only for scenario InstallSim, it is hard to tell if the average number of DNS requests is a good metric or not, given the divergent behavior of the two malware datasets. Although a high number of requests may indicate a malware when using scenario InstallSim, a lower number is not necessarily associated with a good application. Another interesting observation is the gap between averages for scenario InstallSim and InstallStart, as only the DroidAnalytics dataset shows a significant increase from scenario InstallStart to scenario InstallSim.

Table 1: SMS numbers and texts sent by some malware.

ID	SMS Number	SMS Texts
1	10621900, 10626213, 1066185829, 10665123085, ...	YXX1, YXX4, YXX2, C*X1, C*X2, 921X1, 921X2, 921X4
2	3353, 3354	70+224761
3	6005, 6006, 6008	jafun 806 1656764, jafun 806 1575475, jafan 806 2237145, igame 806 1880612, gamejava 806 3979347, gamewap 806 2188482, ...

5.4.2 HTTP Requests

Since DNS and HTTP requests are closely related, the same metric, average number of requests made per application, was used as shown in Figure 5. During experiment InstallSim, where the higher averages were compiled, malware requested HTTP services 16.4 times, while good apps showed an average of 9.5 HTTP requests. The same gap between malware and goodware is found in experiment InstallStart, although the averages are lower: 8.5 requests for malware, against 4.1 queries for legitimate apps.

However, a closer look at averages by dataset shows an average of 31.1 requests for the MalGenome dataset, while averages for GooglePlay, AppsAPK and DroidAnalytics datasets are respectively 12.0, 6.8 and 7.3 requests. Therefore, the high average of the MalGenome applications are pulling up the average for the combined malware dataset. Thus, we can conclude that a high number of HTTP requests may designate a malware, while a lower number may not necessarily indicate a good application.

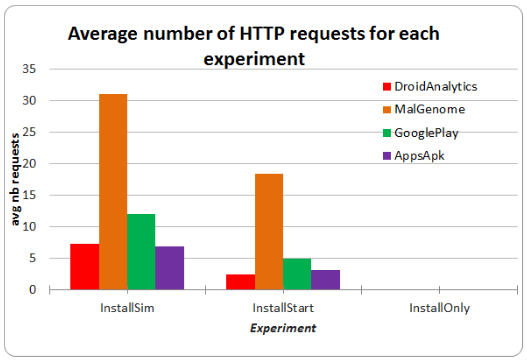

Figure 5: Average number of HTTP requests sent, by experiment and dataset.

5.4.3 Related Network Information

As for SMS, other information related to DNS and HTTP requests was considered: the domain name and URLs enclosed in the request. As some malware showed averages of both DNS and HTTP requests similar to good applications, domain names and URLs could be gathered and compared to known malware domains and URLs in order to further sort malware from good applications.

6 FUTURE WORK

To obtain a clearer understanding of the malware behavior, lots of work remains to be done. As a starting point, we would run the same scenarios against new network configurations and observe the behavior of an application when DNS responses are provided and when they are not, when HTTP traffic is proxied versus when it is not, and so on. That would allow us to study fallback plans when network resources are missing versus when they available. Also, in order to lower variability even further, we are looking at the possibility of using network traffic replay.

The same can be said about other parameters in our experiments. In this paper, the same emulator and Android OS version were used to process applications. We plan on replaying these experiments on multiple versions of Android, in order to compare behaviors of applications in early versions versus newer ones.

We would also modulate the simulation of user actions, by using short sequences versus longer sequences of simulation or random simulation against component-driven simulation. Equally, we would measure our code coverage when using simulation, to quantify what features our experiments are reaching when analyzed.

Finally, we would like to increase our malware dataset with more samples, as it would give credence to preliminary observations. We would equally like to classify those samples using different metrics (classified by date, by family of malwares, by country of origin or market of origin, etc.) in order to gain insight about the evolution of applications according to the aforementioned metrics.

7 CONCLUSION

In this paper, we presented a fuzzy-like approach to dynamic malware analysis on Android, where a given parameter is modified in each new experiment to compare variation in the application behavior. As expected, malware behavior varied with different contexts of user simulation. Thus, we were able to determine that only installing an application does not yield interesting results in any case. Also, we have found that basic user simulation generally triggers malware behavior better than no simulation, the exception being leaks of sensitive data. Finally, malware tended to have higher results for monitored metrics, even if a divergence between malware dataset was recorded for DNS and HTTP requests.

