Marshall Plan Scholarship: Final Report

Research Exchange with Northeastern University
Boston, USA
October 1, 2014 - February 28, 2015

Martina Lindorfer
Margaretenstrasse 145
1050 Vienna
mlindorfer@iseclab.org

to:
Marshall Plan Foundation
Ungargasse 37
1030 Vienna

May 2, 2015

Contents

1 Introduction

This report summarizes my 5 months research internship at the Systems Security Lab at Northeastern University in Boston, Massachusetts. During that time period I contributed to several projects related to the main topics of my PhD thesis – malware analysis and mobile security. This internship was graciously supported by the Marshall Plan Foundation.

1.1 General Information

Scholarship Recipient:
Martina Lindorfer
Margaretenstrasse 145
1050 Vienna, Austria
mlindorfer@iseclab.org
https://iseclab.org/people/mlindorfer

Home Institution:
Vienna University of Technology
Karlsplatz 13
1040 Vienna, Austria
Advisor: Edgar Weippl
https://www.sba-research.org/team/management/edgar-weippl

Host Institution:
Northeastern University
College of Information and Computer Science
202 West Village H
360 Huntington Avenue
Boston, MA 02115, USA
Advisors: Engin Kirda and William Robertson
https://www.ccs.neu.edu/home/ek
https://wkr.io

Scholarship Duration: October 1, 2014 - February 28, 2015 (5 months)
Total Duration: October 14, 2014 - March 31, 2015 (5 1/2 months)[1]

[1] Due to unfinished business at Vienna University of Technology, as well as personal reasons, I was not able to start the internship on October 1, as planned, and thus extended my stay until the end of March.

1.2 Acknowledgements

First and foremost I thank the Marshall Plan Foundation for facilitating this research exchange through a scholarship. Secondly, I thank my advisors in Boston, Prof. Engin Kirda and Prof. William Robertson for hosting me and providing an incredible learning experience, as well as my advisor in Vienna, Prof. Edgar Weippl for allowing me to take this opportunity. Furthermore, I would like to thank Prof. Manuel Egele and Prof. Dave Choffnes for inviting me to contribute to their projects.

The Secure Systems Lab proved to be a very productive and inspiring environment and I would like to thank my fellow lab members for this, most notably Collin Mulliner, Michael Weissbacher, Patrick Carter and Abdelberi Chaabane.

1.3 Short Biography

I am a fourth-year PhD student at the Secure Systems Lab at Vienna University of Technology in collaboration with SBA Research. I hold a Master's degree in Software Engineering and Internet Computing from the Vienna University of Technology, as well as a Bachelor's degree in Computer and Media Security from the University of Applied Science in Hagenberg.

My research focuses on the analysis of malicious software (malware) and mobile security. I use dynamic analysis methods to investigate the behavior of un-known software and combine them with machine-learning techniques to automatically classify malware. Furthermore, I work on identifying anti-analysis techniques that malware uses to evade dynamic analysis and, thus, to circumvent detection.

Since smartphones are becoming increasingly attractive targets for attackers (due to their direct monetization techniques), my research also shifted from desktop-based threats to mobile security more recently. Thus, I am heavily involved in the development and maintenance of Andrubis, which is an analysis sandbox to detect Android malware that is publicly available at `http://anubis.iseclab.org`. Additionally, I also work on the efficient detection and removal of malware from and across app stores, which are often used by attackers to distribute malicious apps that they disguise as mobile bank-ing apps or popular apps, like Angry Birds, to trick users into downloading them.

1.4 Motivation

The Systems Security Lab at Northeastern University is, like the Secure Systems Lab in Vienna, where I am pursuing my PhD, part of the International Secure Systems Lab[2]. Members of this lab, and the Systems Security Lab in particular, are well respected for their research in the field of malware analysis and cybercrime. Thus, the goal of this exchange was to foster the collaboration between our lab at Vienna University of Technology and Northeastern University.

One specific topic for collaboration is the analysis of Android malware: As part of this exchange we integrated CuriousDroid, a system for the automated stimulation of Android application user interfaces (developed in Boston), into our dynamic Android application analysis system Andrubis (developed in Vienna). The end result significantly enhances the behavior coverage of the current Android malware analysis and and allows us to build better Android application verification mechanisms based on machine learning.

Another goal was to benefit from the Systems Security Lab's expertise on malware analysis and mobile security and start working on the final part of my PhD thesis on malware analysis and detection techniques, with a special focus on mobile malware.

[2] `http://www.iseclab.org`

2 Events

Besides my work at university and regular meetings with my advisors to discuss the individual projects I was working on, I also took part in several other (regular) events within the Systems Security Lab, Northeastern University, as well as externally.

