

Android Forensics: Automated Data Collection and Reporting from a
Mobile Device

by
Justin Grover

Committee Members

Bill Stackpole (chair)

Dr. Tae Oh

Dr. Yin Pan

Thesis submitted in partial fulfillment of the requirements for the
degree of

**Master of Science in Computing Security and Information
Assurance**

Rochester Institute of Technology

B. Thomas Golisano College

of

Computing and Information Sciences

01/31/2013

UMI Number: 1534833

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 1534833

Published by ProQuest LLC (2013). Copyright in the Dissertation held by the Author.

Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against unauthorized copying under Title 17, United States Code



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346



Android Forensics: Automated Data Collection and Reporting from a Mobile Device

MITRE Sponsored Research
Project: Android System Integrity Measurement
Project No.: 51MSR606-TA

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

Approved for Public Release; Distribution Unlimited. 13-0585

©2013 The MITRE Corporation.
All rights reserved.

McLean, VA

Justin N Grover

January 2013

Abstract

As Android smartphones gain popularity, industry and government will face increasing pressure to integrate them into their environments. The implementation of these devices on an enterprise can save on costs and add capabilities previously unavailable; however, the organizations that incorporate this technology must be prepared to mitigate the associated risks. These devices can contain vast amounts of personal and work-related data that can impact internal investigations, including (but not limited to) those of policy violations, intellectual property theft, misuse, embezzlement, sabotage, and espionage. Physical access has been the traditional method for retrieving data useful to these investigations from Android devices, with the exception of some limited collection abilities in commercial mobile device management systems and remote enterprise forensics tools. As part of this thesis, a prototype enterprise monitoring system for Android smartphones was developed to continuously collect many of the data sets of interest to incident responders, security auditors, proactive security monitors, and forensic investigators. Many of the data sets covered were not found in other available enterprise monitoring tools. The prototype system neither requires root access privileges nor exploiting weaknesses in the Android architecture for proper operation, thereby increasing interoperability among Android devices and avoiding a spyware classification for the system. An anti-forensics analysis on the system was performed to identify and further strengthen areas vulnerable to tampering. The results of this research include the release of the first open-source Android enterprise monitoring solution of its kind, a comprehensive guide of data sets available for collection without elevated privileges, and the introduction of a novel design strategy implementing various Android application components useful for monitoring on the Android platform.

Acknowledgements

In addition to the thesis committee members, the author would like to thank the following people for reviewing this work:

Brian Shaw, Chuck Bonneau, Elvira Caruso, Eoghan Casey, Jared Ondricek, Marc Brooks, Mark Guido, Meg Cormier, Michael Peck, Monica Grover, John Parlee, and Shane Shroeder.

Table of Contents

List of Figures	viii
List of Tables	ix
1 Introduction	1
2 Scope	3
3 Background	3
3.1 Android Framework	4
3.2 Key Application Components	5
3.2.1 Android Activities	5
3.2.2 Android Services	5
3.2.3 Android Content Providers	6
3.2.4 Android Broadcast Receivers	6
3.3 Android Application Security	6
3.4 Rooting: Advantages and Disadvantages	7
3.5 Smartphone Investigations	8
3.6 Privacy Concerns	11
4 Related Work	13
4.1 Commercial Products	13
4.2 Academic Work	15
5 DroidWatch	17
5.1 Overview	17

5.2	Design and Implementation.....	17
5.2.1	System Architecture	17
5.2.2	User Consent to Monitoring	18
5.2.3	Data Collection Components.....	19
5.2.4	Design Strategy	20
5.2.5	Local Storage.....	21
5.2.6	Enterprise Server	21
5.2.7	Data Process Flow	22
5.2.8	Data Sets.....	22
5.3	Analysis and Evaluation.....	33
5.3.1	General Usage Trends	35
5.3.2	Suspicious Contacts and Communications.....	35
5.3.3	Location Monitoring.....	37
5.3.4	Internet History.....	38
5.3.5	Malicious Applications.....	39
5.3.6	Attribution	40
5.4	Anti-Forensics	41
5.4.1	Destroying Evidence	41
5.4.2	Hiding Evidence	42
5.4.3	Altering Evidence Sources	43
5.4.4	Counterfeiting Evidence.....	44

5.4.5	Detecting Forensics Tools	45
6	Future Work	45
6.1	Resolve Current Issues	46
6.2	Additional Data Collections	46
6.3	Anti-Tampering Mechanisms	47
6.4	Future Considerations.....	48
7	Conclusion.....	50
	Appendix A: Android Manifest.....	51
	Appendix B: Juniper Product Evaluation	53
	Appendix C: PHP Server Code	64
	Appendix D: DroidWatch Detectors	65
	Appendix E: Content Providers.....	66
	Appendix F: Glossary.....	67
	Appendix G: Acronyms.....	71
	Bibliography	72

List of Figures

Figure 1. Mobile Phone Trends in the U.S.	1
Figure 2. Android System Architecture Framework	4
Figure 3. Obtaining GPS Coordinates Through the Android Framework.....	5
Figure 4. DroidWatch System Architecture	18
Figure 5. User Consent Banner	19
Figure 6. Data Process Flow Diagram.....	22
Figure 7. Events Logged Over 24 Hours.....	34
Figure 8. Detected Screen Unlock Actions (Splunk)	35
Figure 9. Communication Related Search Results (Splunk)	36
Figure 10. Photo and MMS Search Results (Splunk).....	37
Figure 11. Location Related Search Results (Splunk).....	38
Figure 12. Browser History Results (Splunk)	39
Figure 13. App Related Search Results (Splunk).....	40
Figure 14. Device Account Search Results (Splunk)	41
Figure 15. Mobile Security Gateway Settings.....	55
Figure 16. MSG SMS Interface.....	58
Figure 17. GPS Capture Interface	59
Figure 18. Image Capture Interface.....	60
Figure 19. App Capture Interface.....	61
Figure 20. Contacts Capture Interface.....	62

List of Tables

Table 1. Design of Data Set Collections	23
Table 2. Junos Pulse Mobile Security Suite - Capability Chart	56
Table 3. DroidWatch Detectors	65
Table 4. Content Provider URIs Used in this Research	66

This page was intentionally left blank.

1 Introduction

In the United States (U.S.), 121.3 million people—roughly one third of the population—owned a smartphone as of October 2012 (comScore, Inc., 2012; U.S. Census Bureau, 2010). The major players in the smartphone market included Google, Apple, Research In Motion (RIM), and Microsoft. Google’s Android platform showed significant gains, up to 53.6% of the market, while RIM continued its decline to 7.8% (comScore, Inc., 2012). As Google’s prevalence in the smartphone market grows, so will the pressure for organizations to move from existing mobile options to Android. These organizations include U.S. government agencies, some of which are considering transitioning from RIM to new smartphone systems (Kang, 2012; Lai, 2012; Marks, 2012; Rauf, 2012). Some organizations have begun offering personnel the ability to use personal devices (including Android smartphones) on corporate networks under Bring Your Own Device (BYOD) policies (Citrix, 2011). BYOD is forecasted to spawn a \$181.39 billion industry by 2017 (MarketsandMarkets, 2012). Figure 1 depicts the mobile phone market trends in the U.S. through the second quarter of 2012.

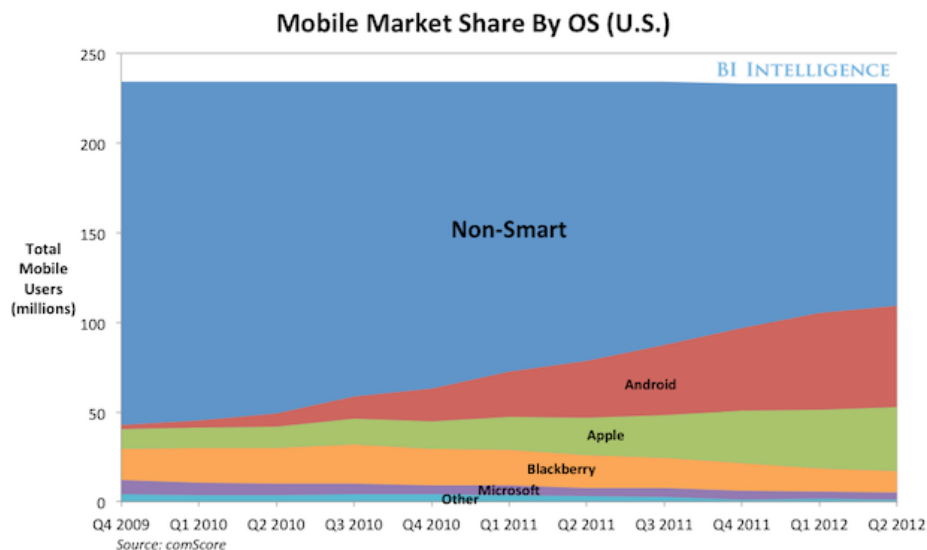


Figure 1. Mobile Phone Trends in the U.S.¹

¹ Image courtesy of Business Insider. Retrieved from <http://www.businessinsider.com/the-smartphone-race-is-just-getting-started-2012-8>.

Given the mobile device trends in government and industry, the challenge of securing smartphones on an enterprise has emerged. Most vendors do not design smartphones primarily for business use. Instead, their efforts focus on consumers who will utilize their phones as personal devices. RIM is an exception, as corporate customers can maintain a BlackBerry Enterprise Server for device management. Unlike RIM, Android device vendors do not ship with built-in mobile device management (MDM) systems. Third-party MDM is a quickly evolving field that addresses the security gaps left by the smartphone industry. Within the next five years, 65% of all enterprises are expected to adopt third-party MDM (Petty, 2012). BYOD increases the need for MDM systems due to the inherent risks involved with allowing untrusted devices on an enterprise; organizations need strict enforcement of mobile device security policies to mitigate the BYOD risks.

