CLOUD INTEGRATED LOGGING OF ANDROID SYSTEM EVENTS

DISSERTATION

submitted by

JAMSHEED K CB.EN.P2CYS13010

in partial fulfillment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

CYBER SECURITY

TIFAC-CORE IN CYBER SECURITY

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

JULY 2015

CLOUD INTEGRATED LOGGING OF ANDROID SYSTEM EVENTS

DISSERTATION

submitted by

JAMSHEED K CB.EN.P2CYS13010

in partial fulfillment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

CYBER SECURITY

Under the guidance of

Prof. Prabhaker Mateti
Associate Professor
Computer Science and Engineering
Wright State University
USA

TIFAC-CORE IN CYBER SECURITY

AMRITA SCHOOL OF ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112

JULY 2015

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING,COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that this dissertation entitled **"CLOUD INTEGRATED LOGGING OF ANDROID SYSTEM EVENTS"** submitted by **JAMSHEED K, Reg.No:CB.EN.P2.CYS13010** in partial fulfillment of the requirements for the award of the **Degree of Master of Technology** in **CYBER SECURITY** is a bonafide record of the work carried out under my guidance and supervision at Amrita School of Engineering.

Dr. Prabhaker Mateti — (Supervisor)

Dr. M. Sethumadhavan — (Professor and Head)

This dissertation was evaluated by us on .....

INTERNAL EXAMINER

EXTERNAL EXAMINER

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF ENGINEERING,COIMBATORE -641 112

TIFAC-CORE IN CYBER SECURITY

DECLARATION

I, JAMSHEED K, (Reg.No: CB.EN.P2.CYS13010) hereby declare that this dissertation entitled **"CLOUD INTEGRATED LOGGING OF ANDROID SYSTEM EVENTS"** is a record of the original work done by me under the guidance of **Prof. Prabhaker Mateti**, Associate Professor, Department of Computer Science and Engineering, Wright State University, Dayton, Ohio and this work has not formed the basis for the award of any degree / diploma / associateship / fellowship or a similar award, to any candidate in any University, to the best of my knowledge.

Place : Coimbatore

Date : Signature of the Student

COUNTERSIGNED

Dr. M. Sethumadhavan

Professor and Head, TIFAC-CORE in Cyber Security

ACKNOWLEDGMENTS

At the very outset, I would like to give the first honors to **Amma, Mata Amritanandamayi Devi** who gave me the wisdom and knowledge to complete this dissertation under her shelter in this esteemed institution.

I express my gratitude to my guide, **Prof. Prabhaker Mateti**, Associate Professor, Computer Science and Engineering, Wright State University, USA, for his valuable suggestion and timely feedback during the course of this dissertation.

I would like to thank **Dr M. Sethumadhavan**, Professor and Head of the TIFAC-CORE in Cyber Security, for giving me useful suggestions and his constant encouragement and guidance throughout the progress of this dissertation.

I express my special thanks to my colleagues who were with me from the starting of the dissertation, for the interesting discussions and the ideas they shared. Thanks also go to my friends for sharing so many wonderful moments. They helped me not only enjoy my life, but also enjoy some of the hardness of this dissertation.

In particular, I would like to thank **Mr. Praveen K**, Assistant Professor, TIFAC-CORE in Cyber Security, Amrita Vishwa Vidyapeetham, Coimbatore and **Mrs. Jevitha K. P**, Assistant Professor, TIFAC-CORE in Cyber Security, Amrita Vishwa Vidyapeetham, Coimbatore, for providing me with the advice, infrastructure and all the other faculties of TIFAC-CORE in Cyber Security and all other people who have helped me in many ways for the successful completion of this dissertation.

Abstract

Smartphones have become an unavoidable part of our personal and work life. This has made them an obvious target for attacks. Android, being the market leader in smartphone operating systems, is thus most targeted by the attackers. The open source nature of Android adds to the possibility of more and more sophisticated attacks to be devised. Even though considerable has been done to prevent attacks from the application level, little has been done to audit them from the operating system level. This thesis proposes a model in which an Android OS Framework level audit is done to gather information on system events, so that it could provide malware analysis in the future. For that, a new system level service is added which manages the auditing process along with event handlers and file loggers that actually handles the auditing part. To generate events relevant to android, hooks are inserted in several parts of the system in the framework level and in the application libraries.

Keywords: Android, Audit, Framework, Logging, Events

Contents

Abbreviations

ADB	**A**ndroid **D**ebug **B**ridge
AOSP	**A**ndroid **O**pen **S**ource **P**roject
AVD	**A**ndroid **V**irtual **D**evice
LAuS	**L**inux **Au**dit **S**ubsystem
NDK	**N**ative **D**evelopment **K**it
SDK	**S**oftware **D**evelopment **K**it

List of Figures

List of Tables

Chapter 1

Introduction

Developed by the Open Handset Alliance (visibly led by Google), Android is a widely anticipated open source operating system for mobile devices that provides a base operating system, an application middleware layer, a Java software development kit (SDK), and a collection of system applications. Although the Android SDK has been available since late 2007, the irst publicly available Android-ready G1 phone debuted in late October 2008. Since then, Androids growth has been phenomenal. Currently Android powers hundreds of millions of mobile devices in more than 190 countries around the world. It's the largest installed base of any mobile platform and growing fastevery day another million users power up their Android devices for the first time and start looking for apps, games, and other digital content[Enck et al., 2009].

Building on the contributions of the open-source Linux community and more than 300 hardware, software, and carrier partners, Android has rapidly become the fastest-growing mobile OS. Androids openness has made it a favorite for consumers and developers alike, driving strong growth in app consumption. Android users download more than 1.5 billion apps and games from Google Play each month.With its partners, Android is continuously pushing the boundaries of hardware and software forward to bring new capabilities to users and developers[Google].

Android applications make use of advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To protect that value, the platform must offer an application environment that ensures the security of users, data, applications, the device, and the network. Securing an open platform requires a robust security architecture and

rigorous security programs. Android was designed with multi-layered security that provides the flexibility required for an open platform, while providing protection for all users of the platform.

1.1 Motivation

Android has 78.0% market share of Smartphones shipped during Q1 2015[IDC, 2015]. This trend seems to be here for a while, and is predicted to stay for a while. This making Android the leading smartphone operating system in the market. Android is subjected to a large number of attacks with newer ones evolving every day. The need for withstand existing and future attacks is more important than mitigating each and every attack that is being discovered every day. Thus, we propose a self audit framework for Android to check itself for suspicious behaviour and solve them.

1.2 Problem Statement

The objective of the thesis is to evaluate current Android security level, analyse existing systems for auditing linux and Android systems, evaluate them, try to port them or make a new one inspired from good auditing models. Auditing is very important in the field of security, where logs and data provenance plays an important role. Auditing provides us with logs that will come handy in doing any kind of analysis that we do in the system, for purposes like malware analysis, forensics etc.

1.3 Scope of the Thesis

Android Security Report in the recent times shows us that Android is one of the most targeted platforms recently. In 2014, the Android security team provided patches for 41 Moderate, 30 High, and 8 Low Severity vulnerabilities. There are about 26 publicly known local privilege escalation vulnerabilities. Many of these vulnerabilities had patches available prior to 2014, but there are devices that have not been patched for all publicly known vulnerabilities.

No matter how many patches are released, most of them wont make it to the end user. Further, new vulnerabilities are being discovered very often. So, just releasing patches would not do much good against new and upcoming vulnerabilities. What we need is a system that could track malicious patterns in real time or within a reasonable amount of delay. As a first step towards that system, our thesis aims to bring an auditing subsystem that could help a lot in logging relevant system related information and thus help analysis by a very large extent. We expect our system to be a part of something big that would help detect security issues to a good extent.

1.4 Organization of the Report

Chapter 2 provides a background on Android security. It discussed the status of Android security, terms relevant to the thesis, status of the Android Open Source Project, and the Android Platform overview. Chapter 3 discusses our proposed system, its architecture , different parts and its functionality. It gives a good idea about what this thesis is. Chapter 4 provides information on related works that was done on this or related domains in the past years. Chapter 5 evaluates the thesis, how this work differs from the previous work, impact on performance and battery consumption and the details of the actual code written. Chapter 6 concludes the thesis report and discusses what could be done in the future.

Chapter 2

Background

2.1 Status of Android Security

Android Security Report 2014 [Google, 2015] shows us that Android is one of the most targeted platforms recently. In 2014, the Android security team provided patches for 41 Moderate, 30 High, and 8 Low Severity vulnerabilities. There were no critical vulnerabilities found in 2014. To provide OEMs with opportunity to patch prior to disclosure, patches are provided to partners but not publicly disclosed until the next API update to AOSP. At that time, patches are released to open source. But most patches never reached the end user, as manufacturers are reluctant to provide updates to budget smartphones. There are about 26 publicly known local privilege escalation vulnerabilities. Many of these vulnerabilities had patches available prior to 2014, but there are devices that have not been patched for all publicly known vulnerabilities.

In 2014, there were a number of high-profile vulnerabilities affecting implementations of SSL. The most significant vulnerability affecting most platforms, Heartbleed affected Android 4.1.1 devices. A vulnerability affecting Android 4.4 and earlier received broad attention following disclosure at Black Hat. This vulnerability was named FakeID and was formally identified as Android-13678484.

In another report by Fortinet [Apvrille, 2015], which computed statistics on 790000 Android malware found that:

- 1.7% of malware use root / superuser / mods tools

- 43% of malware send SMS messages. This includes cases of SMS Fraud, but also spyware which forward incoming SMS to another phone number.
- 56% of malware implement a SMS receiver, i.e a mechanism notifying them when there is an incoming SMS message. This is typically how spyware read and process incoming SMS.
- 20% actually retrieve the current geolocation.
- 8.6% malware ask to be notified whenever an outgoing call is about to be placed

Those percentages show that Spyware and SMS fraud - whether they are declining or not - are still a very important issue. de Ponteves and Apvrille, 2013 says that besides malware, all adkits leak private data, and often through insecure channels.

2.2 What is Provenace?

Data provenance documents the inputs, entities, systems, and processes that influence data of interest, in effect providing a historical record of the data and its origins. The generated evidence supports essential forensic activities such as data-dependency analysis, error/compromise detection and recovery, and auditing and compliance analysis. Scientific research is generally held to be of good provenance when it is documented in detail sufficient to allow reproducibility. Scientific workflows assist scientists and programmers with tracking their data through all transformations, analyses, and interpretations. Data sets are reliable when the process used to create them are reproducible and analyzable for defects. Current initiatives to effectively manage, share, and reuse ecological data are indicative of the increasing importance of data provenance.Within computer science, informatics uses the term 'provenance' to mean the lineage of data, as per data provenance, with research in the last decade extending the conceptual model of causality and relation to include processes that act on data and agents that are responsible for those processes. Semantic web standards bodies, such as the World Wide Web Consortium, have recently (2014) ratified a standard for provenance representation in known as PROV which draws from many of the better known provenance representation systems that preceded it, such as the Proof Markup Language and the Open Provenance Model.

2.3 A Quick Overview of Android AOSP Source Code

The Android Open Source Project (AOSP) maintains a complete software stack to be ported by OEMs and other device implementors and run on their own hardware. To maintain the quality of Android, Google has contributed full-time engineers, product managers, user interface designers, quality assurance testers, and all the other roles required to bring modern devices to market.

Accordingly, Google maintain a number of "code lines" to clearly separate the current stable version of Android from unstable experimental work. Google rolls the open source administration and maintenance of the Android code lines into the larger product development cycle.

The chart below depicts at a conceptual level how AOSP manages code and releases. It is referring to these as "code lines" instead of "branches" simply because at any given moment there may be more than one branch for a given "code line". For instance, when a release is cut, it may or may not become a new branch based on the needs of the moment.

1. At any given moment, there is a current latest release of the Android platform. This typically takes the form of a branch in the tree.
2. Device builders and contributors work with the current latest release, fixing bugs, launching new devices, experimenting with new features, and so on.
3. In parallel, Google works internally on the next version of the Android platform and framework according to the product's needs and goals. They develop the next version of Android by working with a device partner on a flagship device whose specifications are chosen to push Android in the direction we believe it should go.
4. When the "n+1"th version is ready, it will be published to the public source tree and become the new latest release.

Terms and Caveats:

- A release corresponds to a formal version of the Android platform, such as 1.5, 2.1, and so on. Generally speaking, a release of the platform corresponds to the version in the SdkVersion field of AndroidManifest.xml files and defined within frameworks/base/api in the source tree.

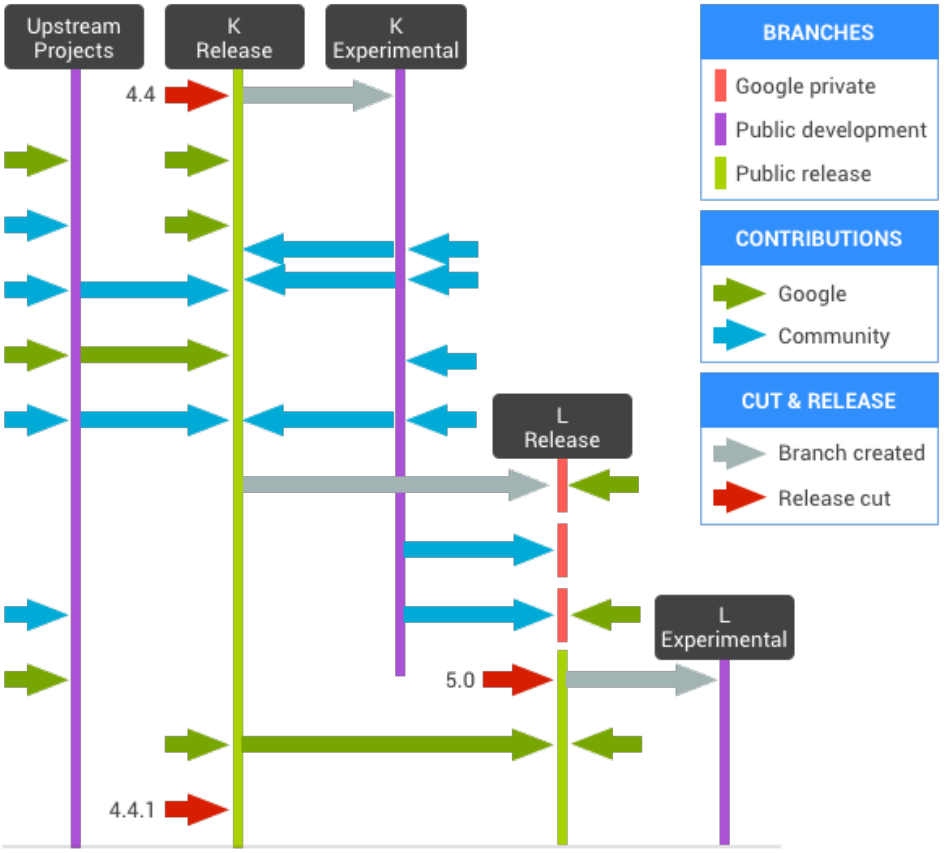

FIGURE 2.1: AOSP code and releases [Google]

- An upstream project is an open source project from which the Android stack is pulling code. These include obvious projects such as the Linux kernel and WebKit. Over time we are migrating some of the semi-autonomous Android projects (such as ART, the Android SDK tools, Bionic, and so on) to work as "upstream" projects. Generally, these projects are developed entirely in the public tree. For some upstream projects, development is done by contributing directly to the upstream project itself. See Upstream Projects for details. In both cases, snapshots will be periodically pulled into releases.

- At all times, a release code-line (which may actually consist of more than one actual branch in git) is considered the sole canonical source code for a given Android platform version. OEMs and other groups building devices should pull only from a release branch.

- "Experimental" code-lines are established to capture changes from the community so they can be iterated on with an eye toward stability.

- Changes that prove stable will eventually be pulled into a release branch. Note this applies only to bug fixes, application improvements, and other changes that do not affect the APIs of the platform.
- Changes will be pulled into release branches from upstream projects (including the Android "upstream" projects) as necessary.
- The "n+1"th version (that is, next major version of the framework and platform APIs) will be developed by Google internally. See About Private Codelines for details.
- Changes will be pulled from upstream, release, and experimental branches into Google's private branch as necessary.
- When the platform APIs for the next version have stabilized and been fully tested, Google will cut a release of the next platform version. (This specifically refers to a new SdkVersion.) This will also correspond to the internal code-line being made a public release branch, and the new current platform code-line.
- When a new platform version is cut, a corresponding experimental code-line will be created at the same time.

The table 2.1 shows the number of lines of code in Android 5.1.1 Release 5 branch.