2.1 Research Jour Fixe

The Systems Security Lab holds regular meetings, usually once a week, amongst all professors, post-doctoral researchers and students. During this meetings, the students briefly present their projects, discuss their current progress as well as open problems. Meetings such as this are beneficial to get an overview of other students' work, provide and receive overall feedback, as well as more detailed support from students with related expertise.

2.2 Hiring Talks

During my time at Northeastern the university was looking to fill several open faculty positions. PhD students are encouraged to attend the hiring talks intended to give the applicants a chance to present their research to faculty and students, as well as subsequent informal meetings to introduce them and their teaching goals to students. I attended several of those talks and meetings, including the ones by and with Alexandros Kapravelos, Xiao Feng Wang and Vasileios Kemerlis.

2.3 Cyber Security Awareness Week (CSAW)

The Cyber Security Awareness Week (CSAW)[3] is an annual student-run cyber security event including a research conference, organized by the NYU School of Engineering in Brooklyn, New York. Since New York is only a short bus ride from Boston, and the research conference featured several high-profile speakers and talks related to my research interests, I joined several of my lab mates in attending CSAW in November 2014.

2.4 Boston Girl Geek Dinners

The Boston Girl Geek Dinners[4] (most recently renamed to She Geeks Out) are monthly events in Boston that supports local organizations focused on

[3] `http://csaw.isis.poly.edu`

[4] `http://www.bostongirlgeekdinners.com`

encouraging girls to take part in STEM related fields. The dinners are a great opportunity to meet and network with other women from all fields of science and usually feature inspiring talks by women in science sharing their experiences.

3 Mobile Malware Analysis

3.1 Background

Introduction. Android is undoubtedly the most popular mobile operating system for smartphones and tablets with a market share of almost 80% [18], however, it is also the undisputed market leader when it comes to mobile malware: Due to its widespread distribution and wealth of application (app) distribution channels besides the official Google Play Store, including a plethora of alternative markets that often deploy limited or no app review process at all [21], as many as 97% of mobile malware families target Android [14]. Estimations by anti-virus (AV) vendors as to the number of Android malware in the wild vary widely. McAfee reports about 68,000 distinct malicious Android apps [24] and Sophos collected a total of 650,000 unique Android malware samples to date, with 2,000 new samples being discovered every day [32]. Clearly, automated and scalable analysis tools for the detection of Android malware are necessary (1) to remove Android malware from markets or prevent them from entering markets in the first place and (2) to warn users about potentially harmful apps installed on their devices.

Analysis Techniques. Analysis methods have been mainly adapted from years of experience with dealing with Windows malware: *Static Analysis* is performed on an app's Android Application Package (APK) file, without executing an app. The APK file contains an app's bytecode stored in Dalvik Executable (DEX) format, resources, such as UI layouts, as well a manifest file, that, amongst other things, contains the requested permissions. In contrast, *Dynamic Analysis* executes the app in an isolated environment, a so-called sandbox, or even on a real device, in order to observe its actual behavior and collect detailed runtime traces. Both methods have their drawbacks: Static analysis can be defeated by code obfuscation, the use of reflection and dynamically loading code at runtime. Dynamic analysis suffers from incomplete code coverage as only one execution path is observed and the runtime is usually limited to several minutes. Furthermore, dynamic analysis is prone to evasion by anti-analysis techniques that aim at detecting the analysis environment and consequently do not perform any malicious activity when a sandboxed execution environment is detected. Thus, analysis tools often leverage *Hybrid Analysis* techniques that combine both static and dynamic techniques to achieve the best possible analysis results.

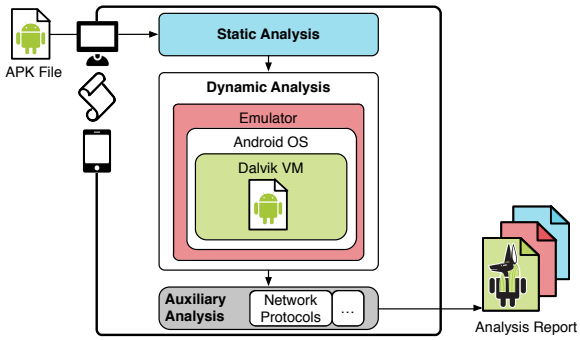

Figure 1: System Overview of ANDRUBIS [20].