Research on several leading MDM products revealed a general lack of features that perform automated collections of forensic data from Android devices at the enterprise level. The availability of this data would aid common security programs found within organizations including incident response, security auditing, proactive security monitoring, and forensic investigations. According to the 2010/2011 Computer Security Institute Computer Crime and Security Survey, 61.5% of respondents from various companies reported that internal audits were performed within their organizations as a security mechanism. Additionally, 44% reported that data-loss prevention and user-content monitoring programs were in place (Richardson, 2010). These statistics show that many organizations are aware of insider threat enterprise risks and have taken steps to mitigate them; however, given the lack of technology to monitor Android smartphones, many actions performed on these devices are not covered. The lack of monitoring options, coupled with the high impact that continuously collected smartphone data can have on

internal investigations, led to the proposal and development of a prototype solution named DroidWatch, which is the subject of this thesis.

2 Scope

This work focuses on the design and implementation of an Android application (further known as “app”) that automates the collection of useful data for internal investigations including (but not limited to) policy violations, intellectual property theft, misuse, embezzlement, sabotage, and espionage. Data was collected from a Samsung Galaxy S II Epic 4G Touch Android smartphone running Gingerbread 2.3.6. It was then sent to a central server running Ubuntu 12.04, PHP 5.3.10, MySQL 5.5.28, Apache 2.2.22, and Splunk 5.0.1. Capabilities such as anti-virus, root detection, and protections against the app’s termination or uninstallation are necessary for system and enterprise security, but are out of scope for this research. All data collected by the implemented app will occur with user consent by means of a banner (henceforth referred to as “user consent banner”) similar to those commonly found on corporate or government networks. Data will not be obtained with root privileges or exploitation of the Android architecture. References made to “future work” are meant to assume work performed by the author, the author’s company, or other future researchers.

3 Background

The following subsections discuss the basics of the Android Framework (Section 3.1), fundamental app components (Section 3.2), Android app security model (Section 3.3), implications of rooting a device (Section 3.4), handling of mobile devices in current investigations (Section 3.5), and privacy concerns related to smartphones (Section 3.6).

3.1 Android Framework

The Android System Architecture Framework (see Figure 2) describes the interaction of underlying Android device components in four distinct layers: Applications, Application Framework, Libraries, and the Linux Kernel. Each layer relies on the next layer to function properly.

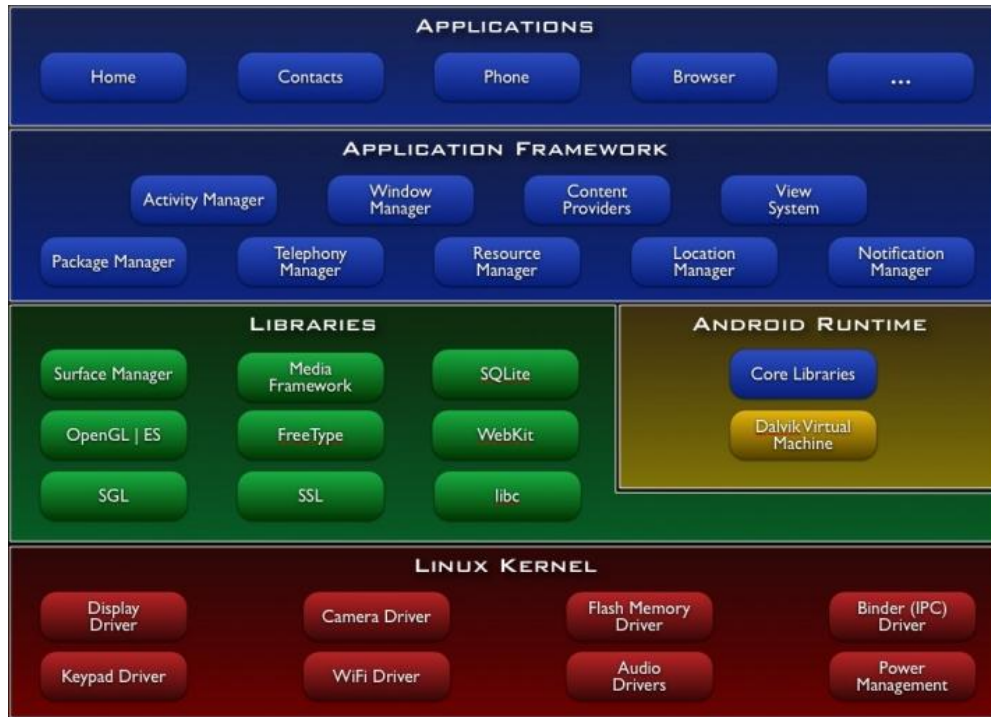


Figure 2. Android System Architecture Framework²

Figure 3 illustrates a specific implementation of the Android System Architecture Framework. For an app to receive a new set of Global Positioning System (GPS) coordinates from an Android device, it must call the `LocationManager` Java class from the Application Framework Layer, which uses the Java Native Interface to interact with the `libgps.so` C library, which finally calls the appropriate kernel driver to get the device's current location (Townsend, 2010).

² Portions of this page are reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License. Image retrieved from <http://developer.android.com/about/versions/index.html>.

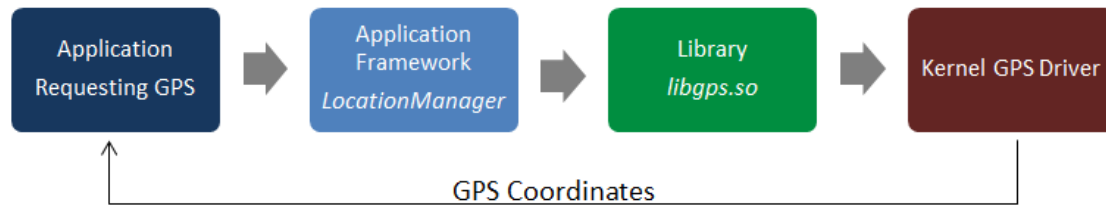


Figure 3. Obtaining GPS Coordinates Through the Android Framework

3.2 Key Application Components

Apps use four key Android framework components: activities, services, content providers, and broadcast receivers. Each app component, described in the sections that follow, serves a distinct and useful purpose.

3.2.1 Android Activities

According to Google’s official documentation, activities are individual screens that implement a user interface. They display information, prompt user interaction, and start other activities within apps (Google, n.d.-b). In DroidWatch, activities are rarely used; most of the work happens in the background so as to not impact the overall user experience. Activities are only viewable in the user consent banner, which prompts an acceptance of the stated terms and conditions.³

3.2.2 Android Services

Services are long-running background operations requiring no user interaction. Other application components, such as activities, can launch services and have them persist even when other apps and services are running (Google, n.d.-b). DroidWatch relies on a service that constantly runs to spawn data collections and transfers.

³ More information about the user consent banner can be found in Section 5.2.2.

3.2.3 Android Content Providers

Content providers are app components that manage the access and sharing of application data.

Apps interface with content providers through mechanisms called content resolvers, which

communicate with content providers through predefined uniform resource identifiers (URIs).

Permission to access a content provider's data is verified upon the initial registration of a content resolver and during each query performed through it (Google, n.d.-b). DroidWatch uses content providers in two ways:

1. Reading data stored within other applications (e.g., call logs and contacts)
2. Reading and writing data stored within the DroidWatch app.

3.2.4 Android Broadcast Receivers

Broadcast receivers are app components that handle and respond to broadcast system events on an Android device. They are intended to perform a minimal amount of work and often serve as gateways to the other app components (Google, n.d.-b). They are implemented into DroidWatch to detect events such as incoming short message service (SMS) messages and app installs.

3.3 Android Application Security

Google's security model for Android apps involves the declaration of permissions within an app's `AndroidManifest.xml` file (further known as an "AndroidManifest"). By default, an app with no requested permissions cannot "perform any operations that would adversely impact other applications, the operating system, or the user" (Google, n.d.-e). This means that an app cannot access the private data of other apps, use network services, write to the internal/external memory, or perform other basic functions. A newly downloaded app must present its declared permissions to the user prior to installation. The AndroidManifest for DroidWatch is provided in Appendix A.

Once an app's permissions have been presented, a user can assess the inherent risks of installing it. If consent is not granted, the app is not installed. If the user accepts the risks, an app becomes enabled to send and receive information to the newly allowed areas of a phone through transfer mechanisms called intents. Intents are the only legitimate way of accessing data inside another app without elevated or "root" privileges since all apps are sandboxed and run separately under different user IDs.

Android device users are subject to the same security and permissions as apps and cannot navigate into an app's private folder directly by default. For example, to access the `/data/data/com.droidwatch/database` directory of the DroidWatch app, a user must have the DroidWatch app's user ID or root privileges, which is not possible under normal circumstances. Users do, however, have the ability to uninstall apps such as DroidWatch (methods to prevent this are out of scope for this research), thus erasing any privately stored data within the app.

3.4 Rooting: Advantages and Disadvantages

Rooting enables users to perform higher privileged functions on a device that ordinarily are not possible under normal privileges and can be used for legitimate or illegitimate purposes.

Individuals with malicious intentions may want to circumvent restrictions imposed by carriers, policies, or applications to tamper with an app like DroidWatch or its underlying data. While DroidWatch is susceptible to attacks by root users, the detection and prevention of root tampering is out of scope for this research. In an enterprise setting, any mobile device with root privileges can be deemed a threat because of its ability to bypass an organization's security controls (Souppaya & Scarfone, 2012).