Building on these results, we plan to extend our list of parameters to vary, such as the version of the Android SDK for the emulator or the network configuration of the sandbox, to further the comparison of malware behavior. We also wish to increase the number of malware samples in our dataset, in order to cover a broader period of time, and classify those malware into families in order to trace parallels between different applications of the same family, all in hope of better understanding the global picture of malware behavior.

REFERENCES

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., and Rieck, K. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proceedings of the 2013 Network and Distributed System Security (NDSS) Symposium*.

Arzt, S., Rasthofer, S., Christian Fritz and, E. B., Bartel, A., Klein, J., Traon, Y. L., Octeau, D., and McDaniel, P. (2014). FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecyle-aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 259–269.

Au, K. W. Y., Zhou, Y. F., Huang, Z., and Lie, D. (2012). Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM.

Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., and Kruegel, C. (2009). A View on Current Malware Behaviors. In *LEET*.

Bayer, U., Kruegel, C., and Kirda, E. (2006). *TTAnalyze: A tool for analyzing malware*. na.

Burguera, I., Zurutuza, U., and Nadjm-Tehrani, S. (2011). CrowDroid: Behavior-Based Malware Detection System for Android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26.

Dunham, K., Hartman, S., Morales, J. A., Quintans, M., and Strazzere, T. (2014). *Android Malware And Analysis*. Auerbach Publications.

Eder, T., Rodler, M., Vymazal, D., and Zeilinger, M. (2013). Ananas-a framework for analyzing android applications. In *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pages 711–719. IEEE.

Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. (2014). TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2).

Gagnon, F., Lafrance, F., Frenette, S., and Hall, S. (2014a). AVP-An Android Virtual Playground. In *DCNET*, pages 13–20.

Gagnon, F., Poisson, J., Frenette, S., Lafrance, F., Hall, S., and Michaud, F. (2014b). Blueprints of an Automated Android Test-Bed. In *E-Business and Telecommunications*, pages 3–25. Springer.

Gonzalez, H., Stakhanova, N., and Ghorbani, A. A. (2014). DroidKin: Lightweight Detection of Android Apps Similarity. In *Proceedings of the 10th International Conference on Security and Privacy in Communication Networks*.

Neugschwandtner, M., Lindorder, M., Fratantonio, Y., Veen, V. v. d., and Platzer, C. (2014). ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors. In *Proceedings of the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 161–190.

PulseSecure (2015). 2015 Mobile Threat Report. Technical report, Pulse Secure Mobile Threat Center.

Rastogi, V., Chen, Y., and Enck, W. (2013). AppsPlayground: Automatic Security Analysis of Smartphone Applications. In *Proceedings of the ACM SIGSAC Conference on Computer And Communications Security*, pages 209–220.

Reina, A., Fattori, A., and Cavallaro, L. (2013). A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors. In *Proceedings of 6th European Workshop on Systems Security*.

Sasnauskas, R. and Regehr, J. (2014). Intent fuzzer: crafting intents of death. In *Proceedings of the 2014 Joint International Workshop on Dynamic Analysis (WODA) and Software and System Performance Testing, Debugging, and Analytics (PERTEA)*, pages 1–5. ACM.

Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T., and Hoffmann, J. (2013). Mobile-Sandbox: Having a

Deeper Look into Android Applications. In *Proceedings of the 28th Symposium On Applied Computing*, pages 1808–1815.

van der Veen, V., Bos, H., and Rossow, C. (2013). Dynamic analysis of android malware. *Internet & Web Technology Master thesis, VU University Amsterdam*.

Willems, C., Holz, T., and Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy*, 5(2):32–39.

Ye, H., Cheng, S., Zhang, L., and Jiang, F. (2013). Droidfuzzer: Fuzzing the android apps with intent-filter tag. In *Proceedings of International Conference on Advances in Mobile Computing & Multimedia*, page 68. ACM.

Zheng, M. and Sun, M. (2013). DroidAnalytics: A Signature Based Analytic System to Collect, Extract, Analyze and Associate Android Malware. In *Proceedings of 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 163–171.

Zhou, Y. and Jiang, X. (2012). Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the IEEE Symposium on Security and Privacy 2012*, pages 95–109.

Zhou, Y., Wang, Z., Zhou, W., and Jiang, X. (2012). Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *NDSS*.