Numerous such analysis tools and services have been proposed and are operated by researchers [9,11,31,33,38] and security companies [1,2,4] alike. Google itself introduced Bouncer in February 2012 [22], that checks apps in a dynamic analysis environment for anomalous behavior before listing them in the Play Store. There are no details available about this system but Google claims that Bouncer has reduced malicious app downloads by 40%. At the International Secure Systems Lab we have developed our own Android app analysis sandbox ANDRUBIS [20]. To overcome the issue of limited or no availability of related work [26], we integrated ANDRUBIS into the publicly available Windows malware analysis sandbox ANUBIS and provide this service for submissions via its web interface[5] and a dedicated mobile app available in the Google Play Store[6].

ANDRUBIS generates detailed analysis reports of unknown Android apps based on features extracted during static analysis and behavior observed through dynamic analysis during runtime. Figure 1 shows a high-level overview of ANDRUBIS: Users can submit apps through the web interface, mobile apps or batch scripts for large-scale analyses. ANDRUBIS then performs static analysis to extract information from an apps manifest and its bytecode. This mainly provides information about requested and used permissions, as well a about the app's components that is useful for the dynamic analysis. In the core dynamic analysis stage ANDRUBIS then executes the app in a complete Android environment, and its actions are monitored at both the Dalvik and the system level through virtual machine introspection (VMI) in the emulator. Besides monitoring an apps file system and network behavior as well as phone-specific activities such as sending SMS and per-

[5] `https://anubis.iseclab.org`

[6] `https://play.google.com/store/apps/details?id=org.iseclab.andrubis`

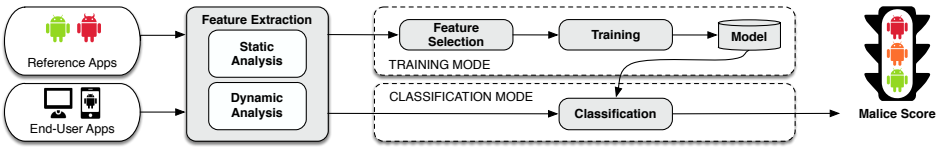

Figure 2: System Overview of MARVIN [19].

forming calls, ANDRUBIS builds on TaintDroid [13] to monitor potential data leaks to files, SMS, and most importantly the network. Finally, our system captures the network traffic from outside the Android operating system and performs a detailed network protocol analysis during post-processing.

While a detailed analysis report such as the one provided by ANDRUBIS and other aforementioned tools is useful for the in-depth analysis of potentially malicious apps by security researchers, often a simple distinction between malicious apps (malware) and benign apps (goodware) is useful. For this purpose we built an extension to ANDRUBIS, called MARVIN [19]. We leverage machine learning techniques in order to automatically classify samples as benign or malicious based on features learned from a large corpus of known malicious and benign apps. Figure 2 shows an overview of this process: In the *training mode* we use features extracted from the ANDRUBIS analysis of reference apps to train a linear classifier. The resulting model is then used in *production mode* to classify previously unseen apps from end users and provide a *malice score* indicating whether an app is malware or not. In our evaluation we learned that static analysis features are equally, or even more decisive to create the model for an accurate assessment than dynamic analysis features (but not both). Related work on Android malware classification based on static features alone also reported good results [5, 7, 16]. However, dynamic features are indispensable for capturing an app's network behavior and accurately classifying apps that are heavily obfuscated or dynamically load code at runtime. Thus, improving the coverage of dynamic analysis and thus the dynamic features for MARVIN by simulating realistic user input is one of the topics of the research collaboration with Northeastern University (CURIOUSDROID; described in Section 3.2).

Privacy Leaks. From a privacy perspective one particularly interesting behavior of mobile apps, both malware and benign apps, is the leakage of personally identifiable information (PII). Mobile apps process and store a multitude of privacy-sensitive and security-critical data that especially

malware, but also ad and tracker libraries included in widely-used legitimate apps, are interested in. In a recent study based on results collected from ANDRUBIS we found an alarming trend of an increasing number of apps leaking data over the network, almost half of the most recent apps we studied in 2014 [20]. ANDRUBIS determines these data leaks during dynamic analysis with taint tracking based on TaintDroid [13]. TaintDroid tracks the data flow of all sensitive data sources to the network, such as the user's contacts, her phone number, location, IMEI and IMSI numbers, SMS and call logs, or the browser history. Thus, building a system that makes users aware of this privacy leaks and allows them to block the leakage of PII was the topic of another research collaboration with Northeastern University (RECON; described in Section 3.3).

iOS Security. While Android has received significant attention from the research community, few researchers have tackled the topic of iOS security so far. However, related work on privacy leaks on iOS [12] and preliminary experiments with RECON have shown that a large number of iOS apps also leak sensitive information about the user to third parties. Furthermore, researchers have managed to publish malicious apps in the iTunes App Store as a proof-of-concept despite Apple's strict review policies [36]. Thus, as a final research collaboration we started looking into the security of iOS (described in Section 3.4).