Root access can be used legitimately by forensic investigators to extract data from a device. In addition, security apps may use root privileges to gain access to unwatched parts of a phone. However, rooting should be avoided whenever possible. The rooting process typically exploits a security flaw in a specific device or operating system version and may lead to further security vulnerabilities. Rooting also alters portions of a device (an action that is contradictory to forensic practices); nonetheless, rooting may be unavoidable depending on the circumstances and types of data needed (Vidas, Zhang, & Christin, 2011). While root access may increase the number of features for an app like DroidWatch, the consequences can undermine the system's security and put the smartphone provider's warranty at risk.

3.5 Smartphone Investigations

Law enforcement and government agencies are the primary players in the mobile security field, as they are authorized to investigate crimes and secure sensitive government information. Corporations are also very interested in mobile security to protect themselves from commercial espionage, financial theft, and intellectual property theft. Private interests involving divorce settlements, custody battles, estate disputes, etc. also gain from advances in the field (Hoog, 2011).

Consequently, the types of investigations that would benefit from user data collections and monitoring of smartphones are law enforcement investigations, internal investigations, and private investigations. Law enforcement investigations are conducted by government personnel and often involve a crime scene, rigid forensic device acquisition and preservation procedures, and strict evidence handling through chain-of-custody so that any results gleaned from analysis hold up in court. Private investigations are conducted by licensed private investigators, suspicious

spouses or parents, hackers, etc. and often involve financial, legal, or personal matters (Bureau of Labor Statistics, U.S. Department of Labor, 2012).

This research focuses on internal investigations performed by personnel, contracted or otherwise, within an organization (e.g., incident responders, security auditors, analysts performing proactive monitoring, etc.) to investigate potential policy violations, intellectual property theft, misuse, embezzlement, sabotage, espionage, and other inquiries or allegations. Internal investigators are not required to conform to the strict forensic acquisition and preservation procedures of law enforcement investigations but typically try to adhere to commonly practiced forensic techniques and rules. Some U.S. states have or are considering legislation that further requires computer forensic examiners and incident responders be licensed to ensure that these individuals have received proper training (Garnett, 2010).

To obtain valuable smartphone data for internal investigations, physical access to a device is typically required. An exception to this is EnCase Enterprise,⁴ which as of October 2012 performs remote forensic imaging of Android devices over a network. Some MDMs can also perform limited monitoring (specific products are covered in Section 4.1). Assuming that a mobile device is physically retrieved, investigators can use a variety of tools to take logical or physical snapshots of a device's current state. Logical snapshots contain a dump of records or files, while physical snapshots are bit-by-bit copies of sectors or pages (Garfinkel, 2011). The following tools are available to capture information from Android smartphones (Hoog, 2011):

⁴ More information about Encase Enterprise can be retrieved from <http://www.guidancesoftware.com/encase-enterprise.htm>.

- Android Debug Bridge
- AFLogical
- AFPhysical
- viaExtract
- Cellebrite UFED
- EnCase Neutrino
- Micro Systemation XRY
- Paraben Device Seizure
- AccessData MPE+.

Some tools listed above are handheld hardware devices and others are software products; however, all work through a universal serial bus (USB) connection and require physical access to the smartphone to function. In addition, many of the tools require root access (Valle, 2013). Related work has found that when two distinct tools target and output the same data set, the results are slightly different because of the varying acquisition techniques employed in each tool (Kovacik & O'Day, 2010; Lessard & Kessler, 2010). For example, a tool may find a few text messages not found by another.

The types of smartphone content that are valuable to investigations are found in the National Institute of Standards and Technology's Guidelines on Cell Phone Forensics. These data sets include (Jansen & Ayers, 2007):

- Subscriber and equipment identifiers
- Date/time, language, and other settings
- Phonebook information
- Appointment calendar information
- Text messages
- Dialed, incoming, and missed call logs
- Email
- Photos
- Audio and video recordings
- Multi-media messages
- Instant messaging and Web browsing
- Electronic documents
- Location information.

Many of these data sets are incorporated into DroidWatch and are detailed in Section 5.2.8.

3.6 Privacy Concerns

As more data is generated and stored on smartphones, more privacy concerns will emerge. In June 2010, the U.S. Supreme Court ruled in *City of Ontario v. Quon* that an audit performed on a government-owned, alphanumeric pager did not violate the Fourth Amendment, which protects against unreasonable search and seizure in the U.S. The pager was operated by Jeff Quon, a member of the Ontario, California police department's Special Weapons and Tactics (SWAT) team. His device exceeded the allotted monthly character limits, causing surcharges to be incurred by the city, which Quon reimbursed. When his supervisor reviewed the audit of Quon's device, Quon was allegedly reprimanded for sending a large quantity of non-work related text messages, some of which were sexually explicit. Quon argued that his rights were violated since he paid the incurred surcharges (U.S. Supreme Court, 2010). The Court's opinion stated that this verdict was not meant to be broad; however, there are potential implications to BYOD policies since it can be construed that Quon's mobile device, although provided by his employer, was partially his since he paid for some of the charges. This interpretation may lead organizations to install monitoring apps like DroidWatch on all enterprise smartphones, including those that are personally owned.

Mobile phone privacy was discussed in the case of Fannie Garcia, a former police dispatcher for the city of Laredo, Texas. Garcia was fired after a coworker's wife discovered inappropriate content on Garcia's personal smartphone and reported it to supervisory personnel. The phone was removed from an unlocked storage locker where Garcia worked without Garcia's permission. The Fifth Circuit U.S. Court of Appeals affirmed a district court's ruling that Garcia's phone content was not protected under the Stored Communications Act (SCA) and, as a result, upheld her

dismissal (U.S. Court of Appeals, 5th Circuit, 2012). The ruling means that the SCA does not protect data stored on a smartphone, because the data is not stored within a “facility through which wire and electronic communications are kept in temporary storage, or as back-up storage.” Therefore, data collected through an app like DroidWatch may not be protected by the SCA.

Another case concerning mobile phone privacy is that of the California-based company Carrier IQ⁵ and its software customers (Apple, HTC, Samsung, Motorola, AT&T, Sprint, and T-Mobile). These companies were sued in 2011 for violating the Federal Wiretap Act. There were allegations that the Carrier IQ software spied on more than 150 million smartphone users across the U.S. The defendant companies countered that the software was meant for tracking network health issues, such as dropped calls, and did not take advantage of software bugs that could allow content tracking for email and text messages. Several federal and state court cases involving Carrier IQ’s software are still pending (Davis, 2012; Epstein, 2011). To avoid the legal and spyware issues referenced in the Carrier IQ case, DroidWatch is careful to ensure that users are aware that they are operating in a monitored environment.

U.S. v Ziegler ruled that an organization has a right to monitor its own equipment, on which employees have little to no expectation of privacy. One key factor in the ruling was that Jeffrey Ziegler knew of his company’s policy stating that the company’s computers were subject to monitoring (U.S. Court of Appeals, 9th Circuit, 2007). Organizations often inform users of monitoring policies through the display of network banners;⁶ however, network banners were not specifically referenced in *U.S. v. Ziegler*. Nonetheless, the court found that the policy handbook

⁵ More information about Carrier IQ can be retrieved from <http://www.carrieriq.com/>.

⁶ For more information about network banners, Department of Justice guidelines are available and can be retrieved from <http://www.justice.gov/criminal/cybercrime/docs/ssmanual2009.pdf>.

presented to new hires at Ziegler’s company, coupled with verbal instructions from management, served as a sufficient warning that employees’ computer actions were monitored.

DroidWatch displays a user consent banner during a phone’s boot process to inform users of privacy expectations and to garner their consent (more details on the consent banner’s implementation can be found in Section 5.2.2). This helps the app avoid a spyware classification and acts as a deterrent for system misuse. Previous work reveals that awareness of computer monitoring “has a significant effect on users’ perceived certainty and severity of sanctions,” which provides empirical evidence that the display of user consent banners deter misuse (D'Arcy, Hovav, & Galletta, 2009).

4 Related Work

The following subsections discuss several products and efforts that complement or have otherwise influenced this research. Commercially available products are discussed in Section 4.1 and academic efforts are described in Section 4.2 .

4.1 Commercial Products

Third-party MDM products increase the overall security of an enterprise that has mobile devices; however, most MDMs do not include in-depth user monitoring capabilities. Instead, they focus on the following areas (National Security Agency, 2012):

- Data-at-rest protection
- Remote wipe
- Application policies
- Enterprise app deployment
- Version reporting
- Detection of user-initiated platform modification
- Virtual private network

- Wi-Fi and cellular network restrictions
- USB Connection policy
- Certificate trust management

Some leading MDM offerings,⁷ such as Zenprise, AirWatch, and MobileIron, offer limited monitoring capabilities (e.g., GPS tracking and SMS monitoring), but fail to collect other available data sets that are helpful to incident responders, security auditors, proactive security analysts, and other internal investigators. Of the researched MDM products, Juniper's Junos Pulse Mobile Security Suite (version 3.0R3) offers the most user monitoring capabilities and was evaluated as part of this research effort. Findings show that the product collects and monitors roughly 50% of the data sets covered in DroidWatch; however, the data must be stored and hosted by Juniper-controlled systems. This may preclude an organization's requirement to store its audit data internally. Refer to Appendix B for a full evaluation.

Specialized apps outside the MDM realm, such as Lookout,⁸ are not designed for enterprise use and typically target a limited number of data sets on a phone. Other commercially available products, such as personal "spy" apps⁹ (e.g., Mobistealth, StealthGenie, FlexiSpy, and Mobile Spy), collect many of the same data sets as DroidWatch, but have limiting factors such as requirements for elevated privileges and a lack of enterprise storage and analysis capabilities.

⁷ More information about each researched MDM product can be found on its product website:

- <http://www.zenprise.com/products/zenprise-mobilemanager>
- <http://www.air-watch.com/downloads/brochures/solution-brochure.pdf>
- <http://www.mobileiron.com/en/smartphone-management-products/advanced-management>
- <http://www.juniper.net/us/en/products-services/software/junos-platform/junos-pulse/mobile-security/>

⁸ More information about Lookout can be retrieved from <https://www.lookout.com>.