SLOC	Directory	SLOC-by-Language (Sorted)
23233972	external	cpp=12402820, ansic=7027888, java=1323123, asm=688128, python=661444, sh=373797, xml=287805, perl=165910, objc=141593, cs=40419, fortran=26495, ruby=24458, yacc=19328, pascal=19011, lex=7239, tcl=6958, ml=5254, lisp=2554, exp=2468, php=2404, awk=2243, ada=1681, csh=368, sed=334, haskell=250
6964755	prebuilts	xml=5055564, ansic=1025346, cpp=521784, python=354359, sh=5437, perl=1758, ml=322, lisp=180, ruby=5
3183263	frameworks	java=1866631, cpp=733481, xml=408750, ansic=132921, asm=28393, python=5787, cs=3455, sh=1392, yacc=1134, lex=593, perl=418, pascal=286, sed=22

2167329	packages	java=987027, xml=962864, ansic=166349, cpp=48430, python=2304, sh=355
1482575	hardware	ansic=1096287, cpp=379207, java=2879, asm=2869, sh=542, perl=503, xml=171, awk=107, python=10
574178	libcore	java=562275, cpp=10803, xml=952, sh=147, ansic=1
462951	cts	java=416702, cpp=18083, xml=17452, python=6858, ansic=2238, sh=917, cobol=701
374021	device	cpp=157863, ansic=114770, xml=95989, sh=2815, java=1598, python=986
295547	development	ansic=119528, java=107901, xml=37402, cpp=23771, python=4464, sh=1556, asm=393, lisp=261, ruby=183, perl=88
270306	art	cpp=227101, java=30559, asm=8926, python=2171, sh=1466, ansic=83
256250	bionic	ansic=137415, cpp=75482, asm=40025, python=3068, sh=139, java=121
252505	ndk	cpp=142841, ansic=79830, sh=23343, python=1985, perl=1422, sed=933, asm=755, awk=729, yacc=407, java=228, xml=32
216224	sdk	java=172460, cpp=25737, xml=12125, ansic=4169, sh=1258, objc=197, python=168, lisp=110
183964	system	ansic=95987, cpp=73526, xml=6627, asm=3467, python=1932, sh=1418, java=1007
79928	dalvik	java=66571, cpp=8476, ansic=3312, sh=1194, awk=364, perl=11
62743	developers	java=43175, xml=19367, sh=201
22659	build	cs=9520, python=6199, cpp=2752, sh=1803, ansic=1378, java=1007
17657	bootable	ansic=13537, cpp=3460, sh=372, lex=112, yacc=105, python=71
7878	pdk	java=6227, xml=1310, python=335, cpp=6
3313	tools	java=3313
1843	libnativehelper	cpp=1843
765	docs	cs=497, xml=176, python=84, sh=8
450	abi	cpp=450

Table 2.1: AOSP Source Lines of Code

2.4 Libraries libc and bionic

Both GNU C Library (glibc) and Bionic Library (bionic) are standard libraries provide support for the C and C++ language but are used on different platforms. Where glibc is used within Linux distributions, bionic is used on Android based systems. Bionic is an adaptation of the BSD standard C library. Since Android is based on linux it would be logical to assume that they used the same C library, however due to the specifications of Android a new library was used. The word bionic means "having artificial parts", this directly shows that the bionic library consists of different parts: elements from glibc from linux, from the BSD library and uniquely written pieces of code specific for the bionic library. The reason for this assembly is because of the earlier mentioned Android specifications. Since Android is designed to run on mobile devices it tends to have several characteristics. Some of these characteristics are what determined to use bionic as the standard C/C++ library in these devices. The most profound characteristics are:

- Limited space and thus limited storage
- Lower CPU speed (around 30-50% lower compared to notebooks and desktops)
- Not fully open source

glibc was based on GNU Public License. It was a goal for Android to create an operating system that allows third party applications. Not only could people charge money for these applications, they wanted it so that the code of these third party apps didnt fall under the GNU Public License. A solution was found within the BSD licenses. These licenses do not fall under the copyleft rule and thus can have copyrights. This allows Android programmers to not be forced into open source when coding their software. Bionic posses some advantage over the standard glibc. A first advantage is the optimization of code due to the removal of all comments from the header files. This reduces the size of all those headers and helps with a speedy process in a storage-constricted space. Second advantage is that the GPL is kept out of user-space. This was not really an issue but with glibc moving towards LGPLv3 it might become an issue. The use of the BSD license prevents this as a whole and keeps the GPL license out of the userspace in Android. As stated earlier, since Android is running on space limited devices it helps immensely that bionic is around 200 kilobytes. Compared to glibc that is around 400 kilobytes. This is done by leaving out bloated code and reducing

unneeded parts or even removing them from the library. A last advantage for bionic can be found with the optimization for slower CPU(s). This means that changes were made to the implementation of pthread for lower clockspeeds. While this is an advantage for bionic, it can be seen as a disadvantage compared to other libraries as this library is specifically designed for lower CPU speeds[Devos, 2014].

2.5 Android Platform Security Architecture

Android is a modern mobile platform that was designed to be truly open. Android applications make use of advanced hardware and software, as well as local and served data, exposed through the platform to bring innovation and value to consumers. To protect that value, the platform must offer an application environment that ensures the security of users, data, applications, the device, and the network[Enck et al., 2009].

Securing an open platform requires a robust security architecture and rigorous security programs. Android was designed with multi-layered security that provides the flexibility required for an open platform, while providing protection for all users of the platform.

Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:

- Protect user data
- Protect system resources (including the network)
- Provide application isolation

To achieve these objectives, Android provides these key security features:

- Robust security at the OS level through the Linux kernel
- Mandatory application sandbox for all applications
- Secure interprocess communication
- Application signing
- Application-defined and user-granted permissions

The sections below describe these and other security features of the Android platform. Figure 2.2 summarizes the security components and considerations of the various levels of the Android software stack. Each component assumes that the components below are properly secured. With the exception of a small amount of Android OS code running as root, all code above the Linux Kernel is restricted by the Application Sandbox.

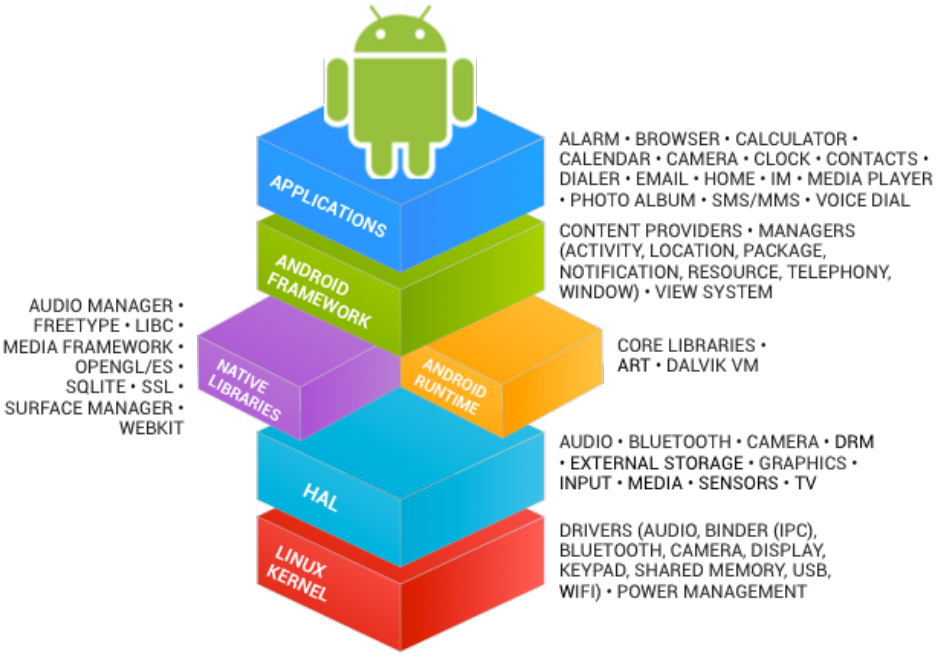

FIGURE 2.2: Android software stack. [Google]

2.6 System and Kernel Level Security

At the operating system level, the Android platform provides the security of the Linux kernel, as well as a secure inter-process communication (IPC) facility to enable secure communication between applications running in different processes. These security features at the OS level ensure that even native code is constrained by the Application Sandbox. Whether that code is the result of included application behavior or a exploitation of an application vulnerability, the system would prevent the rogue application from harming other applications, the Android system, or the device itself.

The foundation of the Android platform is the Linux kernel. The Linux kernel itself has been in widespread use for years, and is used in millions of security-sensitive environments. Through its history of constantly being researched, attacked, and

fixed by thousands of developers, Linux has become a stable and secure kernel trusted by many corporations and security professionals.

As the base for a mobile computing environment, the Linux kernel provides Android with several key security features, including:

- A user-based permissions model
- Process isolation
- Extensible mechanism for secure IPC
- The ability to remove unnecessary and potentially insecure parts of the kernel

As a multiuser operating system, a fundamental security objective of the Linux kernel is to isolate user resources from one another. The Linux security philosophy is to protect user resources from one another. Thus, Linux:

- Prevents user A from reading user B's files
- Ensures that user A does not exhaust user B's memory
- Ensures that user A does not exhaust user B's CPU resources
- Ensures that user A does not exhaust user B's devices (e.g. telephony, GPS, bluetooth)

2.7 What is Audit?

A computer security audit is a manual or systematic measurable technical assessment of a system or application. Manual assessments include interviewing staff, performing security vulnerability scans, reviewing application and operating system access controls, and analyzing physical access to the systems. Automated assessments include system generated audit reports or using software to monitor and report changes to files and settings on a system. Systems can include personal computers, servers, mainframes, network routers, switches. Applications can include Web Services, Microsoft Project Central, Oracle Database etc. An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of

the sequence of activities that have affected at any time a specific operation, procedure, or event. The process that creates an audit trail is typically required to always run in a privileged mode, so it can access and supervise all actions from all users; a normal user should not be allowed to stop/change it. Furthermore, for the same reason, trail file or database table with a trail should not be accessible to normal users. Another way of handling this issue is through the use of a role-based security model in the software. The software can operate with the closed-looped controls, or as a 'closed system,' as required by many companies when using audit trail functionality[Wikipedia, 2015].

Using traditional logging methods, applications and components submit free-form text messages to system logging facilities such as the Unix Syslog process, or the Microsoft Windows System, Security or Application event logs. Java applications often fall back to the standard Java logging facility, log4j. These text messages usually contain information only assumed to be security-relevant by the application developer, who is often not a computer- or network-security expert.

The fundamental problem with such free-form event records is that each application developer individually determines what information should be included in an audit event record, and the overall format in which that record should be presented to the audit log. This variance in formatting among thousands of instrumented applications makes the job of parsing audit event records by analysis tools (such as the Novell Sentinel product, for example) difficult and error prone. Such domain and application specific parsing code included in analysis tools is also difficult to maintain, as changes to event formats inevitably work their way into newer versions of the applications over time[Wikipedia, 2015].

2.8 Linux System Auditing

The Linux Audit system [Enck et al., 2009] provides a way to track security-relevant information on your system. Based on pre-configured rules, Audit generates log entries to record as much information about the events that are happening on your system as possible. This information is crucial for mission-critical environments to determine the violator of the security policy and the actions they performed. Audit does not provide additional security to your system; rather, it can be used to discover violations of security policies used on your system. These violations can further be prevented by additional security measures such as SELinux.

The following list summarizes some of the information that Audit is capable of recording in its log files:

- Date and time, type, and outcome of an event.
- Sensitivity labels of subjects and objects.
- Association of an event with the identity of the user who triggered the event.
- All modifications to Audit configuration and attempts to access Audit log files.
- All uses of authentication mechanisms, such as SSH, Kerberos, and others.
- Changes to any trusted database, such as /etc/passwd.
- Attempts to import or export information into or from the system.
- Include or exclude events based on user identity, subject and object labels, and other attributes.

2.8.1 Use Cases

Watching file access Audit can track whether a file or a directory has been accessed, modified, executed, or the file's attributes have been changed. This is useful, for example, to detect access to important files and have an Audit trail available in case one of these files is corrupted.

Monitoring system calls Audit can be configured to generate a log entry every time a particular system call is used. This can be used, for example, to track changes to the system time by monitoring the settimeofday, clock _adjtime, and other time-related system calls.

Recording commands run by a user Because Audit can track whether a file has been executed, a number of rules can be defined to record every execution of a particular command. For example, a rule can be defined for every executable in the /bin directory. The resulting log entries can then be searched by user ID to generate an audit trail of executed commands per user.

Recording security events The pam_faillock authentication module is capable of recording failed login attempts. Audit can be set up to record failed login attempts as well, and provides additional information about the user who attempted to log in.

Searching for events Audit provides the ausearch utility, which can be used to filter the log entries and provide a complete audit trail based on a number of conditions.

Running summary reports The aureport utility can be used to generate, among other things, daily reports of recorded events. A system administrator can then analyze these reports and investigate suspicious activity furthermore.

Monitoring network access The iptables and ebtables utilities can be configured to trigger Audit events, allowing system administrators to monitor network access.

2.8.2 Audit System Architecture

The Audit system consists of two main parts: the user-space applications and utilities, and the kernel-side system call processing. The kernel component receives system calls from user-space applications and filters them through one of the three filters: user, task, or exit. Once a system call passes through one of these filters, it is sent through the exclude filter, which, based on the Audit rule configuration, sends it to the Audit daemon for further processing.

The user-space Audit daemon collects the information from the kernel and creates log file entries in a log file. Other Audit user-space utilities interact with the Audit daemon, the kernel Audit component, or the Audit log files:

audisp The Audit dispatcher daemon interacts with the Audit daemon and sends events to other applications for further processing. The purpose of this daemon is to provide a plug-in mechanism so that real-time analytical programs can interact with Audit events.

auditctl The Audit control utility interacts with the kernel Audit component to control a number of settings and parameters of the event generation process.

auditd The audit daemon is responsible for writing the audit messages that were generated through the audit kernel interface and triggered by application and

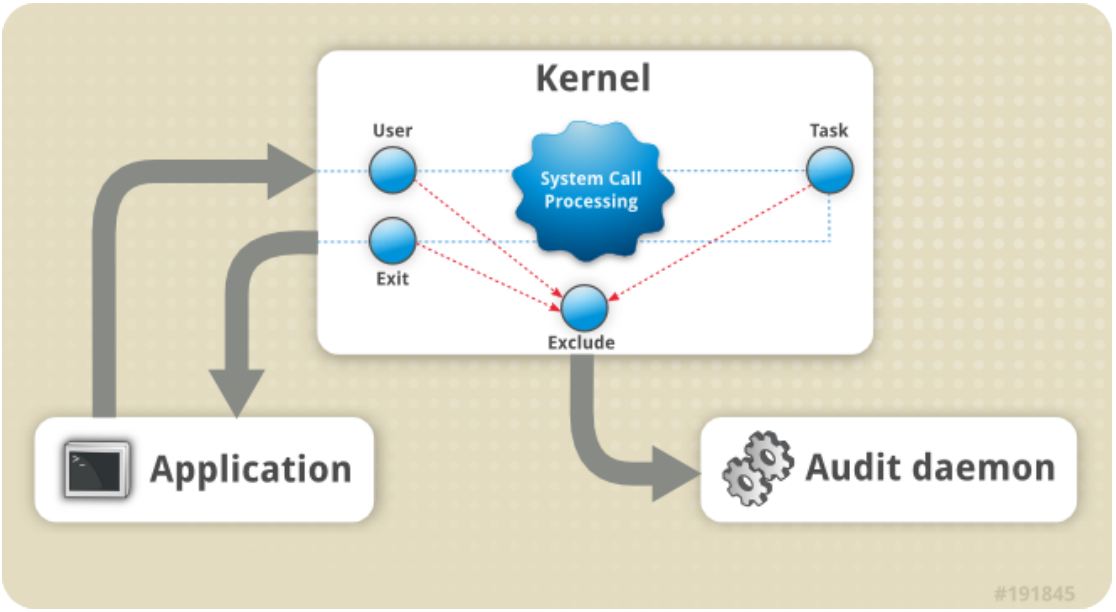

FIGURE 2.3: Audit System Architecture [RedHat]

system activity to disk. The way the audit daemon is started is controlled by its configuration file, */etc/sysconfig/auditd*. The audit system functions (once started) are controlled by */etc/audit/auditd.conf*.

audit rules The file */etc/audit/audit.rules* contains a sequence of auditctl commands that are loaded at system boot time immediately after the audit daemon is started.

aureport The aureport utility allows you to create custom reports from the audit event log. This report generation can easily be scripted, and the output can be used by various other applications, for example, to plot these results.

ausearch The ausearch utility can search the audit log file for certain events using various keys or other characteristics of the logged format.

autrace The autrace utility traces individual processes in a fashion similar to strace. The output of autrace is logged to the audit log.

