Vulnerability analysis on Android framework level

Heng Chao Li,
Srinath Nadimpalli,
Visvanathan Thothathri,
Xu Yan
Department of Computer Science and Engineering,
Texas A&M University

October 1, 2013

Abstract

Increased adoption of smartphones makes them a likely target for malicious attacks. Current research in Android framework concentrates on the applications user-space and on the permission model in Android. Our work focuses more on the lower levels of the Android OS.

List of Figures

List of Tables

1 Introduction

Smart-phones are being increasingly adopted by consumers given the slew of functionalists they offer and their affordability. Gartner [1] estimates show that 225 million smart-phone units were sold in Q2, 2013 [2]. This approximates to about 0.5 billion units being sold per year. [2] also show that android phones were the most bought smart-phones till Q2, 2013. The estimates show that Q2 worldwide market share was about 178000 units. On the other hand, in 2011, it was estimated that Windows OS install base is around 1.25 billion units [3]. Given the rapid growth of android phones, it is estimated that android phone sales will cross the Windows user base by Q1, 2014 [4].

1.1 Motivation

This wide spread adoption of Android makes it a favorite target for malicious content authors to infect and attack users. Hence protecting end-users irrespective of the device they use becomes paramount. Amongst the different ways of protecting, we take the offensive approach where we identify vulnerabilities in the Android environment. Identifying such vulnerabilities and disclosing them ethically prevents abuse by malware content authors.

1.2 Problem Statement

Figure 1 shows the architecture diagram of Android OS [5]. There are 5 different layers in Android OS. The topmost layer is Applications. All user applications exist in this layer of the OS. The next layer is the application framework layer that applications directly interact with. This layer provides important resource and service management. The third layer is android runtime layer. It consists of core java libraries and Dalvik virtual machine. The fourth layer contains a few third party libraries such as OpenGL, SQLite etc... The lowest layer is Linux Kernel layer, which contains device drivers to provide interfaces for higher applications to access hardware resources. In summary, we can have a brief view of the five layers below:

A lot of focus in the current literature has been on the application level and the permission model of Android. Examples include Intent fuzzer, built-in Monkey UI fuzzer, Dynodroid, and so forth. Our motivation is to develop an Android Framework Layer fuzzing tool that can be used to detect vulnerabilities in the lower levels of Android Platform.

Our fuzzer aims to fuzz a few modules in application framework to see which input will lead to crash of that module and investigate the reason for

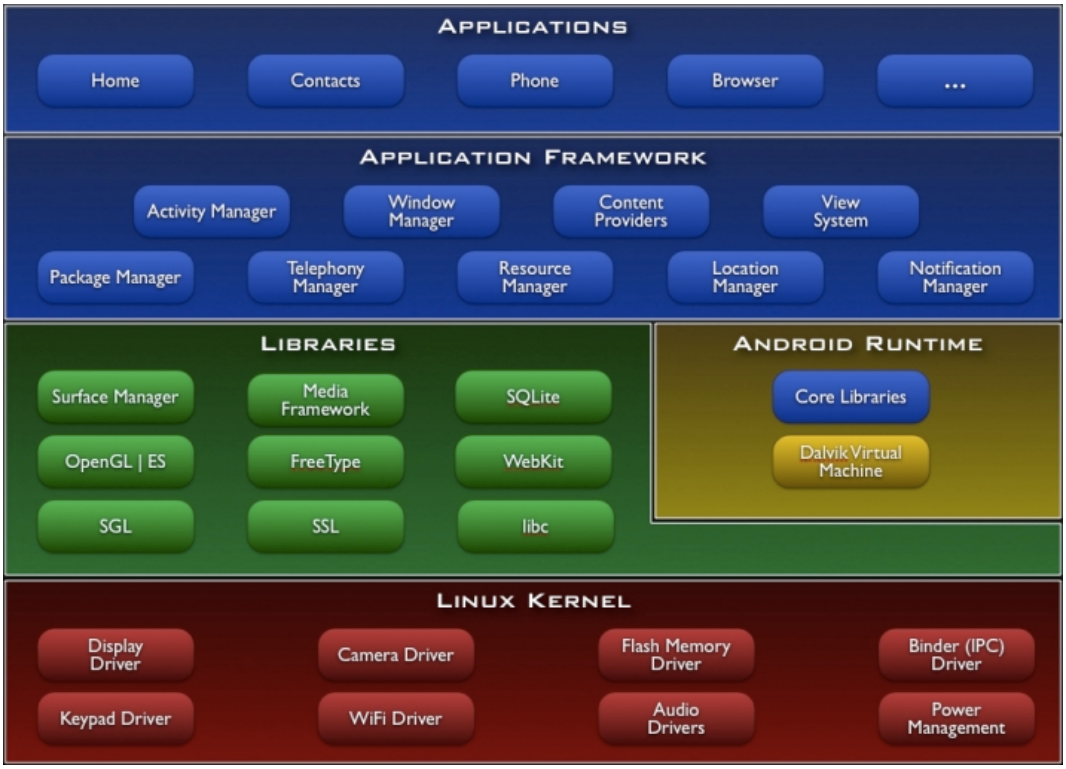

Figure 1: Android Architecture

the crash. The idea is similar to black box testing - we do not worry about the concrete implementation of different modules in application framework. We run all possible input to the module to test whether it is reliable enough. If time permits, we will extend to fuzz Android Runtime level and propose ways of hardening the system.