⁹ More information about each researched "spy" product can be found on its website:

- <http://www.mobistealth.com/>
- <http://www.stealthgenie.com/>
- <http://www.flexispy.com/>
- <http://www.mobile-spy.com/>

They are also often classified as spyware because they collect personal information without knowledge of the user (Juniper Networks, 2012; TechGuy, 2009).

Remote enterprise forensic collection tools also seek to increase enterprise security. Google Rapid Response (GRR), an open-source project introduced at the 2011 Digital Forensics Research Workshop (DFRWS), allows forensic investigators and incident responders to locate and forensically acquire evidence from a large number of machines over a network (Cohen, Bilby, & Caronni, 2011). Commercially available solutions similar to GRR are EnCase Enterprise, AccessData Enterprise, F-Response Enterprise Edition, and Mandiant Intelligent Response, among others.¹⁰ EnCase Enterprise works on Android smartphones, while the others do not. However, all the aforementioned remote forensic tools differ from DroidWatch because they do not continuously collect and store data. Instead, they take one-time snapshots of data when instructed by an investigator. Code from DroidWatch could be used to help extend the capabilities of these tools to collect data from Android devices.

4.2 Academic Work

Several scholarly efforts have shaped this research effort. One system, proposed by Xinfang Lee et al., uses an Android app located on a Secure Digital Card (SDCard) to extract data quickly from a smartphone onto the same SDCard using the Android Application Programming Interface (API). This fundamental application-level approach to collecting data is similar to DroidWatch, except their system does not monitor a device continuously. The work by Lee, Yang, Chen, and

¹⁰ More information about each researched remote forensic tool can be found on its website:

- <http://code.google.com/p/grr/>
- <http://www.guidancesoftware.com/encase-enterprise.htm>
- <http://www.accessdata.com/products/digital-forensics/ad-enterprise>
- http://www.f-response.com/index.php?option=com_content&view=article&id=171&Itemid=80
- <http://www.mandiant.com/products/platform/>

Wu helped highlight various data sets that can be obtained without root privileges (2009).

AFLogical, released as an open-source product by viaForensics, takes a similar approach. It uses a specialized SDCard and expands on the number of retrieved data sets (Hoog, 2010). Follow-on work that substituted cloud computing for the SDCards was proposed by Chung-Huang Yang and Yen-Ting Lai (2012). Unlike DroidWatch, the aforementioned research efforts require physical access and take one-time snapshots of data from an Android smartphone.

Related work by Angel Villan and Josep Esteve described a native Android app that performs real-time monitoring, similar to virtual network computing (VNC), on a smartphone without root privileges (Villan & Esteve, 2011). Their research involved streaming a user's screen to a remote location for usability purposes (i.e., for use at corporate help desks). The ability to watch a user's screen could complement the features of DroidWatch in the future.

Research presented by Clay Shields et al. introduced a forensic acquisition and monitoring system named Proactive Object Fingerprinting and Storage (PROOFS), the first system to perform true forensic acquisitions over a network in a continuous, proactive manner using a novel object fingerprinting approach (Shields, Frieder, & Maloof, 2011). While PROOFS does not operate on Android smartphones, it does highlight the criteria required for a monitoring tool to be considered forensically sound. Future work on DroidWatch includes the incorporation of some of these criteria discussed in their research.

Previous anti-forensics work on the Android platform was instrumental in evaluating how DroidWatch could be compromised or thwarted. Several general anti-forensic concepts, such as data hiding and destruction, were ported to Android and discussed by Alessandro Distefano et al. (Distefano, Me, & Pace, 2010). The work served as a guide during the assessment of anti-forensic techniques to DroidWatch app components (Section 5.4). Research presented at the 45th Hawaii

International Conference on System Sciences detailed another anti-forensics concept involving the real-time detection of a forensics tool on an Android smartphone (Azadegan, Yu, Liu, Sistani, & Acharya, 2012). This concept was also considered during the aforementioned assessment of DroidWatch.

5 DroidWatch

5.1 Overview

DroidWatch is an automated system prototype composed of an Android application and an enterprise server. After obtaining user consent, the app continuously collects, stores, and transfers forensically valuable Android data to a Web server without root privileges. The following subsections discuss the details of DroidWatch's design and implementation, the analysis and evaluation of its results, and the potential anti-forensic opportunities available in the app.

5.2 Design and Implementation

5.2.1 System Architecture

DroidWatch's system architecture is described in Figure 4. Each descending layer represents a higher level of abstraction. Various collection, storage, and transfer components are handled by a service that facilitates actions within the app and the enterprise server. The individual components labeled within the diagram are explained throughout the design subsections.

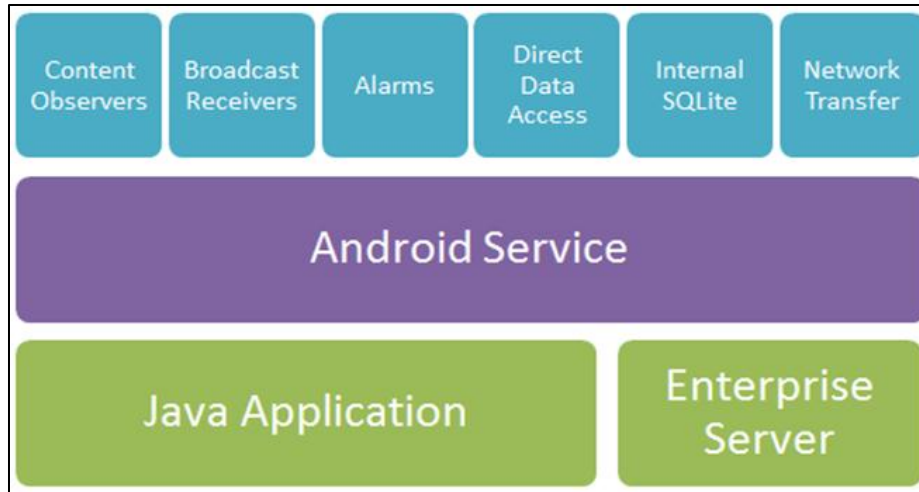


Figure 4. DroidWatch System Architecture

5.2.2 User Consent to Monitoring

The DroidWatch app is dependent upon a user consent banner (see Figure 5) that appears when the app is started during the phone's boot process or launched manually through the Android app menu. Upon acceptance of its terms and conditions, which are configurable prior to the app's installation in the `strings.xml` file, a long-running service is launched to perform the data collection, storage, and transfer components of the system architecture. The option to reject the user consent terms is also presented for the purposes of this research, although organizations may wish to omit this option in real deployments. Rejecting or closing the user consent banner causes the DroidWatch service not to run. The banner is implemented using an activity that is registered in the `AndroidManifest` to receive the `android.intent.action.BOOT_COMPLETED` broadcast to ensure its display during the boot process.



Figure 5. User Consent Banner

5.2.3 Data Collection Components

The DroidWatch app utilizes several Android app components to perform its data monitoring and collection. These components (further known as “data collection components”) are content observers, broadcast receivers, and alarms.

5.2.3.1 Content Observers

Content observers are mechanisms that, when associated with content providers, receive notifications when a targeted data set’s underlying SQLite database content is modified (Google, n.d.-d). However, these real-time notifications do not contain information about what changed. To gain meaningful data from content observers, additional processes must be in place to determine the altered content. Content observers are prone to false positive and duplication issues since any changes to a data set’s underlying database, even on data not collected, can trigger notifications.

5.2.3.2 Broadcast Receivers

Broadcast receivers handle and respond to system-wide broadcast messages. Like content observers, they involve real-time notifications that may lack content details; however, they are not generally prone to false positive or duplication issues because of their implementation details.

Broadcast receivers react to atomic events and have ten seconds to perform their functions before they are terminated by the Android operating system (Google, n.d.-c).

5.2.3.3 *Alarms*

Alarms are scheduled operations configured in DroidWatch to periodically query content providers or directly call static Java methods to pull in new data. They are reliable, but only run at designated times. Data that is deleted or modified between collection runs may not be captured in its original form, if at all. In addition, other processes must be in place to ensure that duplicate events from data sets are not added to the local SQLite database (i.e., an alarm that runs twice over the same data set may detect the same event both times).

5.2.4 **Design Strategy**

To simplify the development process of an app that performs monitoring through continuous or periodic data collections, a general design strategy for prioritizing data collection components was applied throughout the development of DroidWatch, as presented below.

Broadcast Receivers → Content Observers → Alarms

The strategy is based on the aforementioned pros and cons (refer to Section 5.2.3) and the relative ease of implementation. First, data sets should be analyzed to determine if they generate system broadcasts. If they do, broadcast receivers should be considered as the collection component. If broadcasts are not available, consider content observers for implementation. Alarms should be used if broadcasts and content observers are unavailable or ineffective for the targeted data collection.

5.2.5 Local Storage

All collected data is stored temporarily in a local SQLite database on the phone and is configured to be accessible to the DroidWatch app only. Standard Structured Query Language (SQL) database functions such as selections, insertions, and deletions are handled by a custom-developed DroidWatch content provider. Using a content provider instead of direct SQLite database calls allows each of DroidWatch's collections to perform data storage functions in a thread-safe and structured manner.

A scheduled alarm periodically transfers the local SQLite database file over hypertext transfer protocol secure (HTTPS) POST to the enterprise server for processing. The transfer process is designed to run in the background with minimal impact on a user's experience, helped by the relatively small size (50-100 kilobytes) of the database file. The file size is minimized by clearing events dated prior to the start time of a successful transfer event. In addition, the clearing mechanism lessens the risk of data loss should the phone be compromised.