3.2 CuriousDroid

CURIOUSDROID is a project by Patrick Carter, PhD student at Northeastern University and Collin Mulliner, post-doctoral researcher at Northeastern University, under the supervision of Prof. William Robertson and Prof. Engin Kirda. I contributed to the evaluation of this project, mainly based on the benefits of its superior user interface stimulation on our Android malware classification results, and supported Patrick in integrating CURIOUSDROID into our existing ANDRUBIS analysis sandbox.

A paper on this project is currently being prepared for submission to the International Symposium on Research in Attacks, Intrusions and Defenses (RAID).

Table 1: Stimulation techniques performed by ANDRUBIS [37].

Category	Target
Activities	Activities declared in the manifest
Services	Services declared in the manifest
Broadcast Receivers	Broadcast receivers declared in the manifest and registered dynamically during runtime
Common Events	Incoming SMS and phone calls, WiFi+3G connectivity, location changes, phone state changes
Random Events	Random input stream by the Application Exerciser Monkey

3.2.1 Motivation

As already mentioned in Section 3.1, dynamic analysis is an important part of Android app analysis. While there are numerous static analysis tools for Android apps available and malware classification approaches related to MARVIN based on static analysis alone lead good classification results, dynamic analysis is indispensable in many cases. One reason is that many static analysis tools are easily defeated by the use of reflection and obfuscation. But most importantly Android apps increasingly load code at runtime to dynamically extend their functionality. While dynamic code loading is popular for legitimate reasons, such as loading external add-on code, shared library code from frameworks, or dynamically updating code during beta and/or A/B testing [28], it is especially interesting for malware. Since apps are typically inspected only once, either by an app market (e.g. by Bouncer in the Google Play Store) or by an AV scanner at installation time, malicious apps can download and load their malicious payload later at runtime to evade detection. In our recent study on apps from 2010 until 2014 [20] we found this behavior increasing, especially in goodware: 29.29% of recent benign apps and 13.15% of recent malicious apps load Dalvik code at runtime.

However, the major drawback of dynamic analysis in general is the fact that only a few of all possible execution paths are traversed within one analysis run. Furthermore, Android apps can have multiple entry points besides the main activity, which is displayed to the user when an app is launched, so that apps can react to system events or interact with each other. Thus, in order to drive program execution, ANDRUBIS implements the stimulation techniques summarized in Table 1: First we invoke all exported activities and services listed in an app's manifest. Furthermore, Andrubis monitors the registration of broadcast receivers, which an app can register

for events of interest such as a phone reboot or incoming SMS. This allows us to invoke both statically and dynamically registered broadcast receivers. In addition, we simulate a set of events malicious applications are likely to react to, even when we do not capture registered broadcast receivers for these events. These events include boot completion, general phone state changes, incoming SMS and phone calls, changes in WiFi and cellular connectivity, and location changes. Finally, ANDRUBIS uses the Application Exerciser Monkey [15] ("monkey") to simulate user input by delivering a stream of pseudo-random user interface interactions such as swipes, taps and keyboard presses.

However, most user interfaces require more targeted user interaction than simple random inputs. For the examples in Figure 3 for example the monkey should try to hit the *Agree* and *Ok* buttons instead of *Cancel* or try to generate meaningful imput for name and date fields. Recent related work such as Dynodroid [23], SmartDroid [40], Swifthand [10], A3E [8], and AppsPlayground [30], has explored numerous approaches in improving the state-of-the-art of user interface exploration. However, so far these approaches to "intelligent monkeys" either require access to source code, prior static analysis of the app to identify paths to visit during dynamic analysis, static instrumentation of the app, or do not scale in a large-scale analysis environment. With CURIOUSDROID we try to overcome this limitations and build an user interface automation tool that emulates human interaction and is applicable in any environment without modification or prior knowledge of an app.

3.2.2 System Overview

In contrast to the random user interface interactions produced by monkey, CURIOUSDROID aims at automating Android app user interfaces in an intelligent, user-like manner. To achieve this, it iterates over all activities of an app in three phases: (1) *user interface decomposition*, (2) *input inference*, and (3) *input generation*.

User Interface Decomposition. In the first stage CURIOUSDROID decomposes the user interfaces of an app on-the-fly at draw-time through dynamic instrumentation with the Dynamic Dalvik Instrumentation (DDI) framework [25]. During this process it discovers all interactive user interface elements, such as editable text fields, buttons, spinners, radio buttons, and checkboxes.