2.8.3 Generation of Events

Kernel and user applications are the sources for the generation of events. Kernel is the most important source as it handles the system calls and the network layer actions. System calls and netlink operations are logged after it is completed. Every

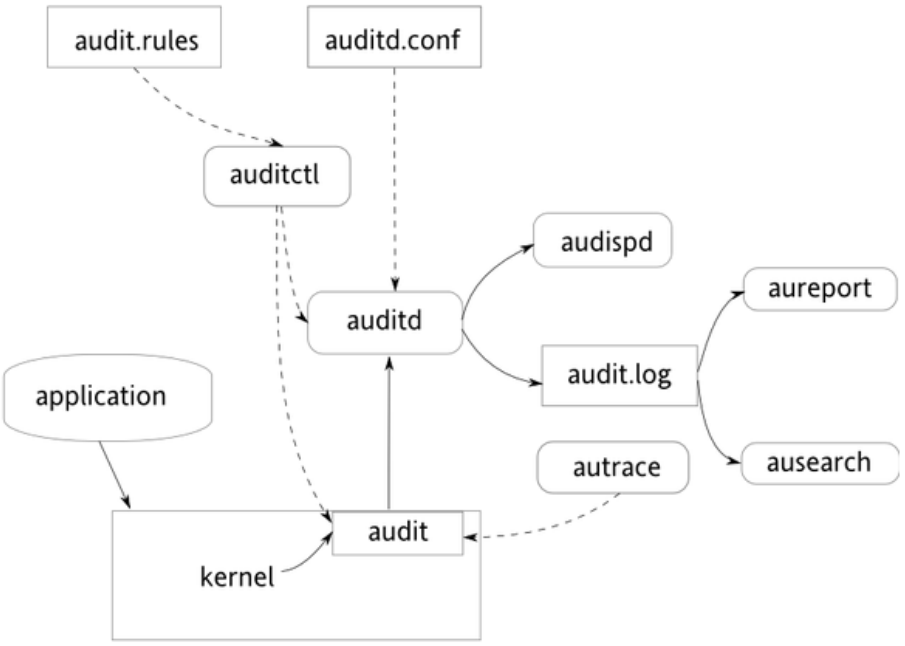

FIGURE 2.4: Components of Linux Audit [OpenSuse]

event generated by the Kernel contains information on the process on behalf of which the kernel generates the event, including the current uid, gid, the "Login ID" and "Audit ID" etc. Event messages are placed into a queue, from where they can be retrieved by the audit daemon through the read system call, one record at a time.

2.8.3.1 Kernel Sources

The kernel patch for audit creates several hooks for monitoring process creation/termination, and system calls entry/return, as well as one hook to track modifications of the system's networkconfiguration. System calls will be monitored on the entry and exit. Every system call is evaluated against the filter rules, and if passed, an event is generated. The Linux kernel network code can be controlled either by using the ioctl(2) system call of by using a netlink socket. The audit subsystem also generates events for process creation and termination including fork and clone.

2.8.3.2 User Sources

User space applications could generate their own audit events. These records are called "Audit User Messages". The applications that need it are the ones that

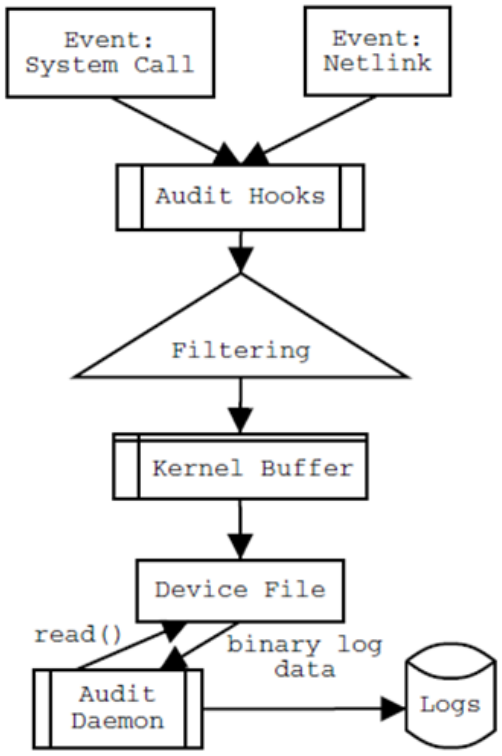

FIGURE 2.5: Data Flow: Kernel Sources [OpenSuse]

provides authentication schemes/change privileges and the ones that change the configuration of the system. The first type of applications does it by the help of Pluggable Authentication Modules. The latter group of applications needs to be modified manually to handle the LAuS interface to the kernel and to send the Audit User Messages.

2.8.3.3 Configuration

Only root has the ability to modify the audit rules. The only component of LAuS that accesses the config files is the audit daemon. The audit daemon needs a main configuration file for defining thresholds and corresponding actions etc, and two files for defining filter rules and filter object sets. These configuration files need to be modified directly by using a text editor and can be made effective by using the tool *aucfg*. *Aucfg* emits a reload message to force re-reading of the configuration. By applying DAC controls only the root user is able to execute *aucfg*, additionally the audit-subsystem only accepts messages generated by user root.

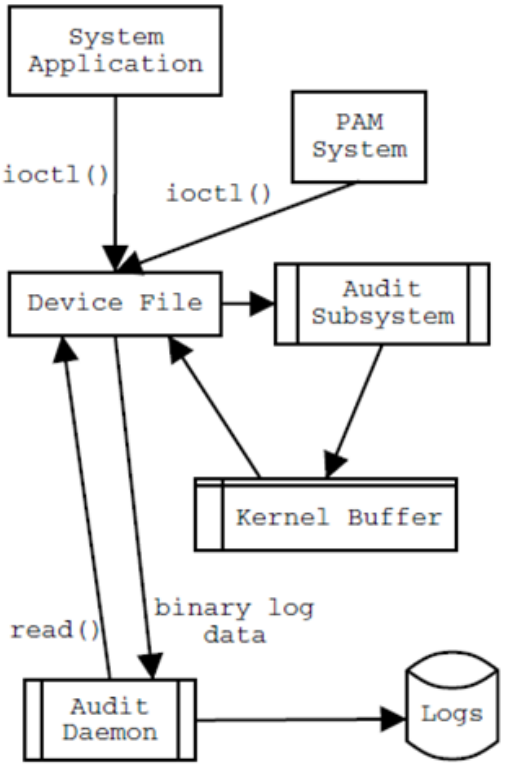

FIGURE 2.6: Data Flow: User Sources [OpenSuse]

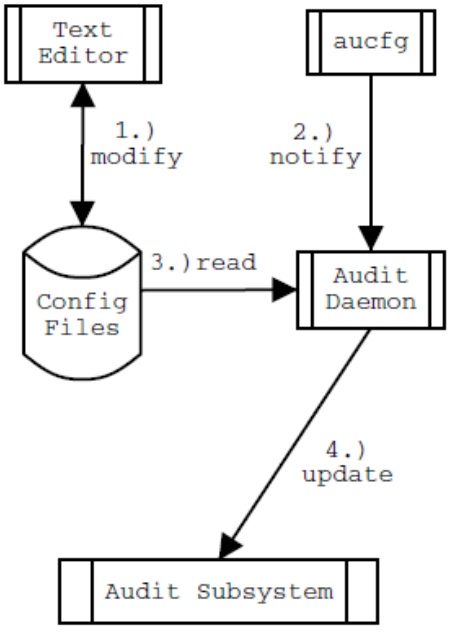

FIGURE 2.7: Data Flow: Configuration [OpenSuse]

Chapter 3

Proposed Android System Auditing

The primary goal of the thesis is to get auditing into Android. The main contribution of the work is to create a new auditing system for Android. The savvy user will thus be presented with more information on the underlying system.

The differences in *libc* and *bionic* [Devos, 2014] when coming from Linux System to Android presents substantial amount of problems in using Linux Audit as is in Android System. Husted et al., 2013 says that Linux Audit can be quite heavy when running on a low powered Android device as it would require a lot of IO operations to write all the logs into the disk. The information obtained from audit logs could be too coarse to extract meaningful information from them.

An alternative solution is to create a new Android specific audit system from ground up which can provide more Android specific event information.

3.1 Android Audit Subsystem

The newly proposed audit subsystem have a central component, Audit Manager Service, in the framework level, along with the existing System Services like Telephony Manager, Activity Manager etc. The service will be started by the System Server on boot, and will work in the background. Just like Linux Audit gets data by inserting "Hooks" into the kernel, our audit service also inserts hooks, but not in the kernel, but in the Android System Libraries and the framework. We modify the Android System Libraries like android.net, android.media etc and framework

components like Activity Manager Service to insert code (hooks) to ask them to call an audit function with necessary parameters so that the audit could log it. Most of the hooks would be from other system services like Package Manager and Activity Manager.

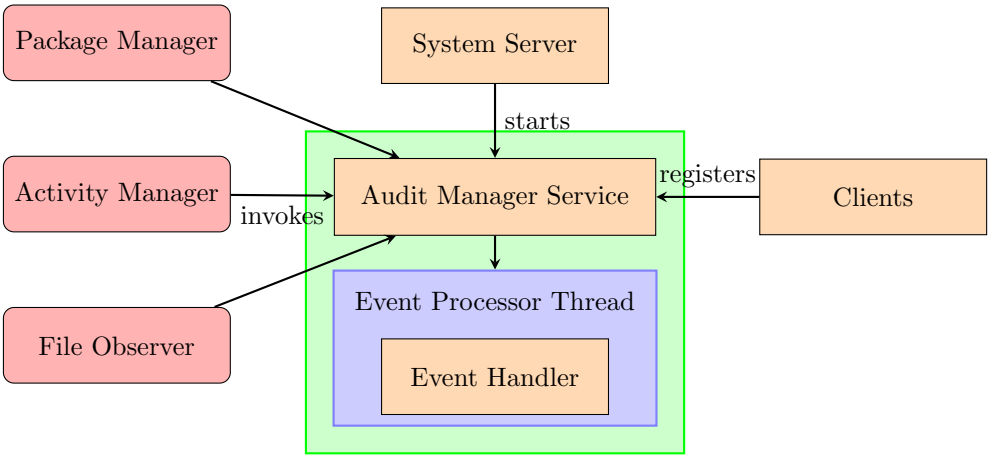

FIGURE 3.1: Audit Architecture

Figure 3.1 describes the proposed Audit architecture. There are two main components in the system - the Audit Manager Service and the Event Handler. The Audit Manager Service is a system service which is started by the System Server and is regarded as a core service. Audit Manager Service exposes a report() function through which hooks communicate to it. The hooks are inserted in other system services like Package Manager, Activity Manager etc, or new services like File Observer, and Android Libraries which are utilized by applications. Audit Manager passes the events to the Event Handler, which is run inside another thread. The Event Handler decides what to do with the Event by observing its contents. Further, other services could register with the Audit Manager Service to alert them if a particular event happens.

Figure 3.2 shows how an event gets logged. When an event happens, for example, User opens an app, and thus Activity's onCreate method is fired, the hook in the Activity class will create an event and send it to the Audit Manager Service. The Audit Manager Service with pass it on to the Event Handler. The Event handler is an implementation of the Android Handler, and thus maintains a message queue, Handler takes each Message, look at what the event is do appropriate action. Event handler will pass the event to the File Logger, which looks at the event and logs all the related data sent by the hook.

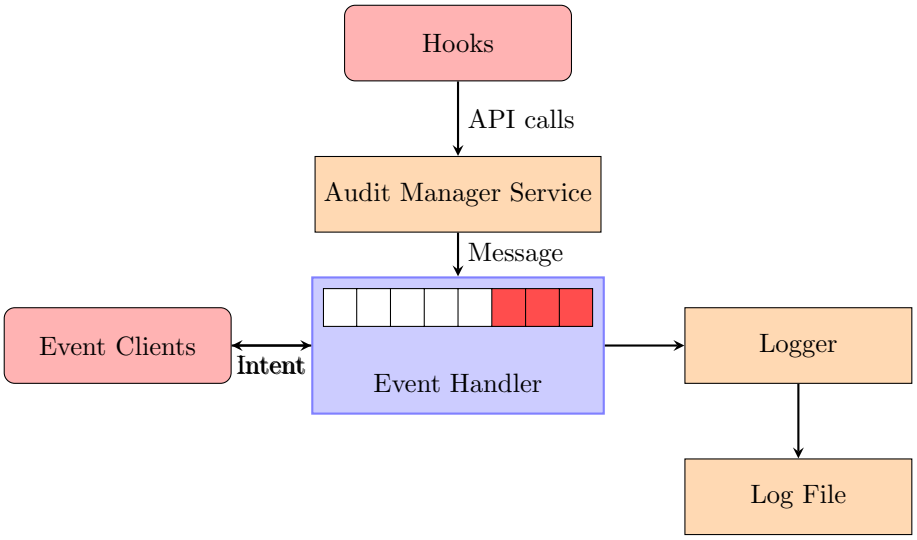

FIGURE 3.2: Audit Data Flow

3.2 Audit Manager Service

The System Service that controls audit. It is the implementation of IAuditManagerService interface defined using AIDL. It contains one hidden function, `void report(in Message reportData);`, with one parameter reportData, which is an Android Message. The Message.what holds an integer that corresponds to the type of event. All event types are declared in a separate class called AuditEvents. The parameter also has a data part associated with it, which is a Bundle, and has all the parameters relevant in the context.

Hooks inserted in the various parts of framework and library invokes the remote method report(Message reportData) and sends relevant event data to the AuditManagerService. A typical hook looks like this:

```
public void onPackageAdded(String packageName, int uid) {
        try {
            Message msg = Message.obtain();
            msg.what = AuditEvents.PACKAGE\_ADDED;
            Bundle data = new Bundle();
            data.putString("Package\_Name",packageName);
            data.putInt("uid", uid);
            msg.setData(data);
            ams.report(msg);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
    }
```

The above hook is from PackageMonitor class (frameworks/base/core/java/com/android/internal/content/PackageMonitor.java) , whose objective is to monitor package related changed. When a package is added it invokes the function onPackageAdded() with the package name and uid as parameters. This is an event that we would like to know about. So we insert a hook into that function as shown above. The hook is essentially the collection of relevant data at that point and then calling the remote method report() of AuditManagerService. AuditManagerService receives the event from the hook, but does not do much processing as to reduce lags. Instead, it passes the event to EventHandler, which runs on a separate thread.

3.3 Event Handler

Event Handler is an Android Handler, which runs on a separate thread. A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it – from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue. Thus Event Handler was made using a Handler on a separate thread so that audit system would not create performance issues by running in the UI thread. AuditManagerService does not do any processing on the events, it just acts as an interface and forwards the logs to the EventHandler. EventHandler queues the event Messages and processes it one by one. This is where an event will be redirected to the different modules. One of the modules that we have built as of now is FileLogger.

3.4 File Logger

The File Logger, as its name describes, logs the events into a log files. It is invoked by the EventHandler and thus is also run on a separate thread. It looks at the type of event and extract the parameters from the bundle. Then it creates a log string and uses a bufferedwriter to write it to a log file. It also takes care of creating the log file, and overwriting to a file after a retention period which is now set to 5 days.

3.5 Events

A typical audit event would contain these parameters:

- Timestamp
- Event Type
- Package Name

There will be more parameters according to the type of events. Some of the events expected are given below:

- Package Installed
- Package Updated
- Package Removed
- Package Modified
- Activity Lifecycle events (onCreate, onStop etc)
- Service Lifecycle events (onCreate, onUnbind etc)
- Broadcasts
- Intents (Start Activity, Bind to Service etc)

3.6 Package Installs and Removal

This can be tracked by inserting hooks into the PackageMonitor Class (frameworks/base/core/java/com/android/internal/content/PackageMonitor.java) The functions are:

- onPackageAdded(String packageName, int uid)
- onPackageRemoved(String packageName, int uid)
- onPackageRemovedAllUsers(String packageName, int uid)

- onPackageUpdateStarted(String packageName, int uid)
- onPackageUpdateFinished(String packageName, int uid)

It gives us the information about which package is modified and the uid associated with the package.

The sample logs obtained from the hooks in Package Monitor is given below.

```
2015-06-22 12:42:19.949|PACKAGE_ADDED|PackageName=com.example.package|uid=10053
2015-06-22 12:43:17.149|PACKAGE_REMOVED|PackageName=com.example.package|uid=10053
2015-06-22 12:43:17.814|PACKAGE_REMOVED_ALL_USERS|PackageName=com.example.package
    ↪ |uid=10053
```

3.7 SELinux Policy

SELinux is a kernel module that controls security policies. What this effectively does is migrate the security of the system to the kernel level. SELinux was developed by Smalley et al., 2001 who is part of Trusted Systems Research Group, US National Security Agency (NSA) and was released into the mainline Linux kernel in 2003. Later, Smalley and Craig, 2013 adapted it for Android. It does an outstanding job of limiting the privileges of applications to prevent security breaches on a system, and it has been a major factor in helping Linux to achieve a level of security no other platform has reached. Android has had a strong application sandbox since the very beginning; Security Enhanced Linux (SELinux) pushes enforcement of the Android security model further into the core of the OS and makes it easier to audit and monitor so there's less room for an attack. With Android 5.0, SELinux Enforcing mode is required for all applications on all devices.