5.2.6 Enterprise Server

The enterprise server prototype, to which the collected data flows, is an Ubuntu Linux virtual machine on a local, private network running Apache, PHP: Hypertext Processor (PHP), MySQL, and Splunk. Apache was configured with a self-signed secure socket layer (SSL) certificate, which was included as an asset file within the DroidWatch app. This allows data to be transferred over an HTTPS connection. PHP code handles the secure POST uploads, extracts the events from the uploaded SQLite file, and stores them in an enterprise MySQL database. Note that the PHP code used on the server is available in Appendix C. Splunk periodically pulls data from the MySQL database and makes the events available in its interface for analysis and reporting.

5.2.7 Data Process Flow

The data process flow within the DroidWatch app is depicted in Figure 6. Data collection and storage is a continuous process, with transfers scheduled every two hours by default (this value is configurable). Upon a successful transfer to the enterprise server, events dated prior to the transfer are wiped from the local phone database. File transfer attempts that fail are logged in the database and do not result in the wiping of any events.

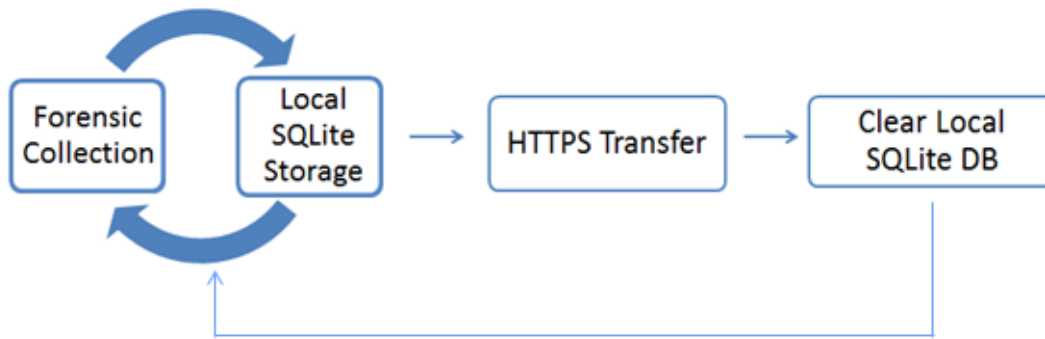


Figure 6. Data Process Flow Diagram

5.2.8 Data Sets

Table 1 lists the targeted data sets collected for this thesis. Each data set can be configured through `droidwatch.properties`, an asset file included within the app that allows for the adjustment of collection intervals (i.e., how long the system waits between collections). Organizations can choose to omit a data set collection by setting its interval value to zero. Fifteen unique data sets could be accessed; two data sets—emails and account passwords—were explored, but not used (as explained below).

Mechanisms to access data within the email app are not part of the standard Android Software Development Kit (SDK) and are not documented in the API (CommonsWare, 2010).

Furthermore, the email app is restricted by a `signatureOrSystem` permission, which prohibits third-party apps from accessing its private data (Android Open Source Project, 2008).

Access to account passwords is protected by similar protection schemes; the calling app must declare the `android.permission.AUTHENTICATE_ACCOUNTS` permission in the AndroidManifest and have its user identification (ID) match that of the requested account (Google, n.d.-a).

The table shows some data sets collected using more than one collection component. For example, Multimedia Message Service (MMS) messages are detected by using a broadcast receiver and an alarm; the component used depends on the message direction. Data set details are provided in the following subsections.

Table 1. Design of Data Set Collections

Data Set	Collection Component Used		
	BroadcastReceiver	ContentObserver	Alarm
App Installs / Removals	✓		
Browser Navigation History			✓
Browser Searches			✓
Calendar Events			✓
Call Logs (Incoming / Outgoing / Missed)		✓	
Contacts Added		✓	
Device Accounts*			
Device ID (e.g., IMEI, MEID, ESN)			✓
GPS Location			✓
Location Settings (e.g., GPS turned off)	✓		
MMS (Incoming / Outgoing)	✓		✓
Pictures Added		✓	
Screen Lock Status (e.g., Unlocked)	✓		
SMS (Incoming / Outgoing)	✓	✓	
Third-Party App Logs (Logcat)			✓

**Device account information is collected, but not through any of the listed collection components. Instead, accounts are directly retrieved through the Android API when the DroidWatch service starts.*

5.2.8.1 App Installs and Removals

When an app is installed or removed, a system-wide broadcast is sent across an Android phone.

Apps are not required to declare permissions in the AndroidManifest to receive this broadcast,

although they must register a broadcast receiver and include intent-filters to handle the `Intent.PACKAGE_ADDED` and `Intent.PACKAGE_REMOVED` events. Additional information, including the name of the involved package, is extracted from the intents and logged accordingly.

5.2.8.2 *Browser Navigation History*

Uniform Resource Locators (URLs) of websites accessed through the built-in Android Web browser are collected through an alarm. While a content observer is available to detect browser changes instantaneously, it is a noisy option, as change notifications are triggered constantly—even when the browser is closed. To avoid battery and performance degradation, an alarm is scheduled and configured to perform a query on the `android.provider.Browser.BOOKMARKS_URI` content provider URI every six hours by default. This content provider contains options to access a browser's visited URL history, counts of each visited URL, and a list of bookmarked websites; however, this research only focuses on the visited URL history and access dates. To gain access to the browser history data, the `com.android.browser.permission.READ_HISTORY_BOOKMARKS` permission must be included in the `AndroidManifest`. To mitigate duplication issues that occur with data retrieved through alarms, checks are made on the local DroidWatch database to ensure that an inserted event is unique.

5.2.8.3 *Browser Searches*

Browser searches are terms or incomplete website addresses entered into the URL bar of the built-in Web browser. These searches are collected by DroidWatch through an alarm. A content observer is not available for this data set; therefore, the underlying data must be retrieved through a scheduled process. Searches are configured to be accessed through the

`com.android.provider.Browser.SEARCHES_URI` content provider URI every six hours by default and are inserted into the DroidWatch database if they are unique events. Access to the content requires the same permission as browser URL collections:

`com.android.browser.permission.READ_HISTORY_BOOKMARKS`.

5.2.8.4 *Calendar Events*

Official support to access calendar events was included with the Android release of Ice Cream Sandwich and API level 14 (Bray, 2011). However, because the phone used in this research runs the older Gingerbread operating system, undocumented parts of the Android framework API are used to retrieve these events and may not work in other Android phones or versions.¹¹ The alarm is configured to scan the `content://com.android.calendar/event_entities` content provider URI for new events every twelve hours by default. Since the content provider does not allow access to any fields that indicate when a calendar event is added to a device, a separate calendar database table is created and maintained. Upon the installation and first execution of DroidWatch, the table is populated with current calendar events and marked with the current timestamp. During periodic scans of the built-in Android calendar, DroidWatch checks each entry against its internal calendar table and adds the event if it is new. While additional data such as location, author, and event details are available through the content provider, this research focuses on event dates and titles. Access to perform these actions on the calendar data set requires the `android.permission.READ_CALENDAR` permission be declared in the `AndroidManifest`.

¹¹ Undocumented content provider URIs were identified through various online Android-related forums and confirmed by their presence in the source code of the Android Open Source Project.

5.2.8.5 *Call Logs*

Call details are retrievable using a variety of methods. A broadcast receiver can be used, but notifications are only triggered when the phone changes its state (e.g., idle, ringing, or “off the hook”). In the rare instance of a missed call appearing without the phone having rung, the event may not send a system broadcast, thus resulting in data loss. To ensure the capture of all new phone calls, a content observer is implemented on the `android.provider.CallLog.Calls.CONTENT_URI` content provider URI. This URI enables the detection of incoming, outgoing, and missed calls, as well as the external phone number, contact name, duration, and start date of the call. Note that this research discovered false-positive issues with this content observer. As users accessed portions of their contacts list (without calling anyone), change notifications were sent; however, because the number of false positives was manageable, checks were implemented to ensure that only new phone call events are inserted into the local DroidWatch database. Access to the logged phone content requires the `android.permission.READ_CONTACTS` permission be added to the `AndroidManifest`.

5.2.8.6 *Contacts Added*

Contact information can be monitored for changes on an Android phone through a content observer; however, there are no indications of specific changes. Furthermore, the dates that contacts are added to a phone are not available through the content provider URI, `android.provider.ContactsContract.CommonDataKinds.Phone.CONTENT_URI`. To detect the addition of a contact, a separate contacts table in the local DroidWatch database is maintained. During the app’s collection process, a check is performed to verify if the DroidWatch’s contacts database has been populated. If not, the phone’s current contact list is copied with the current timestamp inserted as each entry’s “date added.” Following this, any changes noted by the content observer are compared against DroidWatch’s list. If a contact is not

recognized, the name, phone number, and current date/time are inserted in the local database as a new event and new contact in each respective table. The same permission required for monitoring the call log, `android.permission.READ_CONTACTS`, is required for contact monitoring in DroidWatch.

5.2.8.7 *Device Accounts*

Accounts associated with a device are not collected through typical collection components. Instead, this data is gathered as part of the app launching process and assumes that a user has set up the Android device to sync with a personal account (e.g., Gmail or Skype). The account names are retrieved by looping through an array of `Account` objects through the Android API's `AccountManager.get().getAccounts()` method. The only requirement to use this method is to declare the `android.permission.GET_ACCOUNTS` permission in the `AndroidManifest`.

5.2.8.8 *Device ID*

A device ID is a unique identifier that differentiates the Android phone from other phones on an enterprise. In DroidWatch, the device ID is collected at the start of every scheduled transfer process (every two hours by default). In addition, it is included in the data received by the enterprise server. The app uses an alarm that calls the `getDeviceId()` method from the Android API's `TelephonyManager` class to directly access a phone's device ID. According to the Android Developer Documentation, a device ID is one of the following types of identifiers:

- International Mobile Equipment Identity (IMEI)
- Electronic Serial Number (ESN)
- Mobile Equipment Identifier (MEID).