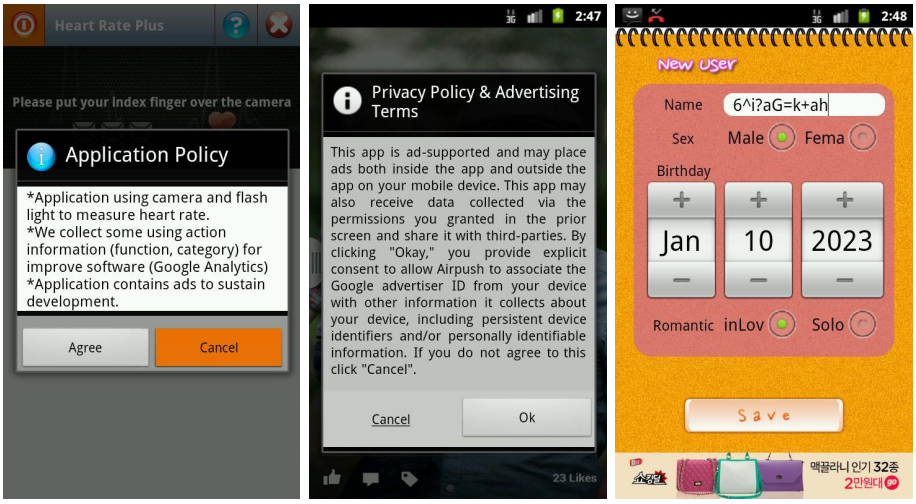

Figure 3: Examples for user interfaces benefiting from intelligent, user-like interaction as provided by CuriousDroid.

Input Inference. In the second stage CuriousDroid infer the user interactions an activity expects in order to generate meaningful inputs. In order to exercise a user interface like a human would do it is first of all necessary to infer the order in which user interface elements should be activated. Furthermore, instead of blindly providing random text as input for editable text fields, CuriousDroid infers the type of input a text field expects, such as names, phone numbers, passwords or email addresses, from its context, and populates those fields accordingly.

Input Generation. In the third stage CuriousDroid translates and feeds the inferred input to the Android touchscreen event driver that performs the desired taps and swipes and inputs the inferred text as virtual keyboard presses.

Finally, one important concern of the design of CuriousDroid was that it is agnostic to its environment. Since CuriousDroid does not require any prior knowledge of an app, can be deployed in any Android environment with minimal effort and without modifications to the Android operating system, and is highly performant, it is an ideal candidate for the integration in large-scale analysis sandboxes such as Andrubis as a replacement for the current state-of-the-art monkey.

Table 2: Dynamic app behavior uncovered by CuriousDroid. By comparison, Andrubis used with monkey produced *no* behavior in each category.

Category	# Apps	# Triggered	% Triggered
Sending SMS	6,871	440	6.40%
Dynamic Code Loading	8,371	358	4.28%
Native Code Loading	7,669	1,945	25.36%
Networking	7,134	2,650	37.15%

3.2.3 Results

For the evaluation we integrated CuriousDroid in 36 virtual instances of Andrubis, running at Northeastern University in order to not interfere with regular operations. This experimental setup achieved a throughput of 5,500 apps per day allowing us to evaluate a dataset of 51,571 apps.

We evaluated CuriousDroid on this dataset in terms of activity (i.e. the number of different user interface screens covered) and behavior coverage (i.e. the number of interesting behaviors elicited) compared to random user interface interactions from the monkey. Apps in this dataset belong to one of the following two categories:

Borderline Apps. Apps that received a "borderline" classification from Marvin, i.e. apps our classification algorithm could not classify as either malicious and benign, possibly due to a lack of expressive dynamic features.

Missing Dynamic Behaviors. Apps that were flagged by static analysis as being capable of interesting behavior in four categories (sending SMS, dynamically loading DEX code, loading native code, and performing network activity), but that did not show this behavior during runtime in Andrubis. Related work has shown that these behaviors are potentially (and often) indicative of malware [42].

For the first category of apps our evaluation demonstrated that the increase in dynamic features from an average of 21.6 per application to 30.8 per application with CuriousDroid indeed improved classification results significantly and allowed relabeling from *unknown* to *benign* for many apps while also reclassifying a group of apps from *unknown* to *malicious*.

For the second category CuriousDroid allowed us to trigger behavior for each category of apps whereas monkey did not trigger this behavior in any of the apps. Table 2 summarizes our results and shows that CuriousDroid elicited interesting behaviors in up to 37.15% of apps in the case of networking and 25.26% in the case of dynamic code loading.

Finally, the integration of CuriousDroid with Andrubis for our evaluation showed, that our methods are indeed suitable for large-scale analysis and thus will be available in the public version of Andrubis in the near future.