So, when a new system service is added, like our AuditManagerService, the system will throw a SecurityException saying PERMISSION DENIED since selinux policy denies to add such a service. It would result in an error that looks like this in logcat:

```
E/SELinux (   51): avc:  denied  { add } for service=Audit scontext=u:r:
    ↪ system_server:s0 tcontext=u:object_r:default_android_service:s0 tclass=
    ↪ service_
manager
05-11 15:49:51.371    54    54 E ServiceManager: add_service('AuditManagerService
    ↪ ',28) uid=1000 - PERMISSION DENIED
05-11 15:49:51.378   248   248 E SystemServer: Failure starting
    ↪ AuditManagerService
05-11 15:49:51.378   248   248 E SystemServer: java.lang.SecurityException
```

```
05-11 15:49:51.378   248   248 E SystemServer:  at android.os.BinderProxy.
    ↪ transactNative(Native Method)
05-11 15:49:51.378   248   248 E SystemServer:  at android.os.BinderProxy.
    ↪ transact(Binder.java:496)
05-11 15:49:51.378   248   248 E SystemServer:  at android.os.ServiceManagerProxy
    ↪ .addService(ServiceManagerNative.java:150)
05-11 15:49:51.378   248   248 E SystemServer:  at android.os.ServiceManager.
    ↪ addService(ServiceManager.java:72)
05-11 15:49:51.378   248   248 E SystemServer:  at com.android.server.
    ↪ SystemServer.startOtherServices(SystemServer.java:551)
05-11 15:49:51.378   248   248 E SystemServer:  at com.android.server.
    ↪ SystemServer.run(SystemServer.java:257)
05-11 15:49:51.378   248   248 E SystemServer:  at com.android.server.
    ↪ SystemServer.main(SystemServer.java:171)
05-11 15:49:51.378   248   248 E SystemServer:  at java.lang.reflect.Method.
    ↪ invoke(Native Method)
05-11 15:49:51.378   248   248 E SystemServer:  at com.android.internal.os.
    ↪ ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:723)
05-11 15:49:51.378   248   248 E SystemServer:  at com.android.internal.os.
    ↪ ZygoteInit.main(ZygoteInit.java:613)
```

All the service related selinux contexts are written in /external/sepolicy/service_contexts file inside the AOSP source. To add our new system service, we could just add it to the list.

```
wallpaper                                u:object_r:system_server_service:s0
webviewupdate                            u:object_r:system_server_service:s0
wifip2p                                  u:object_r:system_server_service:s0
wifiscanner                              u:object_r:system_server_service:s0
wifi                                     u:object_r:system_server_service:s0
window                                   u:object_r:system_server_service:s0
Audit                                    u:object_r:system_server_service:s0
*                                        u:object_r:default_android_service:s0
```

3.8 Tracking Activity Lifecycle

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window. Activities in the system are managed as an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running activity – the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits. The Figure 3.3 shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.

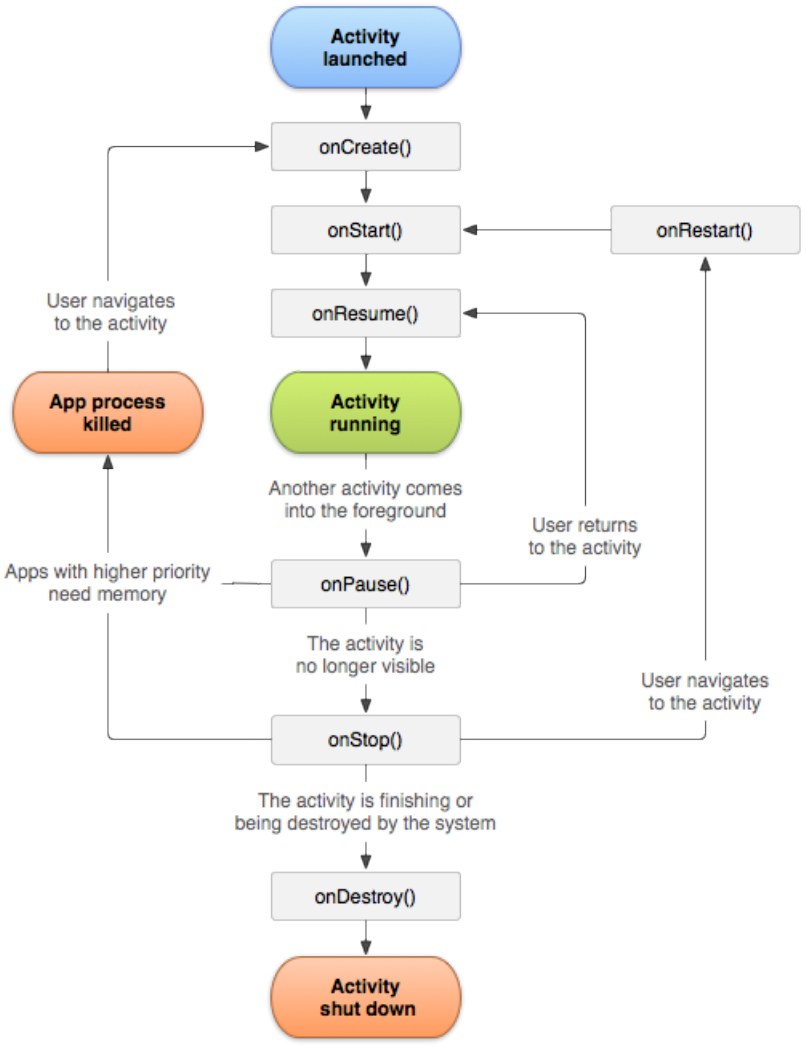

FIGURE 3.3: Activity Lifecycle [Google]

Using audit, we could track the lifecycle of every activity. Since every child class of Activity, if overriding the lifecycle methods should call its super class function in the function body. So, audit hooks are added into Activity.java (frameworks/base/core/java/android/app/Activity.java) inside the original lifecycle event functions like onCreate(), onPause(), onStop() etc. The logs looks like this:

```
2015-06-23 15:51:07.769|ACTIVITY_ONCREATE|Package=com.android.mms|Activity=com.
    ↪ android.mms.ui.ConversationList
2015-06-23 15:51:08.485|ACTIVITY_ONSTART|Package=com.android.mms|Activity=com.
    ↪ android.mms.ui.ConversationList
2015-06-23 15:51:08.534|ACTIVITY_ONRESUME|Package=com.android.mms|Activity=com.
    ↪ android.mms.ui.ConversationList
2015-06-23 15:51:09.96|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com.
    ↪ android.launcher2.Launcher
2015-06-23 15:51:25.548|ACTIVITY_ONSTART|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
```

```
2015-06-23 15:51:25.608|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-23 15:51:27.41|ACTIVITY_ONSTOP|Package=com.android.mms|Activity=com.
    ↪ android.mms.ui.ConversationList
2015-06-23 15:51:27.514|ACTIVITY_ONDESTROY|Package=com.android.mms|Activity=com.
    ↪ android.mms.ui.ConversationList
2015-06-23 15:51:30.356|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com
    ↪ .android.launcher2.Launcher
2015-06-23 15:51:31.705|ACTIVITY_ONCREATE|Package=com.android.browser|Activity=
    ↪ com.android.browser.BrowserActivity
```

All of the above logs points to what happens with the Activity. As a part of auditing, we would also like to know who started the activity. Thus, hooks should be inserted at the point where one activity calls another. An activity can be started by several methods in Context, like Context.startActivity(), Context.startActivityforResult() etc. Thus the hooks should be inserted in the Context Implementation (frameworks/base/core/java/app/android/ContextImpl.java), but it does not have all the information needed to completely understand the caller and callee. A system service called Activity Manager Service (frameworks/base/services/core/java/com/android/server/am/ActivityManagerService.java) actually takes care of all the permission related stuff. This is a good point to insert a hook and extract all the data needed. The hooks was inserted at functions such as startActivity(), startActivityasUser() etc and logs were obtained.

```
2015-06-25 09:53:31.139|START_ACTIVITY|Package=in.timelinestudios.ocha|Activity=
    ↪ in.timelinestudios.ocha.PlayerActivity|userId=0|CallerUID=10053|CallerPid
    ↪ =1184|CallingPackage=in.timelinestudios.ocha
2015-06-25 09:51:47.628|START_ACTIVITY|Package=com.android.gallery3d|Activity=com
    ↪ .android.gallery3d.app.GalleryActivity|userId=0|CallerUID=10007|CallerPid
    ↪ =628|CallingPackage=com.android.launcher
```

3.9 Tracking Service Lifecycle

A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use. Services can be started with Context.startService() and Context.bindService().There are two reasons that a service can be run by the system. If someone calls Context.startService() then the system will retrieve the service (creating it and calling its onCreate() method if needed) and then call its onStartCommand(Intent, int, int) method with the arguments supplied by the client. The service will at this point continue running until Context.stopService() or stopSelf() is called. Multiple calls to Context.startService()

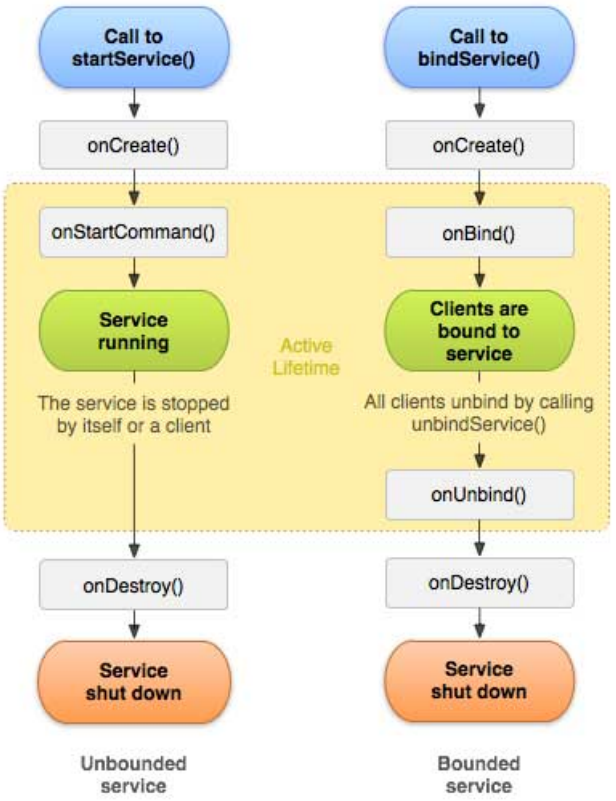

FIGURE 3.4: Service Lifecycle [Google]

do not nest (though they do result in multiple corresponding calls to onStartCommand()), so no matter how many times it is started a service will be stopped once Context.stopService() or stopSelf() is called; however, services can use their stopSelf(int) method to ensure the service is not stopped until started intents have been processed.

Similarly, bindService() calls will result into onBind() call instead of onStartCommand() after onCreate(), and then the clients are bound to the service. Also, when a service is stopped by itself or a client, for a service started via startService() and no clients binded with it, then it calls onDestroy() and then shuts down. Whereas, when a binded service invokes unbindService(), the service calls onUnbind() and after making sure that no other clients are attached to it, it calls onDestroy() and shuts down itself. It is important to audit these lifecycle calls to analyse the behaviour of the apps and the services. Thus, via audit, we insert various hooks into Service.java (frameworks/base/core/java/android/app/Service.java) to track the above mentioned lifecycle events as well as some extra internal calls like attach().

```
2015-06-24 13:37:29.042|SERVICE_ONCREATE|Service=com.android.providers.calendar.
    ↪ EmptyService
```

```
2015-06-24 13:37:48.909|SERVICE_ATTACH|Service=com.android.keychain.
    ↪ KeyChainService
2015-06-24 13:37:48.933|SERVICE_ONCREATE|Service=com.android.keychain.
    ↪ KeyChainService
2015-06-24 13:37:49.095|SERVICE_ONUNBIND|Service=com.android.keychain.
    ↪ KeyChainService|Package=com.android.keychain|IntentAction=android.security.
    ↪ IKeyChainService
2015-06-24 13:37:49.172|SERVICE_ATTACH|Service=com.android.dialer.calllog.
    ↪ CallLogNotificationsService
2015-06-24 13:37:49.213|SERVICE_ONCREATE|Service=com.android.dialer.calllog.
    ↪ CallLogNotificationsService
2015-06-24 13:37:49.675|SERVICE_ATTACH|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-24 13:37:49.689|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
```

All the hooks above logs the lifecycle events of a service. But whats more interesting is who starts the service, who stops it and who binds to it. To understand that, we insert hooks into ActivityManagerService.java (frameworks/base/services/core/-java/com/android/server/am/ActivityManagerService.java) to track Activities that call startService(), stopService() or bindService(). We get info on caller such as its uid and pid.

```
2015-06-25 11:30:06.981|START_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.CalendarProviderIntentService|userId
    ↪ =0|CallerUID=10001|CallerPid=890

2015-06-25 11:29:52.349|START_SERVICE|Package=com.android.calendar|SERVICE=com.
    ↪ android.calendar.alerts.AlertService|userId=0|CallerUID=10021|CallerPid
    ↪ =1028

2015-06-25 11:28:51.704|STOP_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.EmptyService|userId=0|CallerUID
    ↪ =10001|CallerPid=890

2015-06-25 11:29:20.654|STOP_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.AttachmentService|userId=0|CallerUID=10028|CallerPid
    ↪ =1071

2015-06-25 15:32:44.867|BIND_SERVICE|Package=in.timelinestudios.ocha|SERVICE=in.
    ↪ timelinestudios.ocha.services.AudioPlayerService|userId=0|CallerUID=10053|
    ↪ CallerPid=1126
```

3.10 Capturing Broadcasts

Broadcast intents are Intent objects that are broadcast via a call to the sendBroadcast(), sendStickyBroadcast() or sendOrderedBroadcast() methods of the Activity class (the latter being used when results are required from the broadcast). In addition to providing a messaging and event system between application components,

broadcast intents are also used by the Android system to notify interested applications about key system events (such as the external power supply or headphones being connected or disconnected).

When a broadcast intent is created, it must include an action string in addition to optional data and a category string. As with standard intents, data is added to a broadcast intent using key-value pairs in conjunction with the putExtra() method of the intent object. The optional category string may be assigned to a broadcast intent via a call to the addCategory() method. The action string, which identifies the broadcast event, must be unique and typically uses the applications Java package name syntax. For example, the following code fragment creates and sends a broadcast intent including a unique action string and data:

```
Intent intent = new Intent();
intent.setAction("com.example.Broadcast");
intent.putExtra("HighScore", 1000);
sendBroadcast(intent);
```

```
2015-06-26 13:39:10.801|BROADCAST|Package=com.android.deskclock|Class=com.android
    ↪ .alarmclock.AnalogAppWidgetProvider|ProcessRecord=341:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_ENABLED|CallerUID=1000|CallerPid=341
2015-06-26 13:39:10.818|BROADCAST|Package=com.android.deskclock|Class=com.android
    ↪ .alarmclock.AnalogAppWidgetProvider|ProcessRecord=341:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_UPDATE|CallerUID=1000|CallerPid=341
2015-06-26 13:41:26.134|BROADCAST|Package=com.android.mms|Class=com.android.mms.
    ↪ transaction.SmsReceiver|ProcessRecord=999:com.android.mms/u0a9|Action=com.
    ↪ android.mms.transaction.SEND_INACTIVE_MESSAGE|CallerUID=10009|CallerPid=999
2015-06-26 14:43:26.11|BROADCAST|Package=com.android.deskclock|Class=com.android.
    ↪ alarmclock.AnalogAppWidgetProvider|ProcessRecord=341:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_ENABLED|CallerUID=1000|CallerPid=341
2015-06-26 14:43:26.18|BROADCAST|Package=com.android.deskclock|Class=com.android.
    ↪ alarmclock.AnalogAppWidgetProvider|ProcessRecord=341:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_UPDATE|CallerUID=1000|CallerPid=341
2015-06-26 14:46:34.13|BROADCAST|Package=com.android.mms|Class=com.android.mms.
    ↪ transaction.SmsReceiver|ProcessRecord=998:com.android.mms/u0a9|Action=com.
    ↪ android.mms.transaction.SEND_INACTIVE_MESSAGE|CallerUID=10009|CallerPid=998
```

The Action indicates the kind of action it is trying to perform. So `android.appwidget.action.APPWIDGET_UPDATE` as its name suggests broadcast that the widget is updated and possibly some new data. We have listed all the possible Activity Intent Actions, System Broadcast Intent Actions and Intent Categories in the appendix.

Chapter 4

Literature Survey

4.1 Related Works

The paper "An android application sandbox system for suspicious software detection" [Bläsing et al., 2010] discusses about detecting malware application by both static and dynamic analysis. The work aims to track system calls at the kernel level and thus predict malicious usage patterns in applications. They try to detect malicious patterns both statically and dynamically. Both the processes are done by a sandbox created by them called the AASandox which takes apk files and input and does static and dynamic analysis on them. In the static way of analysis, an apk is taken as input and decompressed (an Android package is essentially a zip file. Then the launcher activity is identified from the AndroidManifest.xml file. Using Baksmali, classes.dex file is decompiled and a folder with easily parsable Java byte code is obtained. Finally, this disassembled code is used for scanning suspicious patterns. They typically search for the following for suspicious patterns.

- usage of the Java Native Interface, which can be used to dynamically load native libraries.
- usage of Sytem.getRuntime().exec(..), which can be used to spawn native children processes and surpass the normal application life-cycle.
- usage of reflection (2 patterns total), which may be used to circumvent API restrictions.
- usage of services and IPC provision, which may drain the battery or overload the devices CPU.

- usage of Android Permissions to determine which permissions will be granted at install time.

Dynamic analysis is more complex than static analysis. In dynamic analysis, applications are installed on an emulator and random inputs are given using a monkey testing tool. Meanwhile, AASandbox runs at kernel level and hijacks all system calls for logging. The resulting log file will be then summarized and reduced to a mathematical vector for better analysis. AASandbox was used to analyse over 150 good applications from playstore and a malware application and the malware app stood out as an outlier in the analysis as it called fork() system call repeatedly without any proper purpose.