An IMEI is returned for Global System for Mobile Communications (GSM) phones, while an ESN or MEID is returned for Code Division Multiple Access (CDMA) phones (Google, n.d.-f). The device used in this research returns an MEID, since Sprint (who use CDMA) is the provider of the test phone. The `android.permission.READ_PHONE_STATE` permission must be declared in the `AndroidManifest` for an app to extract the device ID.

5.2.8.9 *GPS Location*

The phone's GPS location is retrieved through the `LocationManager` class in the Android API. The class supports a `requestLocationUpdates()` method, which takes a minimum time interval as an argument to stagger the connection requests. Ideally, this is how GPS collections should be performed; however, faults in this method ignore the specified minimum time interval on devices running Gingerbread and Ice Cream Sandwich. In addition, it continuously makes requests for GPS coordinates, quickly draining the battery and degrading the device performance. This issue was allegedly resolved with the release of the Android's Jelly Bean operating system (Barbeau, 2012). To mitigate the issue for this research, the `getLastKnownLocation()` method was used through an alarm scheduled to execute once every hour by default. This method relies on active GPS usage prior to collection and does not activate the location provider, thus saving the battery life of a device. Access to the GPS provider requires that the `android.permission.ACCESS_FINE_LOCATION` permission be declared in the `AndroidManifest`. Other location permissions, such as `android.permissions.ACCESS_COARSE_LOCATION`, are optional and may save additional battery life but result in less accurate location reporting.

5.2.8.10 *Location Provider Settings Changes*

Android phones consist of a GPS location provider and a network location provider. The GPS provider retrieves location updates through satellites, while the network provider gets a current set of coordinates through Wi-Fi access points and cell towers. When the phone's settings are manually adjusted to enable or disable these providers, a system broadcast is generated. In DroidWatch, a broadcast receiver is established with an intent-filter to handle the `LocationManager.PROVIDERS_CHANGED_ACTION` intent. The broadcast does not contain information that indicates which provider changed; therefore, a separate process must query the state of each provider on each change notification. No permissions are required in the `AndroidManifest` to access this information.

5.2.8.11 *Multimedia Message Service*

The detection and extraction of new MMS messages are a challenge for Android apps that attempt to retrieve this information. A system broadcast is delivered for incoming MMS messages, but not for outgoing messages. In addition, unlike the broadcast seen for app installs and removals, no information contained in the incoming MMS broadcasts can be extracted to determine the event content.

Apps that declare broadcast receivers and include intent-filters to handle the `android.provider.Telephony.MMS_RECEIVED` and `android.provider.Telephony.WAP_PUSH` intents can be notified of incoming MMS messages. When capturing outgoing MMS, a content observer initially was registered on the undocumented `content://mms` content provider URI, which resulted in app crashes upon any detected MMS activity. Furthermore, it was discovered that the MMS content observer triggered

during the creation of a new MMS message, not when a message was sent; therefore, an alarm was scheduled to retrieve the history of outgoing MMS messages once every hour.

It is possible to extract the contact's address (phone number or email address) and name, message timestamp, subject, filename of the picture, and whether text was included with the message (not the message text itself). These fields can be accessed through more undocumented parts of the Android SDK, although Google discourages this for stability reasons. Undocumented API methods are likely to change in future official Android releases; if this happens, an app like DroidWatch may crash (Bray, 2011).

As part of this research, undocumented areas of the Android SDK are explored to discover MMS message content for incoming and outgoing messages.¹² The `content://mms` content provider URI contains the message ID, direction, date, conversation thread ID, and subject fields. The `content://mms/part` content provider URI contains the filename of the picture and the text field. If the text field is null, no text is included with the MMS message. If the field is blank, there is associated text; however, no mechanisms to access the text content were found in the Android SDK to work on the test phone. The `content://mms-sms/conversations` URI is used to find a contact's address (phone number or email address) through a conversation thread ID; however, the presence of an address is largely inconsistent. Conversation threads that last received an incoming SMS or MMS message appear to display the recorded address, while the address field is empty. To mitigate this, an additional check of the undocumented `content://sms` content provider URI is performed to obtain the conversation thread's address, which is a more reliable method of retrieving addresses. The contact name can then be

¹² The following works were useful in performing the research and exploration of MMS content:

- <http://stackoverflow.com/questions/3012287/how-to-read-mms-data-in-android/6446831#6446831>
- <https://play.google.com/store/apps/details?id=com.jensdriller.contentproviderhelper&hl=en>

associated with the address through methods previously explained in “Contacts Added” (see Section 5.2.8.6).

Duplicate message checks are performed for incoming and outgoing MMS messages. This ensures that the local database only contains one copy of the message. The permissions required for MMS message retrieval in DroidWatch’s AndroidManifest include

```
android.permission.RECEIVE_MMS, android.permission.READ_CONTACTS,  
and android.permission.RECEIVE_WAP_PUSH.
```

5.2.8.12 *Pictures Added*

Photos taken while using an Android smartphone’s camera result in system broadcast events, but pictures added through alternate methods (e.g., download, manual insertion) do not. To achieve wider image monitoring capabilities, a content observer is used instead of a broadcast receiver.

The content observer is registered on the built-in Android Gallery’s content URI,

```
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI.
```

Resulting change notifications, concurrent with other collections retrieved using content observers, do not include information about changes in the Gallery. When a change notification triggers, additional checks in DroidWatch are performed to see if the value in the content provider’s “date_added” field contains a timestamp that falls within the last five seconds, thus indicating a new picture. Other fields available through the content provider URI are “latitude,” “longitude,” and “mime_type,” but this research only focuses on the name and date of each image detected. New events are also checked against the local database to ensure that they are not duplicates. This collection requires no additional permissions in the AndroidManifest to gain read access to the content.

5.2.8.13 *Screen Lock Status*

System-wide broadcasts are sent when an Android phone's screen status changes (e.g., unlocked, locked, or off states). To capture these broadcasts, a broadcast receiver must be set up with intent-filters that handle the `Intent.ACTION_USER_PRESENT`, `Intent.ACTION_SCREEN_ON`, and `Intent.ACTION_SCREEN_OFF` intents. No additional permissions are required in the Android manifest to access this information.

5.2.8.14 *Short Message Service*

Broadcast receivers are configured to retrieve incoming SMS messages but are not applicable for outgoing messages, which are captured through a content observer registered on the undocumented `content://sms` content provider URI from the Android SDK. The incoming broadcasts contain extra information in their intents, including the sender's address, date, and body. The address can then be associated with a contact name through methods similar to the "Contacts Added" collection. The aforementioned fields are similarly retrieved through the content provider for outgoing messages.

In the case of outgoing SMS, change notifications are generated through the content observer for each step of the transmittal process (i.e., messages moving from "draft" to "pending" to "sent" statuses). DroidWatch is configured to only filter for the sent messages; this reduces the number of duplicates recorded in the local database. The ability to collect the SMS information in DroidWatch requires that the `android.permission.RECEIVE_SMS`, `android.permission.READ_SMS`, and `android.permission.READ_CONTACTS` permissions be declared in the AndroidManifest.

5.2.8.15 *Third-Party App Logs*

Android's built-in application logging system (`android.util.Log`) is available for any Android app to use. The app logs are temporarily stored in a ring buffer in the phone's `/dev/log/main` character device file by default. The ring buffer operates by taking in new logs and pushing out older ones (White, 2012). Logs can be read through a tool available in the Android SDK's named Logcat. Apps that call Logcat at runtime, such as DroidWatch, must have the `android.permission.READ_LOGS` permission declared in the `AndroidManifest` to gain access to this data set. DroidWatch collects Logcat logs through an alarm scheduled to execute once an hour by default.

Unfiltered Logcat output typically results in an overwhelming number of logs; however, steps were taken to reduce this number by identifying and discarding logs generated by built-in Android apps. Known tags of system apps are included as filters in the Logcat command in addition to a minimum log priority level (i.e., debug logs will not be shown). Attempts are also made to identify the name of the system process that generated the log event. The process IDs (PIDs) that appear in the Logcat output are compared to a list of running processes on the device. If a PID is matched, its associated process name is compared against a list of known, built-in system process names (e.g., `system`, `com.DroidWatch`, and others) for possible exclusion from DroidWatch. If the name is not recognized, it is assumed to be a third-party app and thus inserted into DroidWatch as a new event. Upon the completion of the Logcat log analysis, the ring buffer is flushed to avoid duplication issues.

5.3 Analysis and Evaluation

This section describes the results of DroidWatch experimentation and applies its capabilities to situations where they can aid internal investigations. Experiments were scripted based on the data

sets collected and the author's prior work experience. All logged events captured by detectors (see Appendix D for the full list of detectors) and transferred from the phone to the enterprise server were automatically ingested into Splunk,¹³ a well-known log indexing system with a Web interface for searching. Results were displayed as individual events, formatted as key-value pairs. All data was associated with a device ID and could be searched, filtered, and put into timelines for identification and trend analysis. Note that while Splunk was used in this research because of its cost (free for up to 500 megabytes per day), other products can perform similar functions.

Figure 7 illustrates the number of logged events extracted from Splunk over the span of a single day, broken down by data set. The sample contained 442 logs generated by one device with third-party app logs comprising the majority of the records. Log totals per day will differ based on usage habits; a device that is more heavily used will see increases in log totals. No visible impacts to phone usage or battery life were observed throughout the tests.