3.3 ReCon

ReCon is a project by Jingjing Ren, PhD student at Northeastern University, Ashwin Rao, post-doctoral researcher at the University of Helsinki, and Arnaud Legout, post-doctoral researcher at INRIA Sophia Antipolis under the supervision of Prof. David Choffnes. I contributed to this project by providing insights on the workings of host-based detection of information leaks and support on the dynamic analysis of privacy leaks through TaintDroid as integrated in Andrubis. Furthermore, I was able to gather and contribute a dataset from Andrubis for training and evaluating ReCon's classifier.

A paper on this project is currently in the final stages of evaluation and being prepared for submission to the Internet Measurement Conference (IMC).

3.3.1 Motivation

In our study on the behavior of Android apps we found up to 49.78% of recent apps leaking personally identifiable information (PII), drastically increasing from only 13.45% in 2010 [20]. Reasons for this are that information uniquely identifying a device and user, such as IMSI, IMEI, device ID, or phone number, and also the current location of a user, are of great value to advertisers to track a users actions and tailor the displayed advertisements to the interests of a specific user. In addition, malicious apps are interested in exfiltrating valuable information such as contacts, credentials, usernames, passwords, and credit card numbers.

Tools monitoring PII leaks usually fall into two categories: There are dynamic analysis methods, such as TaintDroid [13], that monitor data flow from sensitive information sources to the network, but require modification of the Android operating system. Static analysis tools, such as AppIntent [39], identify PII leaks based on an app's code but cannot defend against dynamic code loading at run time.

In addition, these approaches do not address the problem of which PII leaks should be blocked (and how), a problem that is difficult to address in

practice [17]. The classification of data leaks is not always black and white: For example, an app recommending restaurants surrounding the current position of the user has a legitimate reason to send the user's position to its server, while the same behavior in an ad library in that same app that uses the location purely for tracking, is undesirable. To this end, user feedback is invaluable when automatically classifying PII leaks.

The goal of ReCon is to provide a user-friendly interface that allows users to monitor data that is leaked by apps installed on their device and decide whether they want to allow that connection, replace the leaked information with fake information, or block the connection altogether.

3.3.2 System Overview

Our key observation is that since PII leaks must occur over the network, the network is the ideal vantage point to monitor them and also gives us the opportunity to modify flows should a data leak be detected. Thus, ReCon uses a software middlebox atop the Meddle[7] [29] platform to man-in-the-middle, i.e. redirect and inspect, the network traffic. Since network traffic increasingly is encrypted, we use SSL bumping [3] to decrypt and inspect SSL flows, however only during our controlled experiments where no real user traffic is intercepted. Figure 4 shows an overview of the deployment of ReCon with the traffic either redirected through our Meddle instance in the cloud (currently Amazon EC2) or a private trusted instances. Users concerned about the privacy of their sensitive traffic can deploy their private Meddle instance and e.g. redirect plaintext traffic to our EC2 deployment and SSL traffic to their trusted instance.

Sine we do not know the contents of the PII in advance, our key challenge is how to detect the PII leaks in the redirected network flows. Creating signatures for every possible app is infeasible, hence we experimented with several machine learning techniques to automatically build a model of PII leaks that we can apply to data from arbitrary users and apps. Training data for this model can for example come from the PII leaks detected through TaintDroid in Andrubis since we already have a large database of over one million apps to start with,with almost half of them leaking data to the network.

Finally, ReCon visualizes the detected PII leaks for users and shows them how their data is shared with third parties. Users then can validate

[7] `http://www.meddle.mobi`

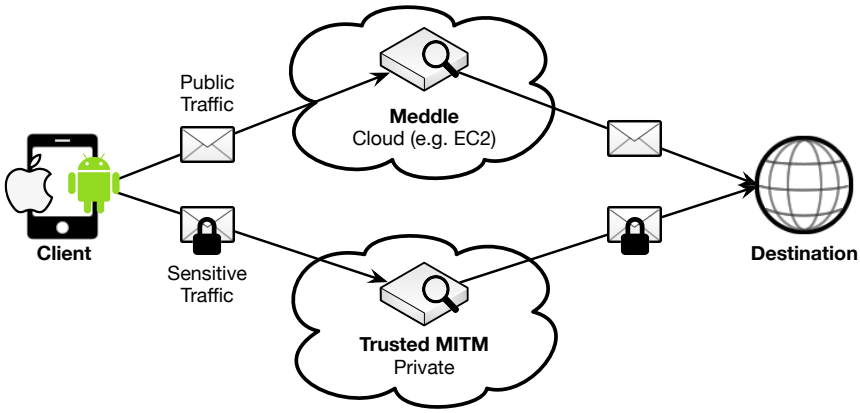

Figure 4: ReCon System Overview.