4.2 Porting Linux Audit to Android

"Android Provenance: Diagnosing Device Disorders" [Husted et al., 2013] proposes operating system auditing and data provenance tracking as mechanisms for generating useful traces of system activity and information flow on mobile devices. The goal of these traces is to enable debugging and profiling of complicated system issues such as increased power drain. It contributes a prototype system for Android-based mobile devices and provides realistic examples of how the system can be used for debugging. System-level auditing provides a rich source of information. A provenance system can then organize the immense amount of information into a coherent, structured form. This allows reasoning and analysis that are not possible with raw audit streams. As Android is a Linux based platform, Linux Audit system is used. The use of a standard system removes the duplication of effort of creating a loadable kernel module.

Audit support was not available on the stock Android platform. They had to perform considerable work to enable system auditing. First, Androids kernel had to be modified to support system call auditing, which allows fine grain functional auditing at the operating system level. Adaptation of the user space components of audit to the Android platform was also non-trivial. One fundamental problem was that the audit user space utilities were written using the standard GNU libc C library. Android has only a nonstandard custom C library called Bionic. Audit was originally designed to run on heavily used servers. Smartphones, when compared to servers, have considerably greater resource constraints in terms of memory, I/O, and processing power. To improve performance, they had to make a fundamental

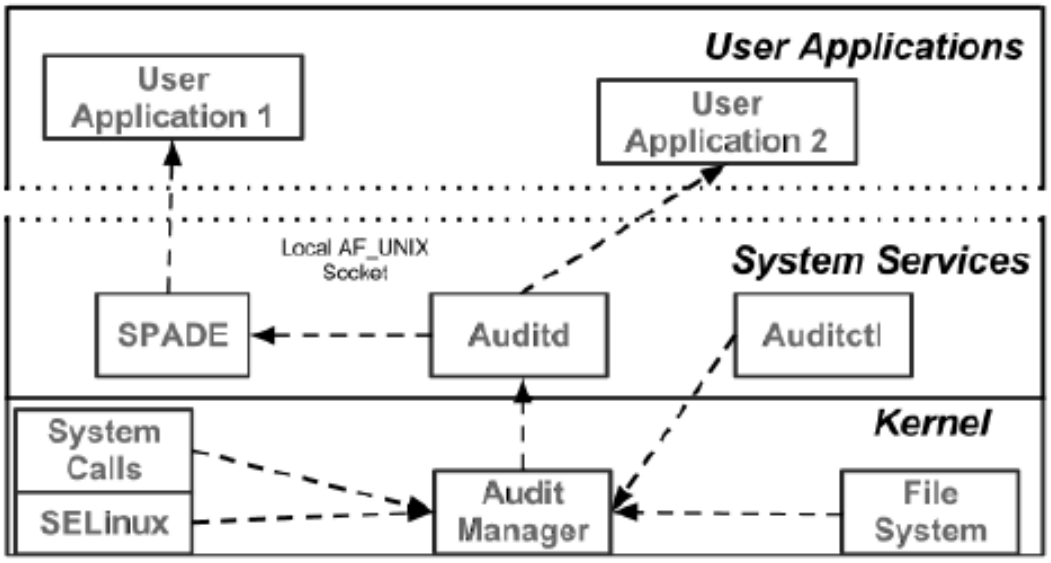

FIGURE 4.1: Architecture of Linux Audit Ported to Android [Husted et al.]

change to audit by removing the level of indirection from the dispatcher audispd. Previously, the audit daemon auditd would connect to the dispatcher through a local socket, which would then allow other applications to connect to it via a local socket. On servers, where processing speed and I/O are not an issue for audit, this socket configuration is not a problem. On smartphones, this leads to lost records when the local socket buffers fill up before they can be read. Removing the dispatcher eliminated one step in the chain of processing. Applications now connect directly with the audit daemon. This decreased the chance of lost or partial records. The resulting architecture is shown in Figure 4.1. The data provenance system SPADE was also ported to Android. The core of SPADE is a provenance kernel that mediates interactions between provenance producers and consumers.SPADE is written in Java and is thus easily cross compiled to Androids Dalvik virtual machine.This system was used to generate logs on a running Android system and used to make certain inferences. One of them was identifying wakelock usage in applications and how they cause battery drainage. Another observation involved finding alternatives for lag reduction like how calls to /dev/random can be optimized. The system was benchmarked using AnTuTu, and found that it causes only negligible performance degradation.

In "Runtime verification meets Android security" [Bauer et al., 2012] which modifies android systems to track system calls and events between kernel space and user space and infers android specific events like an application tries to send an SMS, or the system reports low battery status etc. It compares this data to prewritten

malware patterns like, for example, malware records phonecalls and saves it as an .amr file. So they observe for `write` system call to write to an .amr file and then alerts the user. This approach is good and similar to ours, but this would generate a lot of false positives.

In "Kernel-based behavior analysis for android malware detection" [Isohara et al., 2011], an audit framework called log cat is implemented on the Dalvik virtual machine to monitor the application behavior. However, only the limited events are dumped, because an application developers use the log cat for debugging. The behavior monitoring framework that can audit all activities of applications is important for security inspections on the market places. In this paper, they propose a kernel-base behavior analysis for android malware inspection. The system consists of a log collector in the Linux layer and a log analysis application. The log collector records all system calls and filters events with the target application. The log analyzer matches activities with signatures described by regular expressions to detect a malicious activity. Here, signatures of information leakage are automatically generated using the smart phone IDs, e.g., phone number, SIM serial number, and Gmail accounts. They implemented a prototype system and evaluate 230 applications in total. The result shows that their system can effectively detect malicious behaviors of the unknown applications.

In ""Andromaly": a behavioral malware detection framework for android devices"[Shabtai et al., 2012], a framework called Andromaly is presented for detecting malware on Android mobile devices. The proposed framework realizes a Host-based Malware Detection System that continuously monitors various features and events obtained from the mobile device and then applies Machine Learning anomaly detectors to classify the collected data as normal (benign) or abnormal (malicious). They developed four malicious applications, and evaluated Andromalys ability to detect new malware based on samples of known malware. They evaluated several combinations of anomaly detection algorithms, feature selection method and the number of top features in order to find the combination that yields the best performance in detecting new malware on Android. Empirical results suggest that the proposed framework is effective in detecting malware on mobile devices in general and on Android in particular.

"Flexible data-driven security for android"[Feth and Pretschner, 2012] proposes a more fine-grained security system beyond the standard permission system. With the system, it is possible to enforce complex policies that are built on temporal, cardinality, and spatial conditions ("notify if data is used after thirty days", "blur

data outside company's premises", etc.). Enforcement can be done by means of modification or inhibition of certain events and the execution of additional actions. Leveraging recent advances in information flow tracking technology, the policies can also pertain to data rather than single representations of that data. For instance, we can prohibit a movie from being played more than twice even if several copies have been created. It presents design and implementation of the system and provide a security and performance analysis.

Chapter 5

Evaluation

This chapter discusses the impact and performance of Audit Service on Android Security.

5.1 The Goal of Auditing

The contributions of the thesis, ie., a framework level auditing service for Android does not directly improve security of the system. What it provides is a very detailed log of everything that happens in the system, from the start of an app to background system broadcasts. These logs can be put into analysis and thus infer a lot about the state of the system at each part of the time. It could efficiently be used to track behavioural patterns of the system and thus can lead to the creation of an efficient malware analysis platform in the future. The best thing is that we could analyse the logs somewhere else, in a better platform suited for analysis, rather than doing it in device. Also, the logs could be used for forensics to analyse users usage patterns. Since everything is logged, we know details which include what applications were opened, at what time, and some information about action (opened camera, used bluetooth etc), and when they were stopped. This could further aid malware analysis to predict data and control flows and thus detect anomalous behaviour. Not everything needs to be postmortem. Another system service could request the audit service to be alerted for service requests. For example, if some other malicious activity tracking service is built into android, it could ask audit service to inform them when a set of specific activities are triggered.

5.2 Comparison with Previous Work

A lot of work has been carried out to find malcious applications in android system. Most of the work concentrated on statistically or dynamically analysis android applications, either in a sandbox or in an actual system itself. Very few works actually considered the factor that the whole system itself could go rogue. Even though some of the works done could actually analyse that too, the aim was limited to analysing Android applications. An example is "Android Provenance: Diagnosing Device Disorders"[Husted et al., 2013] which ported linux audit to Android, but used it for analysis application specific disorders like wake locks. One of the reasons stated was that linux audit would make a lot of IO calls if we try to log everything on to a file, which was fine for a server, but for low powered devices on which Android runs, this would create a huge lag. It would create performance and well as memory issues. Thus in that work, instead of logging to a file, all the logs were forwarded to a socket which was collected in a computer and then later analysed using SPADE. This made it impractical for long term real time audit of the system as it cannot be connected to a computer forever. Also, there was a whole lot of information, which we might not need for the analysis. This is why our work concentrated on creating an audit system, that is similar to linux audit, but logs data exclusive to the android system. Also, it was introduced in the framework level, unlike other works, which was in the kernel level, thus enabling us to log android specific system events a lot better. Our audit system was light weight and thus did not create much performance issues and ran well in the system with not much noticeable lag. Also, our system obtained and classified information on a very precise manner which was easier to analyse unlike linux audit which mostly depends on system calls to understand whats going on in the system. in our framework level audit, events are categorized on the basis on what they represent and from where they originated.

5.3 Impact on Battery Consumption

The main goal of the thesis was to get the audit system up and running. Battery consumption is not on the top of the list. So we did not take any explicit measurements to reduce battery consumption. But we made sure that audit system does not hog on the battery much. In android system, most of the battery is consumed by two things - screen and background processes with wakelocks. Since Audit always runs on the background, it makes no effect on screen. Audit Also

does not need a wakelock as it is built into the normal control - data flow of the android system. Whenever something happens, it gets logged. This is something that happens in the background without the need for an explicit wakelock. Whenever something happens in the system, the audit hooks catches them, and pushes them to the AuditServiceManager where it is logged. It does not matter whether it happens on the foreground or in the background. The case is the same even if a wakelock is acquired by the background process. Audit will get executed and it will get logged in the time interval in which the wakelock is possessed. Thus we can with a good level of certainty say that our system does not affect battery consumption significantly.

5.4 Contribution to Lag

Just like Battery consumption, the contribution to lag is not thing on top of the list when the audit service was built. But, since a significant lag will render the addition of audit unusable, we took some significant measures to make sure that it does not create performance issues. Most of the actions regarding audit like logging was done on a separate thread so that it would not affect the UI thread and the user would not experience any lag. Additionally, we used the data structures like Message and Bundle to pass around data, as they are optimized for the use in Android. The AuditServiceManager which serves as the entry point for the hooks is in the main thread, but to avoid lags, no further processing was done there and all the events generated was passed on to an Event Handler which ran on a separate thread. Event Handler (which is an Android Handler Class) maintains its own queue and processes the events one by one. Additionally, the logs are buffered and flushed at intervals for performance.

5.5 Local Storage

All the logs are written to a new directory in `/data` partition called `'audit'`. The file names are named `1.log,2.log...5.log` and are thus retained for 5 days and then starts over. The cloud upload system would have backed up the old logs by that time. The FileLogger class takes the events, checks the type of the event, extract the parameters from the data bundle and makes a log string which is buffered for writing into the file. Eventually the buffer gets flushed and the

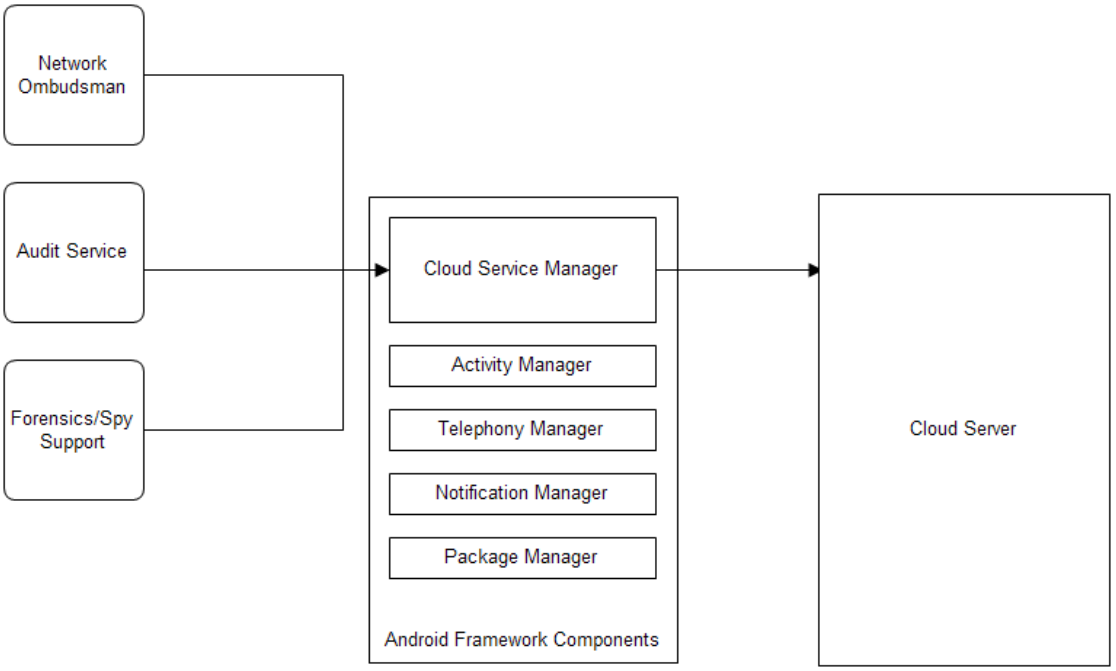

FIGURE 5.1: Cloud Uploader Architecture

contents are written into the log files. All this happens in a separate thread and does not cause any performance issues. Normal applications does not have access to read or write from /data/audit directory. So the logs cannot be tampered with, unless the device is rooted and the user give some malicious app a root permission to access the files.

5.6 Cloud Upload

Obviously, We cannot hold the logs on the local storage for ever. So we have a local data retention period of 5 days. During any one of the days, when the device is connected to a wifi network with internet access, the data is pushed on to a remote server. For this purpose, another system server is to be created just like Audit Service which does the all the uploading. It, by default does not use cellular data. But if there are logs left for the past 5 days, and thus the old logs are about to be written over in the next day, the cloud uploader service uses cellular data to push the data for the day it is about to write over.

Figure 5.1 shows the architecture. As a future work, we would also like the service to download relevant logs upon request for analysis in device.

5.7 Status of the Implementation

A good part of the implementation of the Audit Service is complete and is uploaded to github. The modifications/additions are listed in Table 5.1 All the new files

Status	Path	Lines of Code
NEW	frameworks/base/audit/java/com/android/server/audit/AuditManagerService.java	33
NEW	frameworks/base/audit/java/com/android/server/audit/AuditEvents.java	26
NEW	frameworks/base/audit/java/com/android/server/audit/EventHandler.java	20
NEW	frameworks/base/audit/java/com/android/server/audit/AuditEventProcessor.java	17
NEW	frameworks/base/audit/java/com/android/server/audit/FileLogger.java	160
NEW	frameworks/base/core/java/android/os/IAuditManagerService.aidl	8
MODIFIED	frameworks/base/core/java/android/app/Service.java	98
MODIFIED	frameworks/base/services/core/java/com/android/server/am/ActivityManagerService.java	136
MODIFIED	frameworks/base/core/java/android/app/Activity.java	85
MODIFIED	frameworks/base/core/java/com/android/internal/content/PackageMonitor.java	63
MODIFIED	frameworks/base/core/java/com/android/internal/content/PackageMonitor.java	63
MODIFIED	frameworks/base/services/java/com/android/server/SystemServer.java	7
MODIFIED	/external/sepolicy/service_contexts	1

Table 5.1: Added/Modified Files

added are the files related to Audit Service itself, and the modified files indicate the insertion of hooks in them. The SystemServer.java is modified to start Audit Service at boot as a core service for the system. The service_contexts file is the SELinux policy that allows the new service to make changes in the system.

The code is public and is uploaded to GitHub at https://github.com/Heedster/android-audit

Chapter 6

Conclusion

Our audit system pumps out logs that provides enough info to track most of the mainstream actions and activities in an android system. The system tracks life-cycle events of Activities and Services, starting of Activities and Services, binding to Services, broadcasts etc. This would tell us about which applications started, what all services they use, what do they broadcast etc. Also, there are events when a package is added, modified, updated, or removed from the system. There is a lot of things a malicious analysis tool could infer from the data. For example, there is a package updated event, but there was no activity from play store application or the application installer which is suspicious as a package cannot be installed or updated without either of these two. Thus, by grouping events, one could analyse a lot of things about the system. Also, audit system could watch for specific events and alert clients. For example, a client app could set a watch on a file for maybe, forensics purposes, and can get an alert whenever the file is modified. Similarly, a client could get an alert on any of the activities that the audit service tracks for.

Unlike, other work done on this domain which concentrated on kernel level auditing, our system concentrates on framework level auditing so that we could gain more insights on what happens in the system with information that can be easily understood by someone with some understanding of the android system. This also enables easy analysis of data and thus analysts could infer data more efficiently and easily. Also, we have carefully designed the system to be small, so that anyone could easily extend the project to add more hooks and use it for whatever purposes they want. The whole system is open source and everyone is encouraged to modify it.