2 Proposal

This section introduces the existing control flow in the unmodified android OS. It also shows the position of our fuzzer and its interaction with other modules.

Figure 2 shows how the control flows from the user to the application framework.

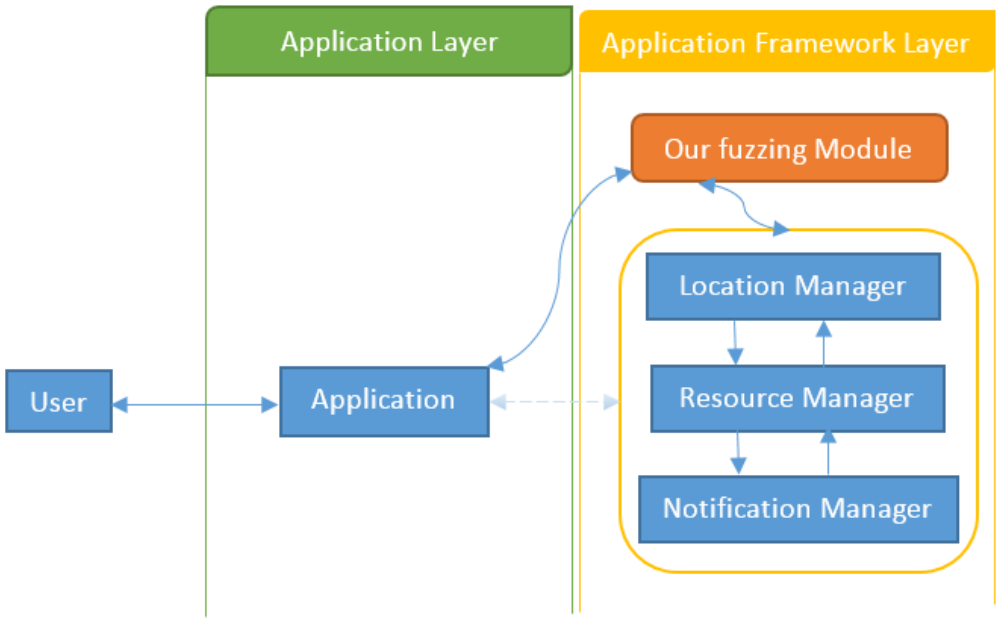

Figure 2: Proposed Fuzzer

When an user provides an input (touch input, keypad input, button input...), the input is received by the application which then sends it to the application framework. This framework processes the request and sends it to the layers below for more processing. In the Figure 2 shows a bidirectional line between the user and application. It also shows a bidirectional dotted line between the application and the framework component.

Our proposed fuzzer module will be located in the application framework layer. It will hook on a call being made to the framework modules, receives the input being sent and fuzzes it. Our custom made fuzzer will be able to generate a lot of malformed input for the module based on a known set of valid input data. The target module will receive our fuzzed input and will respond to the fuzzed input. This response of the target is then received

by our fuzzer module. We will log all the inputs and the outputs for understanding the processing and to isolate which input sequence causes the target module to hang/crash. Once we complete logging, we send the output to the application. Thus we intercept the incoming requests and the outgoing responses to identify the input sequence breaking the target module. Once we identify such vulnerability we will focus on exploiting it to compromise the system/user.

3 Literature Survey

This section presents the existing work in the literature regarding fuzzing and vulnerability analysis on Android framework.

Vidas et al [7] analyze the Android Security Model: Android was designed with security issues in mind. It has privilege separation, permission model to help protecting user's privacy and resources. However, the scheme is not working very well due to users' limited understanding of the permission model and the complicated implementation of security schemes in Android. Because of the fast development of both official and unofficial Android Market, plus the need for better features in Android platform, increasing new vulnerabilities become available facing different kinds of exploitation (remote exploitation, unprivileged attack, physical attack and so forth). Although from users' perspectives, we can do authenticated download, and from Google, manufacturers and carriers' perspectives, we can enhance the patch cycle, the key point is to detect these new vulnerabilities in android platform as soon as possible.

DynoDroid [6] introduces a input fuzzer for fuzzing the input being provided to applications. It employs the Android Debug Bridge (adb) and the monkey UI generator in providing fuzzed inputs to the application being tested. It is modeled in a *observer-selector-executor* model. Dynodroid observes what state the application is in and decides the system and UI inputs to be generated. The selector module selects one of the inputs and passes it on to the executor for execution. The state of the system changes again and the observer observes for possible set of inputs to be generated. Dynodroid responds to just the intents that are declared in the app's manifest file. It does not consider the generic system intents that may be launched during the app's run-time. The results shown in the paper go over how this model is effective in identifying vulnerabilities in android applications using fuzzing. However, it does not talk anything about fuzzing at any of the lower levels of the Android OS framework such as application framework, dalvik virtual machine, application run-time and the Linux kernel.