ReCon s suspected PII, as well as replace the PII with other text (or nothing) for future flows between the app and tracker. We can then feed their feedback about the correctness of our predictions back into our machine learning classifier in order to improve future classification results.

3.3.3 Results

We started by analyzing the most popular apps from the Apple iTunes App Store, the Google Play Store as well as the alternative market `appsapk.com`. In this controlled experiment we intercepted both HTTP and encrypted HTTPS traffic. We found that the IMEI and device ID that can be used to track a user's behavior are the most frequently leaked information. Furthermore we found apps leaking email addresses, a user's contacts and passwords in the cleartext.

In an ongoing IRB-approved user study we collected feedback from users at Northeastern University, currently for 122,176 HTTP flows, in which ReCon detected a large number of PII leaks, and classified them correctly according to user feedback. By far the most often leaked information are device identifiers and the location. Again, in this study ReCon also found the leakage of sensitive information such email addresses and even passwords in the clear.

We will open-source our code in order to allow users to deploy their own trusted instances of Meddle in order to promote the usage of ReCon. By presenting the data leaks of the apps installed on their devices and attributing leaks to trackers, we hope to raise awareness of users and enable them to modify or remove the leaked PII.

3.4 Untitled iOS Security Project

This project is currently my main research project and is carried out under the supervision of Prof. William Robertson and Prof. Engin Kirda from Northeastern University as well as Prof. Manuel Egele from Boston University.

As this project is work in progress we have not scheduled a publication target yet.

3.4.1 Overview

In contrast to Android security the field of iOS security has hardly been studied so far. Reasons are the closed nature of the operating systems that hinders dynamic analysis and the closely guarded app distribution through the iTunes App Store. Still, related work has shown that it is possible to infect iOS devices with malicious apps [35, 36, 41] and that PII leaks is not only a concern for Android users [12]. Consequently, the security of iOS devices deserves a closer look, especially the security implications of new feature additions in the latest version.

With the release of iOS8 in September 2014, Apple opened the strict separation between apps and introduced the new concept of extensions [6]. Extensions allow apps to share data through widgets, photo editing, sharing and action extensions. Naturally, this also opens up ways for attackers to take advantage and extract information from apps. So far, the only way to communicate between on iOS apps was through URL handler, a process that is not without its vulnerabilities and can for example leak a user's authentication credentials [34].

As it is important to have an extensive (ideally inclusive) dataset to base our evaluation on, we started with the collection of as many iOS apps as possible. For this we implemented a crawler to download all available free iOS apps from the iTunes App Store. While Apple reports 1.2 million apps [27] in the store, our crawler discovered only 981,949 apps, with 668,370 of them being free and thus candidates for download and further evaluation.

In the meantime, while this crawler is building our dataset, we started looking into data leaks from a different perspective: Apps might also unknowingly leak more information to the user, and potential attackers, than necessary. So far, we found evidence of this behavior in an airline apps transmitting more information on the upgrade lists than intended for the

standard user or a quiz app already transmitting the correct answers with its questions. Currently, we are exploring this topic further and are in the process of automating the discovery of this behavior. Challenges include first of all the interception of network traffic from the server to the client app, where we can learn from our experience with ReCon. Secondly, we need to distinguish the superfluously transmitted information, for which we are exploring taint tracking, fine-grained method tracing of the Android user interface and user interface stimulation, and can thus use our experience from Andrubis and CuriousDroid.

4 Conclusion

I would like to thank the Marshall Plan Foundation again for supporting me with a scholarship and the Secure Systems Lab at Northeastern University for hosting me. The internship proved to be very productive, with two papers currently being prepared for submission and another one following shortly. The projects I worked on closely relate to the topic of my PhD thesis of malware analysis and mobile security: Automated user interface interaction for Android app analysis sandboxes, controlling privacy leaks in mobile network traffic, and general iOS security and privacy issues. Furthermore, the internship provided valuable exchange of ideas with fellow researchers and served as the basis for future collaborations. I am looking forward to continuing working on our shared projects and hopefully visiting the Secure Systems Lab again in the future.

References

[1] ForeSafe Mobile Security. `http://www.foresafe.com`.

[2] Joe Sandbox Mobile. `http://www.joesecurity.org/joe-sandbox-mobile`.

[3] Squid-in-the-middle SSL Bump. `http://wiki.squid-cache.org/Features/SslBump`.

[4] VisualThreat. `http://www.visualthreat.com`.

[5] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. In *International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2013.

[6] Apple Developers. App Extensions. `https://developer.apple.com/app-extensions/`.

[7] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck. Drebin: Efficient and Explainable Detection of Android Malware in Your Pocket. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2014.

[8] T. Azim and I. Neamtiu. Targeted and Depth-First Exploration for Systematic Testing of Android Apps. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, 2013.