As a future work, one could add more hooks easily to get more data logged. As of now, we have made sure that no sensitive data from the user gets reflected on the logs. Maybe, in the future, the system could be modified to serve as spy support for android devices. Also, the system needs to find some way to make the logs readable from the application layer, so that one could develop apps that could display or even analyse the logs. Anti-virus, anti-malware applications could benefit a lot from this. Thus our system could be developed as a good logging solution that is common to a lot of applications and analysis.

Appendix

Activity Intent Actions

These are the different Activity Intent Actions as per the latest API level 22 (Android 5.1 Lollipop):

```
android.app.action.ADD_DEVICE_ADMIN
android.app.action.PROVISION_MANAGED_PROFILE
android.app.action.SET_NEW_PASSWORD
android.app.action.START_ENCRYPTION
android.bluetooth.adapter.action.REQUEST_DISCOVERABLE
android.bluetooth.adapter.action.REQUEST_ENABLE
android.content.pm.action.SESSION_DETAILS
android.intent.action.ALL_APPS
android.intent.action.ANSWER
android.intent.action.APP_ERROR
android.intent.action.ASSIST
android.intent.action.ATTACH_DATA
android.intent.action.BUG_REPORT
android.intent.action.CALL
android.intent.action.CALL_BUTTON
android.intent.action.CHOOSER
android.intent.action.CREATE_DOCUMENT
android.intent.action.CREATE_LIVE_FOLDER
android.intent.action.CREATE_SHORTCUT
android.intent.action.DELETE
android.intent.action.DIAL
android.intent.action.EDIT
android.intent.action.EVENT_REMINDER
android.intent.action.GET_CONTENT
android.intent.action.INSERT
android.intent.action.INSERT_OR_EDIT
android.intent.action.INSTALL_PACKAGE
android.intent.action.MAIN
android.intent.action.MANAGE_NETWORK_USAGE
android.intent.action.MEDIA_SEARCH
android.intent.action.MUSIC_PLAYER
android.intent.action.OPEN_DOCUMENT
android.intent.action.OPEN_DOCUMENT_TREE
android.intent.action.PASTE
android.intent.action.PICK
android.intent.action.PICK_ACTIVITY
android.intent.action.POWER_USAGE_SUMMARY
```

```
android.intent.action.RESPOND_VIA_MESSAGE
android.intent.action.RINGTONE_PICKER
android.intent.action.RUN
android.intent.action.SEARCH
android.intent.action.SEARCH_LONG_PRESS
android.intent.action.SEND
android.intent.action.SENDTO
android.intent.action.SEND_MULTIPLE
android.intent.action.SET_ALARM
android.intent.action.SET_TIMER
android.intent.action.SET_WALLPAPER
android.intent.action.SHOW_ALARMS
android.intent.action.SYNC
android.intent.action.SYSTEM_TUTORIAL
android.intent.action.UNINSTALL_PACKAGE
android.intent.action.VIEW
android.intent.action.VIEW_DOWNLOADS
android.intent.action.VOICE_COMMAND
android.intent.action.WEB_SEARCH
android.media.action.DISPLAY_AUDIO_EFFECT_CONTROL_PANEL
android.media.action.IMAGE_CAPTURE
android.media.action.IMAGE_CAPTURE_SECURE
android.media.action.MEDIA_PLAY_FROM_SEARCH
android.media.action.STILL_IMAGE_CAMERA
android.media.action.STILL_IMAGE_CAMERA_SECURE
android.media.action.TEXT_OPEN_FROM_SEARCH
android.media.action.VIDEO_CAMERA
android.media.action.VIDEO_CAPTURE
android.media.action.VIDEO_PLAY_FROM_SEARCH
android.net.scoring.CHANGE_ACTIVE
android.net.scoring.CUSTOM_ENABLE
android.net.wifi.PICK_WIFI_NETWORK
android.net.wifi.action.REQUEST_SCAN_ALWAYS_AVAILABLE
android.nfc.action.NDEF_DISCOVERED
android.nfc.action.TAG_DISCOVERED
android.nfc.action.TECH_DISCOVERED
android.nfc.cardemulation.action.ACTION_CHANGE_DEFAULT
android.provider.Telephony.ACTION_CHANGE_DEFAULT
android.provider.calendar.action.HANDLE_CUSTOM_EVENT
android.search.action.SEARCH_SETTINGS
android.settings.ACCESSIBILITY_SETTINGS
android.settings.ACTION_NOTIFICATION_LISTENER_SETTINGS
android.settings.ACTION_PRINT_SETTINGS
android.settings.ADD_ACCOUNT_SETTINGS
android.settings.AIRPLANE_MODE_SETTINGS
android.settings.APN_SETTINGS
android.settings.APPLICATION_DETAILS_SETTINGS
android.settings.APPLICATION_DEVELOPMENT_SETTINGS
android.settings.APPLICATION_SETTINGS
android.settings.BATTERY_SAVER_SETTINGS
android.settings.BLUETOOTH_SETTINGS
android.settings.CAPTIONING_SETTINGS
android.settings.CAST_SETTINGS
android.settings.DATA_ROAMING_SETTINGS
android.settings.DATE_SETTINGS
android.settings.DEVICE_INFO_SETTINGS
android.settings.DISPLAY_SETTINGS
android.settings.DREAM_SETTINGS
```

```
android.settings.HOME_SETTINGS
android.settings.INPUT_METHOD_SETTINGS
android.settings.INPUT_METHOD_SUBTYPE_SETTINGS
android.settings.INTERNAL_STORAGE_SETTINGS
android.settings.LOCALE_SETTINGS
android.settings.LOCATION_SOURCE_SETTINGS
android.settings.MANAGE_ALL_APPLICATIONS_SETTINGS
android.settings.MANAGE_APPLICATIONS_SETTINGS
android.settings.MEMORY_CARD_SETTINGS
android.settings.NETWORK_OPERATOR_SETTINGS
android.settings.NFCSHARING_SETTINGS
android.settings.NFC_PAYMENT_SETTINGS
android.settings.NFC_SETTINGS
android.settings.PRIVACY_SETTINGS
android.settings.QUICK_LAUNCH_SETTINGS
android.settings.SECURITY_SETTINGS
android.settings.SETTINGS
android.settings.SHOW_REGULATORY_INFO
android.settings.SOUND_SETTINGS
android.settings.SYNC_SETTINGS
android.settings.USAGE_ACCESS_SETTINGS
android.settings.USER_DICTIONARY_SETTINGS
android.settings.VOICE_INPUT_SETTINGS
android.settings.WIFI_IP_SETTINGS
android.settings.WIFI_SETTINGS
android.settings.WIRELESS_SETTINGS
android.speech.tts.engine.CHECK_TTS_DATA
android.speech.tts.engine.GET_SAMPLE_TEXT
android.speech.tts.engine.INSTALL_TTS_DATA
```

System Broadcast Intent Actions

These are the different System Level Broadcast intents as per the latest API level 22 (Android 5.1 Lollipop):

```
android.app.action.ACTION_PASSWORD_CHANGED
android.app.action.ACTION_PASSWORD_EXPIRING
android.app.action.ACTION_PASSWORD_FAILED
android.app.action.ACTION_PASSWORD_SUCCEEDED
android.app.action.DEVICE_ADMIN_DISABLED
android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED
android.app.action.DEVICE_ADMIN_ENABLED
android.app.action.LOCK_TASK_ENTERING
android.app.action.LOCK_TASK_EXITING
android.app.action.NEXT_ALARM_CLOCK_CHANGED
android.app.action.PROFILE_PROVISIONING_COMPLETE
android.bluetooth.a2dp.profile.action.CONNECTION_STATE_CHANGED
android.bluetooth.a2dp.profile.action.PLAYING_STATE_CHANGED
android.bluetooth.adapter.action.CONNECTION_STATE_CHANGED
android.bluetooth.adapter.action.DISCOVERY_FINISHED
android.bluetooth.adapter.action.DISCOVERY_STARTED
android.bluetooth.adapter.action.LOCAL_NAME_CHANGED
android.bluetooth.adapter.action.SCAN_MODE_CHANGED
```

```
android.bluetooth.adapter.action.STATE_CHANGED
android.bluetooth.device.action.ACL_CONNECTED
android.bluetooth.device.action.ACL_DISCONNECTED
android.bluetooth.device.action.ACL_DISCONNECT_REQUESTED
android.bluetooth.device.action.BOND_STATE_CHANGED
android.bluetooth.device.action.CLASS_CHANGED
android.bluetooth.device.action.FOUND
android.bluetooth.device.action.NAME_CHANGED
android.bluetooth.device.action.PAIRING_REQUEST
android.bluetooth.device.action.UUID
android.bluetooth.devicepicker.action.DEVICE_SELECTED
android.bluetooth.devicepicker.action.LAUNCH
android.bluetooth.headset.action.VENDOR_SPECIFIC_HEADSET_EVENT
android.bluetooth.headset.profile.action.AUDIO_STATE_CHANGED
android.bluetooth.headset.profile.action.CONNECTION_STATE_CHANGED
android.bluetooth.input.profile.action.CONNECTION_STATE_CHANGED
android.bluetooth.pan.profile.action.CONNECTION_STATE_CHANGED
android.hardware.action.NEW_PICTURE
android.hardware.action.NEW_VIDEO
android.hardware.hdmi.action.OSD_MESSAGE
android.hardware.input.action.QUERY_KEYBOARD_LAYOUTS
android.intent.action.ACTION_POWER_CONNECTED
android.intent.action.ACTION_POWER_DISCONNECTED
android.intent.action.ACTION_SHUTDOWN
android.intent.action.AIRPLANE_MODE
android.intent.action.APPLICATION_RESTRICTIONS_CHANGED
android.intent.action.BATTERY_CHANGED
android.intent.action.BATTERY_LOW
android.intent.action.BATTERY_OKAY
android.intent.action.BOOT_COMPLETED
android.intent.action.CAMERA_BUTTON
android.intent.action.CONFIGURATION_CHANGED
android.intent.action.CONTENT_CHANGED
android.intent.action.DATA_SMS_RECEIVED
android.intent.action.DATE_CHANGED
android.intent.action.DEVICE_STORAGE_LOW
android.intent.action.DEVICE_STORAGE_OK
android.intent.action.DOCK_EVENT
android.intent.action.DOWNLOAD_COMPLETE
android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED
android.intent.action.DREAMING_STARTED
android.intent.action.DREAMING_STOPPED
android.intent.action.EXTERNAL_APPLICATIONS_AVAILABLE
android.intent.action.EXTERNAL_APPLICATIONS_UNAVAILABLE
android.intent.action.FETCH_VOICEMAIL
android.intent.action.GTALK_CONNECTED
android.intent.action.GTALK_DISCONNECTED
android.intent.action.HEADSET_PLUG
android.intent.action.HEADSET_PLUG
android.intent.action.INPUT_METHOD_CHANGED
android.intent.action.LOCALE_CHANGED
android.intent.action.MANAGE_PACKAGE_STORAGE
android.intent.action.MEDIA_BAD_REMOVAL
android.intent.action.MEDIA_BUTTON
android.intent.action.MEDIA_CHECKING
android.intent.action.MEDIA_EJECT
android.intent.action.MEDIA_MOUNTED
android.intent.action.MEDIA_NOFS
```

```
android.intent.action.MEDIA_REMOVED
android.intent.action.MEDIA_SCANNER_FINISHED
android.intent.action.MEDIA_SCANNER_SCAN_FILE
android.intent.action.MEDIA_SCANNER_STARTED
android.intent.action.MEDIA_SHARED
android.intent.action.MEDIA_UNMOUNTABLE
android.intent.action.MEDIA_UNMOUNTED
android.intent.action.MY_PACKAGE_REPLACED
android.intent.action.NEW_OUTGOING_CALL
android.intent.action.NEW_VOICEMAIL
android.intent.action.PACKAGE_ADDED
android.intent.action.PACKAGE_CHANGED
android.intent.action.PACKAGE_DATA_CLEARED
android.intent.action.PACKAGE_FIRST_LAUNCH
android.intent.action.PACKAGE_FULLY_REMOVED
android.intent.action.PACKAGE_INSTALL
android.intent.action.PACKAGE_NEEDS_VERIFICATION
android.intent.action.PACKAGE_REMOVED
android.intent.action.PACKAGE_REPLACED
android.intent.action.PACKAGE_RESTARTED
android.intent.action.PACKAGE_VERIFIED
android.intent.action.PHONE_STATE
android.intent.action.PROVIDER_CHANGED
android.intent.action.PROXY_CHANGE
android.intent.action.REBOOT
android.intent.action.SCREEN_OFF
android.intent.action.SCREEN_ON
android.intent.action.TIMEZONE_CHANGED
android.intent.action.TIME_SET
android.intent.action.TIME_TICK
android.intent.action.UID_REMOVED
android.intent.action.USER_PRESENT
android.intent.action.WALLPAPER_CHANGED
android.media.ACTION_SCO_AUDIO_STATE_UPDATED
android.media.AUDIO_BECOMING_NOISY
android.media.RINGER_MODE_CHANGED
android.media.SCO_AUDIO_STATE_CHANGED
android.media.VIBRATE_SETTING_CHANGED
android.media.action.CLOSE_AUDIO_EFFECT_CONTROL_SESSION
android.media.action.HDMI_AUDIO_PLUG
android.media.action.OPEN_AUDIO_EFFECT_CONTROL_SESSION
android.net.conn.BACKGROUND_DATA_SETTING_CHANGED
android.net.conn.CONNECTIVITY_CHANGE
android.net.nsd.STATE_CHANGED
android.net.scoring.SCORER_CHANGED
android.net.scoring.SCORE_NETWORKS
android.net.wifi.NETWORK_IDS_CHANGED
android.net.wifi.RSSI_CHANGED
android.net.wifi.SCAN_RESULTS
android.net.wifi.STATE_CHANGE
android.net.wifi.WIFI_STATE_CHANGED
android.net.wifi.p2p.CONNECTION_STATE_CHANGE
android.net.wifi.p2p.DISCOVERY_STATE_CHANGE
android.net.wifi.p2p.PEERS_CHANGED
android.net.wifi.p2p.STATE_CHANGED
android.net.wifi.p2p.THIS_DEVICE_CHANGED
android.net.wifi.supplicant.CONNECTION_CHANGE
android.net.wifi.supplicant.STATE_CHANGE
```

```
android.nfc.action.ADAPTER_STATE_CHANGED
android.os.action.POWER_SAVE_MODE_CHANGED
android.provider.Telephony.SIM_FULL
android.provider.Telephony.SMS_CB_RECEIVED
android.provider.Telephony.SMS_DELIVER
android.provider.Telephony.SMS_EMERGENCY_CB_RECEIVED
android.provider.Telephony.SMS_RECEIVED
android.provider.Telephony.SMS_REJECTED
android.provider.Telephony.SMS_SERVICE_CATEGORY_PROGRAM_DATA_RECEIVED
android.provider.Telephony.WAP_PUSH_DELIVER
android.provider.Telephony.WAP_PUSH_RECEIVED
android.speech.tts.TTS_QUEUE_PROCESSING_COMPLETED
android.speech.tts.engine.TTS_DATA_INSTALLED
```

Intent Categories

These are the different Intent Categories as per the latest API level 22 (Android 5.1 Lollipop):

```
android.intent.category.ALTERNATIVE
android.intent.category.APP_BROWSER
android.intent.category.APP_CALCULATOR
android.intent.category.APP_CALENDAR
android.intent.category.APP_CONTACTS
android.intent.category.APP_EMAIL
android.intent.category.APP_GALLERY
android.intent.category.APP_MAPS
android.intent.category.APP_MARKET
android.intent.category.APP_MESSAGING
android.intent.category.APP_MUSIC
android.intent.category.BROWSABLE
android.intent.category.CAR_DOCK
android.intent.category.CAR_MODE
android.intent.category.DEFAULT
android.intent.category.DESK_DOCK
android.intent.category.DEVELOPMENT_PREFERENCE
android.intent.category.EMBED
android.intent.category.HE_DESK_DOCK
android.intent.category.HOME
android.intent.category.INFO
android.intent.category.LAUNCHER
android.intent.category.LEANBACK_LAUNCHER
android.intent.category.LE_DESK_DOCK
android.intent.category.MONKEY
android.intent.category.NOTIFICATION_PREFERENCES
android.intent.category.OPENABLE
android.intent.category.PREFERENCE
android.intent.category.SELECTED_ALTERNATIVE
android.intent.category.TAB
```