Lu et al [8] raises concern about the quality of innumerable Android apps in recent years. It claims that apps may be insecure due to poor engineering practises. Such vulnerabilities can severally undermine users' security and privacy. The paper analyses the component hijacking vulnerability. Permission leakage, intent spoofing and unauthorised access of data belong to the component hijacking vulnerabilities. The authors propose a tool called CHEX to automatically vet Android apps for such vulnerabilities. CHEX analyses the data-flow paths and detects possible hijack-enabling flows. It is to be noted that this analysis is made on the applications layer.

DroidScope [9] is an android analysis platform built on top of QEMU (open source hypervisor). It uses three-tiered APIs - hardware, OS and dalvik virtual machine. In Android, every app is assigned its own userid (UID) at installation and groupid (GID) to request permissions. UID and GID together control the network and file systems. DroidScope looks at system calls, memory maps (especially in Ice Cream Sandwich which has address space randomization enabled), thread information and running processes for OS level knowledge. To collect DALVIK level knowledge it looks at dalvik instructions, machine state, java objects. 4 event based analysis tools implemented as plugins-native instruction tracer, dalvik instruction tracer,API tracer, taint tracker The API tracer traces malware activity at API level including interactions with run-time environment and system calls. The native instruction tracer and dalvik instruction tracer-records machine level and dalvik bytecode instructions. The taint tracker tracks leakage of sensitive information such as IMEI and IMSI numbers. The advantage of virtualization based analysis is that it is isolated and confined. All analysis components including malware have more privilege and hence more details are revealed about them. On the other hand, emulation-aware malware detect if they are running within an emulated environment and evade analysis by staying dormant or simply crashing themselves. Sometimes, the environment is not exactly similar to that of a mobile phone with cross app-functionality or necessarily similar hardware constraints.

4 Project Plan

This section identifies the tasks involved in the project and discusses about the split-up of tasks and the time-line.

4.1 Tasks

- Download android source code
- Compile android source code
- Build a custom module in the application framework
- Intercept requests from the application to the target module and redirect it to our custom fuzzer
- Identify interaction between the modules and application
- Make the module hook into a particular API of the target module
- Understand the target module's data, function members and the valid data sets that the module accepts
- Generate valid malformed data while maintaining the integrity of the data being sent
- Identify vulnerabilities

4.2 Milestones

The table below shows the different tasks involved and the milestones for each of the tasks

Table 1: Task time-line

Dates from - to	Task to be completed
10.1 -10.8	Download, Compile and run emulator on Ubuntu. Understand the communication between Apps and Android Framework Modules
10.8 -10.15	Build our custom module in Android Framework Layer
10.15 -10.22	Intercept the requests to a specified AF module from apps and show it in our custom module
10.22 -10.29	Understand the target module's data, function members and the valid data sets that the module accepts
10.29-11.5	Generate random, valid malformed data in our custom module
11.5-11.12	Send these inputs to target module and check vulnerabilities from Log
11.12-11.21	Write final paper

References

[1] Technology Research — Gartner Inc. [Online] Available: http://www.gartner.com/

[2] Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time [Online] Available: http://www.gartner.com/newsroom/id/2573415

[3] Right Now, There Are 1.25 Billion Windows PCs Worldwide. [Online] Available: http://www.businessinsider.com/right-now-there-are-125-billion-windows-pcs-worldwide-2011-12.

[4] Android installed base might outgrow total Windows devices in early 2014. [Online] Available: http://www.phonearena.com/news/Android-installed-base-might-outgrow-total-Windows-devices-in-early-2014_id33310

[5] Android System Architecture. [Online] Available: http://developer.android.com/images/system-architecture.jpg

[6] A. Machiry, R. Tahiliani and M. Naik. Dynodroid: An input generation system for Android Apps. *ESEC/SIGSOFT FSE, page 224-234. ACM, (2013)*

[7] Timothy Vidas , Daniel Votipka , Nicolas Christin, All your droid are belong to us: a survey of current android attacks, Proceedings of the 5th USENIX conference on Offensive technologies, p.10-10, August 08, 2011, San Francisco, CA

[8] Long Lu , Zhichun Li , Zhenyu Wu , Wenke Lee , Guofei Jiang, CHEX: statically vetting Android apps for component hijacking vulnerabilities, Proceedings of the 2012 ACM conference on Computer and communications security, October 16-18, 2012, Raleigh, North Carolina, USA

[9] Lok Kwong Yan , Heng Yin, DroidScope: seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis, Proceedings of the 21st USENIX conference on Security symposium, p.29-29, August 08-10, 2012, Bellevue, WA