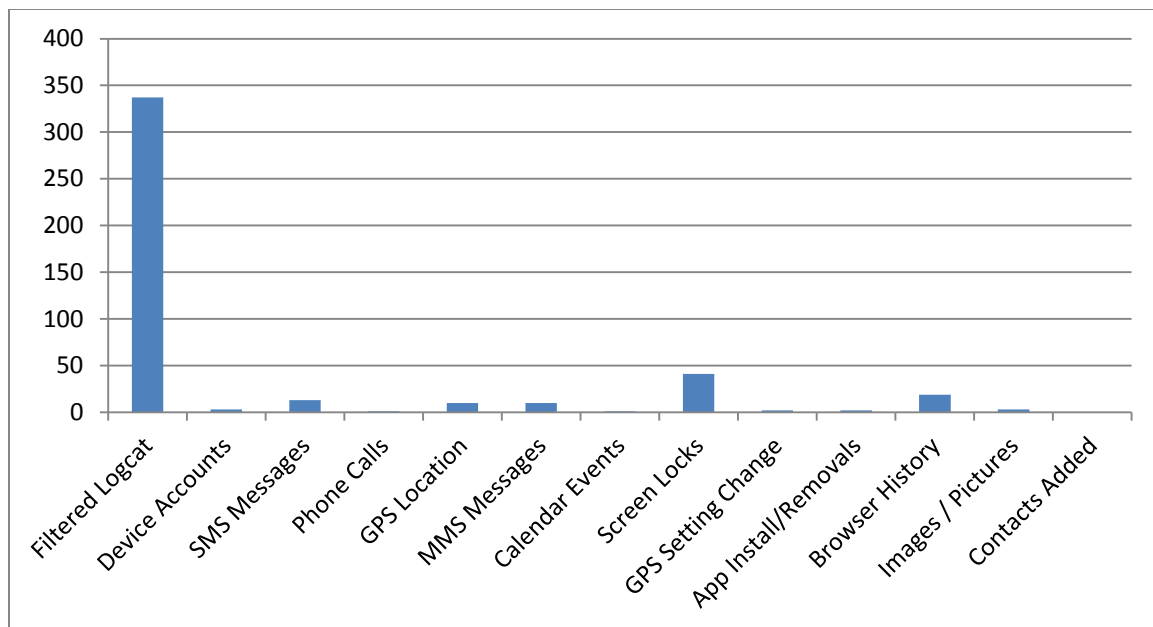


Figure 7. Events Logged Over 24 Hours

¹³ More information about Splunk can be retrieved from <http://www.splunk.com/>.

5.3.1 General Usage Trends

ScreenWatcher, the DroidWatch detector that monitors and records a device’s screen lock status, is useful in determining general phone-use over time. A search for “Screen Unlocked” in Splunk displays a timeline of user-performed actions that indicate active phone usage (e.g., PIN code, gesture, or password entries). The results, which span several days, can be seen in Figure 8. This data can also be used to determine phone activity at a certain time, discover masquerading users (i.e., users other than the originally assigned user), or develop usage patterns for employees.

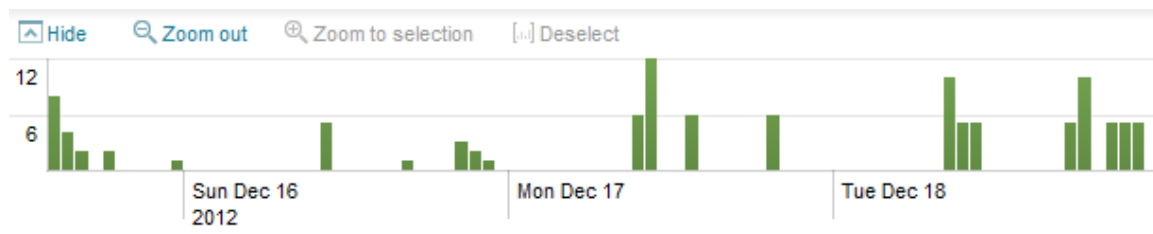


Figure 8. Detected Screen Unlock Actions (Splunk)

5.3.2 Suspicious Contacts and Communications

DroidWatch implements a detector called ContactWatcher that monitors a phone's address book for newly added contacts. Contact names or phone numbers that pose a risk to an organization may be discovered through searches or incorporated into Splunk triggers (available in the Splunk Enterprise edition), which notify investigators of a previously specified activity or activities.

CallWatcher is the DroidWatch detector that identifies incoming and outgoing phone calls on a device. Investigators can perform searches and filters on data recorded in Splunk to discover calls with suspicious numbers or contact names. These may be numbers external to an organization's phonebook or previously identified known-bad individuals.

SMS and MMS detectors are also implemented in DroidWatch for incoming and outgoing message communications and may be beneficial for investigators. The recorded data can be

searched for suspicious activity, similar to phone call data; however, SMS message body content is available for mining information that may provide insight about an investigated event. An incoming SMS in DroidWatch can serve as a third-party time source since the timestamp is carved out of the protocol description unit (PDU) of the raw SMS content. The embedded time within the SMS structure does not rely on the phone's clock (Casey, 2009). Some issues encountered during the analysis of this data were:

- Messages sent to multiple contacts only listed a single recipient in the logs.
- Timestamps carved out of incoming SMS messages did not contain time zone information. The time zone appeared to be that of the sender (not Coordinated Universal Time [UTC]), which may cause problems if a sender's time zone is unknown.
- There was no MMS message text, although a reviewer could see if text was originally included.

As an example of the DroidWatch detectors described above, a Splunk search was performed to find evidence of phone calls, SMS messages, and added contacts. Results are seen in Figure 9.

```
id=86460 _id=1281 detector=ContactWatcher action="Contact Added" date_occurred=1356205083.000 description="Test Thesis"
additional_info="ID:9722; Number:2223334444;" device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail/PhoneWatcher_Events/PhoneWatcher

id=421223 _id=2937 detector=SentSMSWatcher action="SMS Sent" date_occurred=1356280285.000 description="9pm at double o in
wappingers of you can make it!!" additional_info="MSG_ID:4729; ReceiverAddress:845 [REDACTED]; ReceiverContact:Nathan [REDACTED];"
device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail/PhoneWatcher_Events/PhoneWatcher

id=421533 _id=3247 detector=CallWatcher action="Phone Call" date_occurred=1356285115.000 description="Outgoing
additional_info="ID:11981; Number:845 [REDACTED]; Name:Michael [REDACTED]; Duration:33 NumType:1;" device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail/PhoneWatcher_Events/PhoneWatcher
```

Figure 9. Communication Related Search Results (Splunk)

Splunk Enterprise triggers can be configured to notify investigators when a series of events occur. For example, if a picture is taken using a phone's camera and immediately sent over MMS, the activity may warrant further investigation depending on the circumstances. Figure 10 shows a search revealing the logs of a picture taken on Saturday, December 22, 2012, at 3:20 p.m. that was subsequently attached to an outgoing MMS message. If it is found (possibly through GPS

tracking and other mechanisms) that the user was alone in the office, more analysis may be required to determine whether a data leakage occurred.

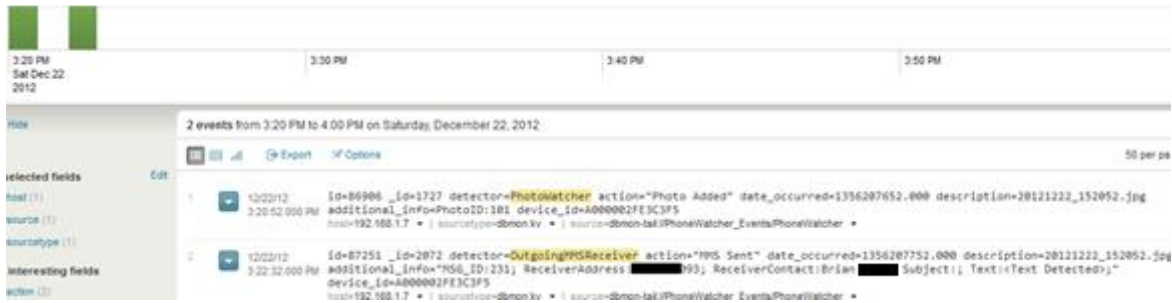


Figure 10. Photo and MMS Search Results (Splunk)

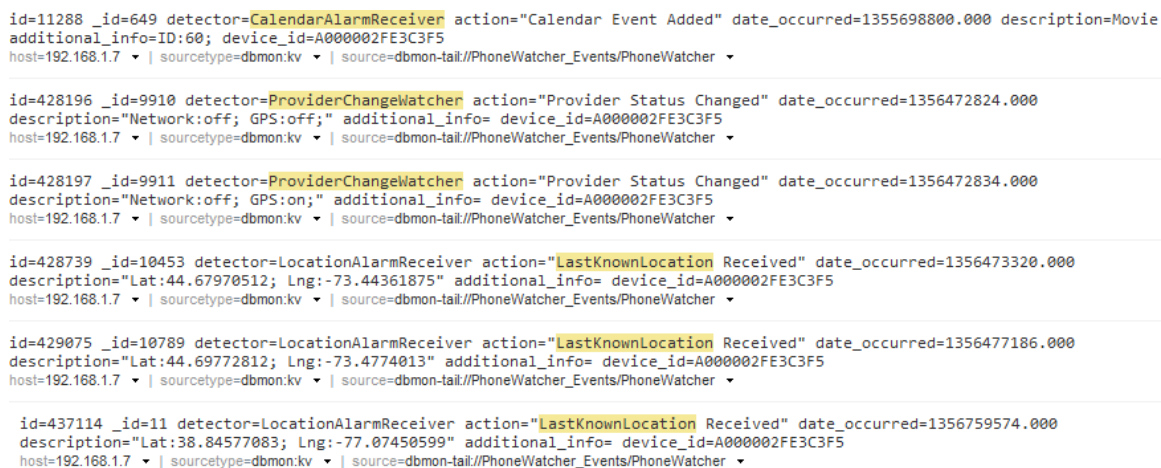
5.3.3 Location Monitoring

Last known locations recorded by DroidWatch include the device ID, latitude, longitude, and capture time. The approach that DroidWatch uses to collect locations conserves battery life, but results in the sparse logging of recorded locations. While the detector is configured to collect the last known coordinates every hour by default, only four locations over a seven-day period were reported. Even though the GPS provider setting is enabled, the last known locations are not stored on a device unless the provider is actively used (i.e., the Google Maps app is opened to display the current position). Furthermore, a phone's last known location is cleared upon device reboots, causing a stored set of coordinates to be lost before being recorded.