Sample Audit Logs

These are sample logs generated by an ARM emulator running our audit service for a few minutes while the user does some normal tasks.

```
2015-06-27 08:55:18.999|BIND_SERVICE|Package=android|SERVICE=com.android.internal
    ↪ .backup.LocalTransportService|userId=0|CallerUID=1000|CallerPid=340
2015-06-27 08:55:20.265|SERVICE_ATTACH|Service=com.android.internal.backup.
    ↪ LocalTransportService
2015-06-27 08:55:37.432|BIND_SERVICE|Package=com.android.server.telecom|SERVICE=
    ↪ com.android.server.telecom.TelecomService|userId=0|CallerUID=1000|CallerPid
    ↪ =340
2015-06-27 08:55:37.57|SERVICE_ATTACH|Service=com.android.server.telecom.
    ↪ TelecomService
2015-06-27 08:55:37.598|SERVICE_ONCREATE|Service=com.android.server.telecom.
    ↪ TelecomService
2015-06-27 08:55:38.303|START_SERVICE|Package=com.android.systemui|SERVICE=com.
    ↪ android.systemui.SystemUIService|userId=0|CallerUID=1000|CallerPid=340
2015-06-27 08:55:43.909|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 08:55:43.975|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 08:55:48.773|BIND_SERVICE|Package=com.android.systemui|SERVICE=com.
    ↪ android.systemui.ImageWallpaper|userId=0|CallerUID=1000|CallerPid=340
2015-06-27 08:55:49.953|BIND_SERVICE|Package=com.android.inputmethod.latin|
    ↪ SERVICE=com.android.inputmethod.latin.LatinIME|userId=0|CallerUID=1000|
    ↪ CallerPid=340
2015-06-27 08:55:55.911|BIND_SERVICE|Package=android|SERVICE=android.hardware.
    ↪ location.GeofenceHardwareService|userId=0|CallerUID=1000|CallerPid=340
2015-06-27 08:55:56.279|SERVICE_ATTACH|Service=com.android.location.fused.
    ↪ FusedLocationService
2015-06-27 08:55:56.324|SERVICE_ONCREATE|Service=com.android.location.fused.
    ↪ FusedLocationService
2015-06-27 08:55:57.453|SERVICE_ATTACH|Service=android.hardware.location.
    ↪ GeofenceHardwareService
2015-06-27 08:55:57.896|SERVICE_ATTACH|Service=com.android.inputmethod.latin.
    ↪ LatinIME
2015-06-27 08:55:58.409|SERVICE_ONCREATE|Service=com.android.inputmethod.latin.
    ↪ LatinIME
2015-06-27 08:55:58.907|BIND_SERVICE|Package=com.android.printspooler|SERVICE=com
    ↪ .android.printspooler.model.PrintSpoolerService|userId=0|CallerUID=1000|
    ↪ CallerPid=340
2015-06-27 08:56:09.831|ACTIVITY_ONCREATE|Package=com.android.settings|Activity=
    ↪ com.android.settings.CryptKeeper
2015-06-27 08:56:10.589|SERVICE_ATTACH|Service=com.android.printspooler.model.
    ↪ PrintSpoolerService
2015-06-27 08:56:10.608|SERVICE_ONUNBIND|Service=com.android.systemui.
    ↪ SystemUIService|Package=com.android.systemui|IntentAction=null
2015-06-27 08:56:10.855|SERVICE_ONCREATE|Service=com.android.printspooler.model.
    ↪ PrintSpoolerService
2015-06-27 08:56:10.9|BIND_SERVICE|Package=com.android.systemui|SERVICE=com.
    ↪ android.systemui.keyguard.KeyguardService|userId=0|CallerUID=1000|CallerPid
    ↪ =340
2015-06-27 08:56:11.208|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ ImageWallpaper
```

```
2015-06-27 08:56:11.369|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ ImageWallpaper
2015-06-27 08:56:18.585|ACTIVITY_ONCREATE|Package=com.android.provision|Activity=
    ↪ com.android.provision.DefaultActivity
2015-06-27 08:56:27.169|BIND_SERVICE|Package=com.android.smspush|SERVICE=com.
    ↪ android.smspush.WapPushManager|userId=0|CallerUID=1001|CallerPid=766
2015-06-27 08:56:30.839|BIND_SERVICE|Package=com.android.smspush|SERVICE=com.
    ↪ android.smspush.WapPushManager|userId=0|CallerUID=1001|CallerPid=766
2015-06-27 08:56:32.404|START_SERVICE|Package=com.android.music|SERVICE=com.
    ↪ android.music.MediaPlaybackService|userId=0|CallerUID=10037|CallerPid=883
2015-06-27 08:56:43.73|SERVICE_ATTACH|Service=com.android.smspush.WapPushManager
2015-06-27 08:56:43.788|SERVICE_ONCREATE|Service=com.android.smspush.
    ↪ WapPushManager
2015-06-27 08:56:45.248|START_SERVICE|Package=com.android.phone|SERVICE=com.
    ↪ android.phone.TelephonyDebugService|userId=0|CallerUID=1001|CallerPid=766
2015-06-27 08:56:45.446|ACTIVITY_ONCREATE|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:26:47.157|SERVICE_ATTACH|Service=com.android.phone.
    ↪ TelephonyDebugService
2015-06-27 14:26:47.173|SERVICE_ONCREATE|Service=com.android.phone.
    ↪ TelephonyDebugService
2015-06-27 14:26:47.198|SERVICE_ONUNBIND|Service=com.android.phone.
    ↪ TelephonyDebugService|Package=com.android.phone|IntentAction=null
2015-06-27 14:26:48.001|ACTIVITY_ONDESTROY|Package=com.android.settings|Activity=
    ↪ com.android.settings.CryptKeeper
2015-06-27 14:26:48.015|ACTIVITY_ONDESTROY|Package=com.android.provision|Activity
    ↪ =com.android.provision.DefaultActivity
2015-06-27 14:26:51.167|ACTIVITY_ONSTART|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:26:51.182|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:26:53.118|BROADCAST|Package=com.android.deskclock|Class=com.android
    ↪ .alarmclock.AnalogAppWidgetProvider|ProcessRecord=340:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_ENABLED|CallerUID=1000|CallerPid=340
2015-06-27 14:26:53.121|BROADCAST|Package=com.android.deskclock|Class=com.android
    ↪ .alarmclock.AnalogAppWidgetProvider|ProcessRecord=340:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_UPDATE|CallerUID=1000|CallerPid=340
2015-06-27 14:26:55.034|SERVICE_ONUNBIND|Service=com.android.printspooler.model.
    ↪ PrintSpoolerService|Package=com.android.printspooler|IntentAction=null
2015-06-27 14:26:56.253|SERVICE_ATTACH|Service=com.android.music.
    ↪ MediaPlaybackService
2015-06-27 14:26:56.274|SERVICE_ONCREATE|Service=com.android.music.
    ↪ MediaPlaybackService
2015-06-27 14:26:56.977|BROADCAST|Package=com.android.deskclock|Class=com.android
    ↪ .alarmclock.AnalogAppWidgetProvider|ProcessRecord=340:system/1000|Action=
    ↪ android.appwidget.action.APPWIDGET_UPDATE_OPTIONS|CallerUID=1000|CallerPid
    ↪ =340
2015-06-27 14:26:58.653|SERVICE_ONCREATE|Service=com.android.printspooler.model.
    ↪ PrintSpoolerService
2015-06-27 14:26:59.053|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 14:26:59.21|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 14:27:00.31|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com.
    ↪ android.launcher2.Launcher
2015-06-27 14:27:03.968|ACTIVITY_ONDESTROY|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
```

```
2015-06-27 14:27:04.689|ACTIVITY_ONCREATE|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:27:08.184|ACTIVITY_ONSTART|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:27:08.264|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:27:08.761|SERVICE_ATTACH|Service=com.android.systemui.keyguard.
    ↪ KeyguardService
2015-06-27 14:27:08.798|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ ImageWallpaper
2015-06-27 14:27:08.811|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ ImageWallpaper
2015-06-27 14:27:26.34|STOP_SERVICE|Package=com.android.email|SERVICE=com.android
    ↪ .email.service.AttachmentService|userId=0|CallerUID=10028|CallerPid=864
2015-06-27 14:27:27.692|SERVICE_ATTACH|Service=com.android.exchange.service.
    ↪ EasService
2015-06-27 14:27:27.772|SERVICE_ONCREATE|Service=com.android.exchange.service.
    ↪ EasService
2015-06-27 14:27:28.186|STOP_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.AttachmentService|userId=0|CallerUID=10028|CallerPid
    ↪ =864
2015-06-27 14:27:29.16|START_SERVICE|Package=com.android.exchange|SERVICE=com.
    ↪ android.exchange.service.EasService|userId=0|CallerUID=10029|CallerPid=826
2015-06-27 14:27:30.098|STOP_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.AttachmentService|userId=0|CallerUID=10028|CallerPid
    ↪ =864
2015-06-27 14:27:35.829|STOP_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.AttachmentService|userId=0|CallerUID=10028|CallerPid
    ↪ =864
2015-06-27 14:27:36.392|START_SERVICE|Package=com.android.providers.downloads|
    ↪ SERVICE=com.android.providers.downloads.DownloadService|userId=0|CallerUID
    ↪ =10005|CallerPid=1073
2015-06-27 14:27:37.494|SERVICE_ATTACH|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:27:37.576|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:27:37.741|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:27:37.798|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:27:37.818|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:27:38.153|SERVICE_ONUNBIND|Service=com.android.providers.downloads.
    ↪ DownloadService|Package=com.android.providers.downloads|IntentAction=null
2015-06-27 14:27:38.923|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:27:39.056|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:27:39.112|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:27:39.122|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:27:39.345|BIND_SERVICE|Package=com.android.keychain|SERVICE=com.
    ↪ android.keychain.KeyChainService|userId=0|CallerUID=1000|CallerPid=340
2015-06-27 14:27:39.938|SERVICE_ATTACH|Service=com.android.keychain.
    ↪ KeyChainService
```

```
2015-06-27 14:27:39.959|SERVICE_ONCREATE|Service=com.android.keychain.
    ↪ KeyChainService
2015-06-27 14:27:40.085|SERVICE_ONUNBIND|Service=com.android.keychain.
    ↪ KeyChainService|Package=com.android.keychain|IntentAction=android.security.
    ↪ IKeyChainService
2015-06-27 14:28:38.454|SERVICE_ONCREATE|Service=com.android.music.
    ↪ MediaPlaybackService
2015-06-27 14:30:26.475|START_ACTIVITY|Package=com.android.browser|Activity=com.
    ↪ android.browser.BrowserActivity|userId=0|CallerUID=10007|CallerPid=898|
    ↪ CallingPackage=com.android.launcher
2015-06-27 14:30:28.118|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com
    ↪ .android.launcher2.Launcher
2015-06-27 14:30:29.168|ACTIVITY_ONCREATE|Package=com.android.browser|Activity=
    ↪ com.android.browser.BrowserActivity
2015-06-27 14:30:40.9|ACTIVITY_ONSTART|Package=com.android.browser|Activity=com.
    ↪ android.browser.BrowserActivity
2015-06-27 14:30:40.914|ACTIVITY_ONRESUME|Package=com.android.browser|Activity=
    ↪ com.android.browser.BrowserActivity
2015-06-27 14:30:47.338|BIND_SERVICE|Package=com.android.defcontainer|SERVICE=com
    ↪ .android.defcontainer.DefaultContainerService|userId=0|CallerUID=1000|
    ↪ CallerPid=340
2015-06-27 14:30:59.273|START_SERVICE|Package=com.android.dialer|SERVICE=com.
    ↪ android.dialer.calllog.CallLogNotificationsService|userId=0|CallerUID
    ↪ =10004|CallerPid=1244
2015-06-27 14:30:59.795|SERVICE_ATTACH|Service=com.android.defcontainer.
    ↪ DefaultContainerService
2015-06-27 14:30:59.865|SERVICE_ONCREATE|Service=com.android.defcontainer.
    ↪ DefaultContainerService
2015-06-27 14:31:01.647|SERVICE_ATTACH|Service=com.android.dialer.calllog.
    ↪ CallLogNotificationsService
2015-06-27 14:31:01.859|SERVICE_ONCREATE|Service=com.android.dialer.calllog.
    ↪ CallLogNotificationsService
2015-06-27 14:31:14.449|ACTIVITY_ONSTART|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:31:14.535|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:31:15.252|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:31:16.551|ACTIVITY_ONSTOP|Package=com.android.browser|Activity=com.
    ↪ android.browser.BrowserActivity
2015-06-27 14:31:24.348|START_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.EmptyService|userId=0|CallerUID
    ↪ =10001|CallerPid=1330
2015-06-27 14:31:25.458|STOP_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.EmptyService|userId=0|CallerUID
    ↪ =10001|CallerPid=1330
2015-06-27 14:31:27.886|SERVICE_ATTACH|Service=com.android.providers.calendar.
    ↪ EmptyService
2015-06-27 14:31:27.932|SERVICE_ONCREATE|Service=com.android.providers.calendar.
    ↪ EmptyService
2015-06-27 14:31:28.008|SERVICE_ONUNBIND|Service=com.android.providers.calendar.
    ↪ EmptyService|Package=com.android.providers.calendar|IntentAction=null
2015-06-27 14:31:28.043|SERVICE_ONCREATE|Service=com.android.providers.calendar.
    ↪ EmptyService
2015-06-27 14:31:28.074|START_SERVICE|Package=com.android.providers.downloads|
    ↪ SERVICE=com.android.providers.downloads.DownloadService|userId=0|CallerUID
    ↪ =10005|CallerPid=1073
```

```
2015-06-27 14:31:28.163|SERVICE_ATTACH|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:31:30.315|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:31:30.36|SERVICE_ONUNBIND|Service=com.android.providers.downloads.
    ↪ DownloadService|Package=com.android.providers.downloads|IntentAction=null
2015-06-27 14:31:30.821|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:31:31.565|START_SERVICE|Package=com.android.providers.media|SERVICE
    ↪ =com.android.providers.media.MediaScannerService|userId=0|CallerUID=10005|
    ↪ CallerPid=1073
2015-06-27 14:31:31.589|START_SERVICE|Package=com.android.providers.media|SERVICE
    ↪ =com.android.providers.media.MediaScannerService|userId=0|CallerUID=10005|
    ↪ CallerPid=1073
2015-06-27 14:31:31.639|SERVICE_ATTACH|Service=com.android.providers.media.
    ↪ MediaScannerService
2015-06-27 14:32:26.088|SERVICE_ONUNBIND|Service=com.android.defcontainer.
    ↪ DefaultContainerService|Package=com.android.defcontainer|IntentAction=null
2015-06-27 14:32:26.416|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:26.599|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:27.533|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:32:27.545|BROADCAST|Package=com.android.mms|Class=com.android.mms.
    ↪ transaction.SmsReceiver|ProcessRecord=1369:com.android.mms/u0a9|Action=com.
    ↪ android.mms.transaction.SEND_INACTIVE_MESSAGE|CallerUID=1000|CallerPid=340
2015-06-27 14:32:27.582|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:32:27.773|SERVICE_ATTACH|Service=com.android.mms.transaction.
    ↪ TransactionService
2015-06-27 14:32:28.862|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.SmsReceiverService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:32:28.923|SERVICE_ATTACH|Service=com.android.mms.transaction.
    ↪ SmsReceiverService
2015-06-27 14:32:30.322|START_SERVICE|Package=com.android.onetimeinitializer|
    ↪ SERVICE=com.android.onetimeinitializer.OneTimeInitializerService|userId=0|
    ↪ CallerUID=10011|CallerPid=1405
2015-06-27 14:32:30.398|SERVICE_ATTACH|Service=com.android.onetimeinitializer.
    ↪ OneTimeInitializerService
2015-06-27 14:32:30.413|SERVICE_ONCREATE|Service=com.android.onetimeinitializer.
    ↪ OneTimeInitializerService
2015-06-27 14:32:30.755|START_SERVICE|Package=com.android.calendar|SERVICE=com.
    ↪ android.calendar.alerts.AlertService|userId=0|CallerUID=10021|CallerPid
    ↪ =1308
2015-06-27 14:32:30.8|SERVICE_ATTACH|Service=com.android.calendar.alerts.
    ↪ AlertService
2015-06-27 14:32:30.923|START_SERVICE|Package=com.android.calendar|SERVICE=com.
    ↪ android.calendar.alerts.InitAlarmsService|userId=0|CallerUID=10021|
    ↪ CallerPid=1308
2015-06-27 14:32:30.976|SERVICE_ATTACH|Service=com.android.calendar.alerts.
    ↪ InitAlarmsService
2015-06-27 14:32:30.996|SERVICE_ONCREATE|Service=com.android.calendar.alerts.
    ↪ InitAlarmsService
2015-06-27 14:32:31.174|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:31.613|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
```

```
2015-06-27 14:32:31.628|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:31.646|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:31.835|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:31.85|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:31.867|PACKAGE_ADDED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:32:49.683|START_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.EmailBroadcastProcessorService|userId=0|CallerUID
    ↪ =10028|CallerPid=864
2015-06-27 14:32:49.747|SERVICE_ATTACH|Service=com.android.email.service.
    ↪ EmailBroadcastProcessorService
2015-06-27 14:32:49.773|SERVICE_ONCREATE|Service=com.android.email.service.
    ↪ EmailBroadcastProcessorService
2015-06-27 14:32:49.953|START_SERVICE|Package=com.android.email|SERVICE=com.
    ↪ android.email.service.EmailBroadcastProcessorService|userId=0|CallerUID
    ↪ =10028|CallerPid=864
2015-06-27 14:32:50.003|SERVICE_ATTACH|Service=com.android.email.service.
    ↪ EmailBroadcastProcessorService
2015-06-27 14:32:50.024|SERVICE_ONCREATE|Service=com.android.email.service.
    ↪ EmailBroadcastProcessorService
2015-06-27 14:32:50.236|SERVICE_ATTACH|Service=com.android.exchange.service.
    ↪ EasService
2015-06-27 14:32:50.308|SERVICE_ONCREATE|Service=com.android.exchange.service.
    ↪ EasService
2015-06-27 14:32:50.355|START_SERVICE|Package=com.android.exchange|SERVICE=com.
    ↪ android.exchange.service.EasService|userId=0|CallerUID=10029|CallerPid=826
2015-06-27 14:33:12.383|START_SERVICE|Package=com.android.providers.downloads|
    ↪ SERVICE=com.android.providers.downloads.DownloadService|userId=0|CallerUID
    ↪ =10005|CallerPid=1073
2015-06-27 14:33:12.46|START_SERVICE|Package=com.android.mms|SERVICE=com.android.
    ↪ mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:33:12.479|SERVICE_ATTACH|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:33:12.516|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:33:12.903|SERVICE_ATTACH|Service=com.android.mms.transaction.
    ↪ TransactionService
2015-06-27 14:33:12.968|SERVICE_ONUNBIND|Service=com.android.providers.downloads.
    ↪ DownloadService|Package=com.android.providers.downloads|IntentAction=null
2015-06-27 14:33:13.148|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:33:13.773|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:13.817|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:13.83|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:15.023|START_SERVICE|Package=com.android.calendar|SERVICE=com.
    ↪ android.calendar.alerts.AlertService|userId=0|CallerUID=10021|CallerPid
    ↪ =1308
2015-06-27 14:33:15.057|SERVICE_ATTACH|Service=com.android.calendar.alerts.
    ↪ AlertService
```

```
2015-06-27 14:33:20.381|START_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.CalendarProviderIntentService|userId
    ↪ =0|CallerUID=10001|CallerPid=1330
2015-06-27 14:33:20.423|SERVICE_ATTACH|Service=com.android.providers.calendar.
    ↪ CalendarProviderIntentService
2015-06-27 14:33:20.448|SERVICE_ONCREATE|Service=com.android.providers.calendar.
    ↪ CalendarProviderIntentService
2015-06-27 14:33:20.696|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:20.766|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:20.794|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:20.979|START_SERVICE|Package=com.android.musicfx|SERVICE=com.
    ↪ android.musicfx.Compatibility$Service|userId=0|CallerUID=10010|CallerPid
    ↪ =1504
2015-06-27 14:33:21.031|SERVICE_ATTACH|Service=com.android.musicfx.
    ↪ Compatibility$Service
2015-06-27 14:33:21.045|SERVICE_ONCREATE|Service=com.android.musicfx.
    ↪ Compatibility$Service
2015-06-27 14:33:23.742|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:23.785|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:23.805|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:25.704|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.SmsReceiverService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:33:25.744|SERVICE_ATTACH|Service=com.android.mms.transaction.
    ↪ SmsReceiverService
2015-06-27 14:33:26.124|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:26.166|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.232|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.413|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:26.466|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.481|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.614|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:26.641|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.658|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.805|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:26.834|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
```

```
2015-06-27 14:33:26.852|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:26.93|START_SERVICE|Package=com.android.providers.calendar|
    ↪ SERVICE=com.android.providers.calendar.CalendarProviderIntentService|userId
    ↪ =0|CallerUID=10001|CallerPid=1330
2015-06-27 14:33:26.96|SERVICE_ATTACH|Service=com.android.providers.calendar.
    ↪ CalendarProviderIntentService
2015-06-27 14:33:26.974|SERVICE_ONCREATE|Service=com.android.providers.calendar.
    ↪ CalendarProviderIntentService
2015-06-27 14:33:31.663|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:33:31.729|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:33:31.744|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:34:00.863|START_ACTIVITY|Package=in.timelinestudios.ocha|Activity=
    ↪ in.timelinestudios.ocha.SplashActivity|userId=0|CallerUID=10007|CallerPid
    ↪ =898|CallingPackage=com.android.launcher
2015-06-27 14:34:03.281|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com
    ↪ .android.launcher2.Launcher
2015-06-27 14:34:04.396|ACTIVITY_ONCREATE|Package=in.timelinestudios.ocha|
    ↪ Activity=in.timelinestudios.ocha.SplashActivity
2015-06-27 14:34:04.638|ACTIVITY_ONSTART|Package=in.timelinestudios.ocha|Activity
    ↪ =in.timelinestudios.ocha.SplashActivity
2015-06-27 14:34:04.657|ACTIVITY_ONRESUME|Package=in.timelinestudios.ocha|
    ↪ Activity=in.timelinestudios.ocha.SplashActivity
2015-06-27 14:34:05.707|START_ACTIVITY|Package=in.timelinestudios.ocha|Activity=
    ↪ in.timelinestudios.ocha.ExplorationActivity|userId=0|CallerUID=10053|
    ↪ CallerPid=1559|CallingPackage=in.timelinestudios.ocha
2015-06-27 14:34:06.107|ACTIVITY_ONCREATE|Package=in.timelinestudios.ocha|
    ↪ Activity=in.timelinestudios.ocha.ExplorationActivity
2015-06-27 14:34:09.22|START_SERVICE|Package=in.timelinestudios.ocha|SERVICE=com.
    ↪ parse.PushService|userId=0|CallerUID=10053|CallerPid=1559
2015-06-27 14:34:13.182|ACTIVITY_ONSTART|Package=in.timelinestudios.ocha|Activity
    ↪ =in.timelinestudios.ocha.ExplorationActivity
2015-06-27 14:34:14.666|ACTIVITY_ONRESUME|Package=in.timelinestudios.ocha|
    ↪ Activity=in.timelinestudios.ocha.ExplorationActivity
2015-06-27 14:34:14.718|SERVICE_ATTACH|Service=com.parse.PushService
2015-06-27 14:34:14.724|SERVICE_ONCREATE|Service=com.parse.PushService
2015-06-27 14:34:32.177|ACTIVITY_ONSTOP|Package=in.timelinestudios.ocha|Activity=
    ↪ in.timelinestudios.ocha.SplashActivity
2015-06-27 14:34:32.184|ACTIVITY_ONDESTROY|Package=in.timelinestudios.ocha|
    ↪ Activity=in.timelinestudios.ocha.SplashActivity
2015-06-27 14:34:39.007|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 14:34:39.03|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ SystemUIService
2015-06-27 14:34:41.919|SERVICE_ATTACH|Service=com.android.systemui.keyguard.
    ↪ KeyguardService
2015-06-27 14:34:41.966|SERVICE_ATTACH|Service=com.android.systemui.
    ↪ ImageWallpaper
2015-06-27 14:34:42.02|SERVICE_ONCREATE|Service=com.android.systemui.
    ↪ ImageWallpaper
2015-06-27 14:41:14.585|START_SERVICE|Package=in.timelinestudios.ocha|SERVICE=com
    ↪ .parse.PushService|userId=0|CallerUID=10053|CallerPid=1559
2015-06-27 14:41:29.125|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
```

```
2015-06-27 14:41:30.198|SERVICE_ATTACH|Service=com.android.mms.transaction.
    ↪ TransactionService
2015-06-27 14:41:33.382|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:41:33.462|START_SERVICE|Package=com.android.providers.downloads|
    ↪ SERVICE=com.android.providers.downloads.DownloadService|userId=0|CallerUID
    ↪ =10005|CallerPid=1073
2015-06-27 14:41:33.539|SERVICE_ATTACH|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:41:33.638|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:41:33.685|START_SERVICE|Package=com.android.mms|SERVICE=com.android
    ↪ .mms.transaction.TransactionService|userId=0|CallerUID=10009|CallerPid=1369
2015-06-27 14:41:33.842|SERVICE_ONUNBIND|Service=com.android.providers.downloads.
    ↪ DownloadService|Package=com.android.providers.downloads|IntentAction=null
2015-06-27 14:41:34.016|SERVICE_ONCREATE|Service=com.android.providers.downloads.
    ↪ DownloadService
2015-06-27 14:42:12.281|ACTIVITY_ONSTART|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:42:12.31|ACTIVITY_ONRESUME|Package=com.android.launcher|Activity=
    ↪ com.android.launcher2.Launcher
2015-06-27 14:42:13.983|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:14.009|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:14.399|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:14.409|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:14.551|START_SERVICE|Package=com.android.providers.contacts|
    ↪ SERVICE=com.android.providers.contacts.VoicemailCleanupService|userId=0|
    ↪ CallerUID=10002|CallerPid=1481
2015-06-27 14:42:14.818|SERVICE_ATTACH|Service=com.android.providers.contacts.
    ↪ VoicemailCleanupService
2015-06-27 14:42:14.834|SERVICE_ONCREATE|Service=com.android.providers.contacts.
    ↪ VoicemailCleanupService
2015-06-27 14:42:15.25|START_SERVICE|Package=com.android.musicfx|SERVICE=com.
    ↪ android.musicfx.Compatibility$Service|userId=0|CallerUID=10010|CallerPid
    ↪ =1504
2015-06-27 14:42:15.284|SERVICE_ATTACH|Service=com.android.musicfx.
    ↪ Compatibility$Service
2015-06-27 14:42:15.298|SERVICE_ONCREATE|Service=com.android.musicfx.
    ↪ Compatibility$Service
2015-06-27 14:42:15.422|START_SERVICE|Package=com.android.gallery3d|SERVICE=com.
    ↪ android.gallery3d.app.PackagesMonitor$AsyncService|userId=0|CallerUID
    ↪ =10031|CallerPid=1095
2015-06-27 14:42:15.468|SERVICE_ATTACH|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:42:15.482|SERVICE_ONCREATE|Service=com.android.gallery3d.app.
    ↪ PackagesMonitor$AsyncService
2015-06-27 14:42:16.591|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:16.593|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:16.664|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:16.672|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
```

```
2015-06-27 14:42:16.674|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:16.691|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:16.693|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:16.899|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:16.946|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:16.952|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:17.461|PACKAGE_REMOVED|PackageName=in.timelinestudios.ocha|uid
    ↪ =10053
2015-06-27 14:42:17.463|PACKAGE_REMOVED_ALL_USERS|PackageName=in.timelinestudios.
    ↪ ocha|uid=10053
2015-06-27 14:42:18.5|START_SERVICE|Package=com.android.keychain|SERVICE=com.
    ↪ android.keychain.KeyChainService|userId=0|CallerUID=1000|CallerPid=1831
2015-06-27 14:42:18.573|SERVICE_ATTACH|Service=com.android.keychain.
    ↪ KeyChainService
2015-06-27 14:42:18.589|SERVICE_ONCREATE|Service=com.android.keychain.
    ↪ KeyChainService
2015-06-27 14:42:19.762|START_ACTIVITY|Package=com.android.settings|Activity=com.
    ↪ android.settings.Settings|userId=0|CallerUID=10007|CallerPid=898|
    ↪ CallingPackage=com.android.launcher
2015-06-27 14:42:21.652|ACTIVITY_ONCREATE|Package=com.android.settings|Activity=
    ↪ com.android.settings.Settings
2015-06-27 14:42:22.177|ACTIVITY_ONSTART|Package=com.android.settings|Activity=
    ↪ com.android.settings.Settings
2015-06-27 14:42:22.201|ACTIVITY_ONRESUME|Package=com.android.settings|Activity=
    ↪ com.android.settings.Settings
2015-06-27 14:42:24.605|ACTIVITY_ONSTOP|Package=com.android.launcher|Activity=com
    ↪ .android.launcher2.Launcher
2015-06-27 14:42:28.375|START_ACTIVITY|Package=com.android.settings|Activity=com.
    ↪ android.settings.SubSettings|userId=0|CallerUID=1000|CallerPid=781|
    ↪ CallingPackage=com.android.settings
2015-06-27 14:42:28.993|ACTIVITY_ONCREATE|Package=com.android.settings|Activity=
    ↪ com.android.settings.SubSettings
2015-06-27 14:42:30.808|BIND_SERVICE|Package=com.android.defcontainer|SERVICE=com
    ↪ .android.defcontainer.DefaultContainerService|userId=0|CallerUID=1000|
    ↪ CallerPid=781
2015-06-27 14:42:32.093|ACTIVITY_ONSTART|Package=com.android.settings|Activity=
    ↪ com.android.settings.SubSettings
2015-06-27 14:42:32.175|ACTIVITY_ONRESUME|Package=com.android.settings|Activity=
    ↪ com.android.settings.SubSettings
2015-06-27 14:42:32.938|SERVICE_ATTACH|Service=com.android.defcontainer.
    ↪ DefaultContainerService
2015-06-27 14:42:33.0|SERVICE_ONCREATE|Service=com.android.defcontainer.
    ↪ DefaultContainerService
2015-06-27 14:42:35.589|ACTIVITY_ONSTOP|Package=com.android.settings|Activity=com
    ↪ .android.settings.Settings
2015-06-27 14:42:39.24|BIND_SERVICE|Package=com.android.defcontainer|SERVICE=com.
    ↪ android.defcontainer.DefaultContainerService|userId=0|CallerUID=1000|
    ↪ CallerPid=340
```