[9] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. Camtepe, and S. Albayrak. An Android Application Sandbox System for Suspicious Software Detection. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software (MALWARE)*, 2010.

[10] W. Choi, G. Necula, and K. Sen. Guided GUI Testing of Android Apps with Minimal Restart and Approximate Learning. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, 2013.

[11] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger. ANANAS - A Framework For Analyzing Android Applications. In *Proceedings on*

the 1st International Workshop on Emerging Cyberthreats and Countermeasures (ECTCM), 2013.

[12] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS)*, 2011.

[13] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2010.

[14] F-Secure. Threat Report H2 2013. `http://www.f-secure.com/static/doc/labs_global/Research/Threat_Report_H2_2013.pdf`, March 2014.

[15] Google. Android Developers: UI/Application Exerciser Monkey. `http://developer.android.com/tools/help/monkey.html`.

[16] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. RiskRanker: Scalable and Accurate Zero-day Android Malware Detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[17] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These Aren't the Droids You'Re Looking for: Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, 2011.

[18] IDC. Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013. `http://www.idc.com/getdoc.jsp?containerId=prUS24676414`, February 2014.

[19] M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis. In *Proceedings of the 39th Annual International Computers, Software & Applications Conference (COMPSAC)*, 2015 (to appear).

[20] 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 *Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2014.

[21] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, and S. Ioannidis. AndRadar: fast discovery of android applications in alternative markets. In *Proceedings of the 11th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, 2014.

[22] H. Lockheimer. Android and Security. `http://googlemobile.blogspot.com/2012/02/android-and-security.html`, February 2012.

[23] A. MacHiry, R. Tahiliani, and M. Naik. Dynodroid: An Input Generation System for Android Apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, 2013.

[24] McAfee Labs. McAfee Threats Report: Second Quarter 2013. `http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q2-2013.pdf`, August 2013.

[25] C. Mulliner. Dynamic Dalvik Intrumentation (DDI). `https://github.com/crmulliner/ddi`.

[26] S. Neuner, V. van der Veen, M. Lindorfer, M. Huber, G. Merzdovnik, M. Mulazzani, and E. Weippl. Enter Sandbox: Android Sandbox Comparison. In *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, 2014.

[27] S. Perez. iTunes App Store Now Has 1.2 Million Apps, Has Seen 75 Billion Downloads To Date. `http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/`, June 2014.

[28] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna. Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium (NDSS)*, 2014.

[29] A. Rao, A. M. Kakhki, A. Razaghpanah, A. Tang, J. Sherry, S. Wang, P. Gill, A. Legout, A. Krishnamurthy, A. Mislove, and D. Choffnes. Using the Middle to Meddle with Mobile. Technical Report NEU-CCS-2013-12-10, Northeastern University, 2013.

[30] V. Rastogi, Y. Chen, and W. Enck. AppsPlayground: Automatic Security Analysis of Smartphone Applications. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2013.

[31] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann. Mobile-sandbox: Having a Deeper Look into Android Applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC)*, 2013.

[32] V. Svajcer. Sophos Mobile Security Threat Report. `http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-mobile-security-threat-report.ashx`, 2014.

[33] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro. CopperDroid: Automatic Reconstruction of Android Malware Behaviors. In *Proceedings of the 21st Annual Network & Distributed System Security Symposium (NDSS)*, 2015.

[34] R. Wang, L. Xing, X. Wang, and S. Chen. Unauthorized Origin Crossing on Mobile Platforms: Threats and Mitigation. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[35] T. Wang, Y. Jang, Y. Chen, S. Chung, B. Lau, and W. Lee. On the Feasibility of Large-Scale Infections of iOS Devices. In *Proceedings of the 23rd USENIX Security Symposium*, 2014.

[36] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee. Jekyll on iOS: When Benign Apps Become Evil. In *Proceedings of the 22nd USENIX Security Symposium*, 2013.

[37] L. Weichselbaum, M. Neugschwandtner, M. Lindorfer, Y. Fratantonio, V. van der Veen, and C. Platzer. Andrubis: Android Malware Under The Magnifying Glass. Technical Report TR-ISECLAB-0414-001, Vienna University of Technology, 2014.

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

[39] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang. AppIntent: Analyzing Sensitive Data Transmission in Android for Privacy Leakage Detection. In *Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

[40] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, and W. Zou. SmartDroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications. In *Proceedings of the 2nd ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2012.

[41] M. Zheng, H. Xue, Y. Zhang, T. Wei, and J. C. Lui. Enpublic Apps: Security Threats Using iOS Enterprise and Developer Certificates. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security ASIACCS*, 2015.

[42] Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.