Bibliography

Annuzzi, J., Darcey, L., and Conder, S. 2013. *Introduction to Android Application Development: Android Essentials*. Developer's Library. Addison Wesley Professional.

Apvrille, A. 2015. Android security report in far less than 44 pages.

Bauer, A., Küster, J.-C., and Vegliach, G. 2012. Runtime verification meets android security. In *NASA Formal Methods*. Springer, 174–180.

Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S. A., and Albayrak, S. 2010. An android application sandbox system for suspicious software detection. In *Malicious and unwanted software (MALWARE), 2010 5th international conference on*. IEEE, 55–62.

Chou, A., Yang, J., Chelf, B., Hallem, S., and Engler, D. 2001. *An empirical study of operating systems errors*. Vol. 35. ACM.

Criswell, J., Dautenhahn, N., and Adve, V. 2014. Virtual ghost: protecting applications from hostile operating systems. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM, 81–96.

de Ponteves, K. and Apvrille, A. 2013. Analysis of android in-app advertisement kits. In *Virus Bulletin conference*.

Devos, M. 2014. Bionic vs glibc report. M.S. thesis, Ghent University.

Enck, W., Ongtang, M., McDaniel, P. D., et al. 2009. Understanding android security. *IEEE security & privacy 7,* 1, 50–57.

Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. 2011. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 627–638.

Feth, D. and Pretschner, A. 2012. Flexible data-driven security for android. In *Software Security and Reliability (SERE), 2012 IEEE Sixth International Conference on*. IEEE, 41–50.

Google. Android open source project documentation. http://developer.android.com/.

Google. 2015. Android security 2014: Year in review.

Heuser, S., Nadkarni, A., Enck, W., and Sadeghi, A.-R. 2014. Asm: A programmable interface for extending android security. *Intel CRI-SC at TU Darmstadt, North Carolina State University, CASED/TU Darmstadt, Tech. Rep. TUD-CS-2014-0063*.

Husted, N., Quresi, S., and Gehani, A. 2013. Android provenance: Diagnosing device disorders. In *TaPP*.

IDC, I. D. C. 2015. Smartphone os market share, q1 2015.

Isohara, T., Takemori, K., and Kubota, A. 2011. Kernel-based behavior analysis for android malware detection. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. IEEE, 1011–1015.

Lee, B., Lu, L., Wang, T., Kim, T., and Lee, W. 2014. From zygote to morula: Fortifying weakened aslr on android. In *IEEE Symposium on Security and Privacy*.

Lu, L., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., and Lu, S. 2014. A study of linux file system evolution. *ACM Transactions on Storage (TOS) 10,* 1, 3.

Mäntylä, M. V. and Lassenius, C. 2006. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering 11,* 3, 395–431.

McCarty, B. 2005. *SELinux: NSA's Open Source Security Enhanced Linux*. O'Reilly Series. O'Reilly.

OpenSuse. Understanding linux audit. http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html.

RedHat. System auditing. http://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html.

Reina, A., Fattori, A., and Cavallaro, L. 2013. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. *EuroSec, April*.

Schreiber, T. 2011. Android binder. *A shorter, more general work, but good for an overview of Binder. http://www. nds. rub. de/media/attachments/-files/2012/03/binder. pdf*.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., and Weiss, Y. 2012. andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems 38,* 1, 161–190.

Smalley, S. and Craig, R. 2013. Security enhanced (se) android: Bringing flexible mac to android. In *NDSS*.

Smalley, S., Vance, C., and Salamon, W. 2001. Implementing selinux as a linux security module. *NAI Labs Report 1,* 43, 139.

Vidas, T., Votipka, D., and Christin, N. 2011. All your droid are belong to us: A survey of current android attacks. In *WOOT*. 81–90.

Wikipedia. 2015. Information technology security audit — Wikipedia, the free encyclopedia. [Online; accessed 11-July-2015].

Yaghmour, K. 2013. *Embedded Android: Porting, Extending, and Customizing*. Oreilly and Associate Series. O'Reilly Media, Incorporated.

List of Publications

Jamsheed K, Praveen K. 2015. A Low Overhead Prevention of Android WebView Abuse Attacks. Accepted in *The Third International Symposium on Security in Computing and Communications (SSCC15)*, Communications in Computer and Information Science Series(CCIS).