Modifications made to a phone's location provider settings are available; however, the logs contain all the statuses (e.g., enabled/disabled) for the network and GPS providers in every event recorded by this detector. Future work may include implementation of a third-party mechanism (e.g., another DroidWatch database table) to keep the state of each provider. This data is potentially useful if the phone's last known location becomes more reliable. It will allow an analyst to locate a device when its GPS setting is turned off manually.

Scheduled calendar events are also beneficial to investigators. Searches can be performed on a user's calendar appointments (fields include event titles and dates), which may assist with activities such as alibi checking, surveillance planning, or providing knowledge of a user's travel plans.

Search results for evidence of a device's last known location, location provider setting changes, and calendar appointments are provided in Figure 11.



The screenshot displays a list of search results in Splunk. Each result line contains the following fields: id, _id, detector, action, date_occurred, description, additional_info, and host. The results are as follows:

id	_id	detector	action	date_occurred	description	additional_info	host
11288	649	CalendarAlarmReceiver	Calendar Event Added	1355698800.000	Movie	device_id=A000002FE3C3F5	192.168.1.7
428196	9910	ProviderChangeWatcher	Provider Status Changed	1356472824.000	Network:off; GPS:off;	device_id=A000002FE3C3F5	192.168.1.7
428197	9911	ProviderChangeWatcher	Provider Status Changed	1356472834.000	Network:off; GPS:on;	device_id=A000002FE3C3F5	192.168.1.7
428739	10453	LocationAlarmReceiver	LastKnownLocation Received	1356473320.000	Lat:44.67970512; Lng:-73.44361875	device_id=A000002FE3C3F5	192.168.1.7
429075	10789	LocationAlarmReceiver	LastKnownLocation Received	1356477186.000	Lat:44.69772812; Lng:-73.4774013	device_id=A000002FE3C3F5	192.168.1.7
437114	11	LocationAlarmReceiver	LastKnownLocation Received	1356759574.000	Lat:38.84577083; Lng:-77.07450599	device_id=A000002FE3C3F5	192.168.1.7

Figure 11. Location Related Search Results (Splunk)

5.3.4 Internet History

DroidWatch collects and makes available the events performed within the built-in Android Web browser. An Internet history event, found in Splunk, includes the action taken (e.g., browse or search), the search term or URL, the event time, and the attributed device ID. This information can be useful to investigators looking for suspicious browser usage on an enterprise, such as uploads of intellectual property to external websites. Search engine searches may also be parsed and utilized to better extrapolate a user's possible intentions behind the detected actions.

Figure 12 shows the results of a Splunk search for “steganography.” The user performed a browser search for “steganography research” and then navigated to two related websites. The search text provides the context that the user may be performing research on the topic.

```
id=437512 _id=22 detector=BrowserWatcher action="Browser Search" date_occurred=1356994814.000 description="performing
steganography research" additional_info= device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher

id=437509 _id=19 detector=BrowserWatcher action="Browser Navigation" date_occurred=1356994815.000
description=http://www.google.com/search?hl=en&redir_esc=&source=android-browser-type&v=133247963&qsubts=135699481464;
q=performing+steganography+research&v=133247963 additional_info=http://www.google.com/search?hl=en&redir_esc=&source=i
browser-type&v=133247963&qsubts=1356994814642&q=performing+steganography+research&v=133247963 device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher

id=437511 _id=21 detector=BrowserWatcher action="Browser Navigation" date_occurred=1356994824.000
description=http://en.m.wikipedia.org/wiki/Steganography additional_info=http://en.m.wikipedia.org/wiki/Steganography
device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher
```

Figure 12. Browser History Results (Splunk)

5.3.5 Malicious Applications

Apps that are installed or removed on mobile enterprise devices are of potential interest to security analysts and investigators. During an internal investigation, it may be discovered that a device has malware or other apps of concern. Alternatively, organizations may have mobile app install policies in place and then subsequently find an installed app that violates the policy. Either case warrants additional concerns and security measures. All app installation and removal data collected by DroidWatch is available for search by investigators. Provided fields include the app’s name, action taken, and install/removal date.

Third-party app logs are also collected by DroidWatch. Several filtering mechanisms attempt to limit logs to only those generated by apps not built-in to the phone; however, results show that these filters are not completely successful. When DroidWatch was initially implemented, averages of 9,826 logs per device per day were generated by the Logcat detector. The initial setup filtered any log generated by the “system” process and other known built-in apps. Later improvements to DroidWatch included filters for app logs generated from the `com.android`, `com.google`, and `com.sec.android` processes, which resulted in a more manageable

average of 337 Logcat logs per device per day. Organizations can do more fine-tuning to tailor filters to their specific environments.

Figure 13 shows the results of a Splunk search for app installs and third-party app logs. An app named StegDroid was found to have been installed, warranting additional attention due to its data hiding abilities. Also listed is the third-party app “com.locationlabs.v3client,” which produces an event in the system logs that appears to indicate that the app is listening for SMS messages.

```
id=437553 _id=60 detector=LogcatWatcher action=Logcat date_occurred=1357005691.000 description=com.locationlabs.v3client
additional_info="12-31 21:01:31.545 I/V3Client(27157): SMS ignored" device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail://PhoneWatcher_Events/PhoneWatcher

id=437527 _id=34 detector=AppWatcher action="Package Installed" date_occurred=1357004793.000
description=uk.ac.cam.tfmw2.stegdroid additional_info= device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon:kv | source=dbmon-tail://PhoneWatcher_Events/PhoneWatcher
```

Figure 13. App Related Search Results (Splunk)

5.3.6 Attribution

Having a device report its device ID and synced account information may be useful to investigators for several reasons:

1. Helps to verify and identify the user (potentially the owner) of the phone
2. May provide clues of personal account handles and aliases that are not previously known
3. Provides indications of how a phone is used
4. Ties together logged events with a specific device

The device ID (an MEID in this research) can be seen in many of the previously provided screenshots. Every logged event is associated with a device. Synced accounts are configured to report less often—every time the DroidWatch service starts. Figure 14 displays the results of a search for a device’s accounts, which, in this case, includes an email address, a Skype username, and a Facebook username.

```

id=437493 _id=3 detector=DeviceAccountRetriever action="Device Account Detected" date_occurred=1357004494.000
description=justin.grover@gmail.com additional_info=com.google device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon.kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher

id=437492 _id=2 detector=DeviceAccountRetriever action="Device Account Detected" date_occurred=1357004494.000
description=justin.grover@gmail.com additional_info=com.facebook.auth.login device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon.kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher

id=437491 _id=1 detector=DeviceAccountRetriever action="Device Account Detected" date_occurred=1357004494.000 description=jgro-
skype5 additional_info=com.skype.contacts.sync device_id=A000002FE3C3F5
host=192.168.1.7 | sourcetype=dbmon.kv | source=dbmon-tail//PhoneWatcher_Events/PhoneWatcher

```

Figure 14. Device Account Search Results (Splunk)

5.4 Anti-Forensics

Research presented at the 2006 DFRWS conference defines anti-forensics as “any attempt to compromise the availability or usefulness of evidence in the forensic process” (Harris, 2006). The upcoming sections are anti-forensics categories drawn from previous work conducted in the Android anti-forensics field and are used to assess the DroidWatch app for anti-forensic vulnerabilities. The categories are destroying evidence (Section 5.4.1), hiding evidence (Section 5.4.2), altering evidence sources (Section 5.4.3), counterfeiting evidence (Section 5.4.4), and detecting forensics tools (Section 5.4.5) (Azadegan, Yu, Liu, Sistani, & Acharya, 2012; Distefano, Me, & Pace, 2010).

Note that DroidWatch, in its current form, is fully susceptible to root attacks, app uninstalls, and process terminations. External protections offered by MDMs, such as root detection and the enforcement of app installation policies, are relied upon to ensure data integrity within DroidWatch. Future work to detect terminations of the DroidWatch process is described in Section 6.3.

5.4.1 Destroying Evidence

On a phone without root privileges, data sets collected through broadcast receivers and content observers are not likely susceptible to evidence destruction methods, since a copy of each event is

recorded in DroidWatch's private directory as the event happens. Normal users, by default, do not have the necessary permissions to directly access content in this location. Data sets retrieved through alarms, however, are susceptible to destruction tactics, since it may be possible to remove evidence (e.g., outgoing MMS messages, third-party app logs, calendar events, browser navigations, browser searches, and last known GPS locations) before the next scheduled collection. Note that it is unlikely that a data set's entire history will be lost, unless there is a continuously running deletion mechanism that removes all events shortly after they occur.

One concern for DroidWatch is that other apps can register for custom intent-filter priorities. An app that uses the maximum intent-filter priority value, 2,147,483,647 ($2^{31}-1$), has the ability to intercept and drop a broadcast before another app receives it. DroidWatch uses the default intent-filter priority value, due to its status as a research prototype, and relies upon external enforcement of an app installation policy for protection. Further work on this issue (see Section 6.3) is proposed for future researchers to pursue.

5.4.2 Hiding Evidence

AFDroid, the anti-forensics tool developed by Distefano et al., can hide and restore data within AFDroid's private folder to circumvent forensic processes. The data sets collected through scheduled alarms (the same as those mentioned in Section 5.4.1) are among those affected by a tool like AFDroid. A user wanting to hide a few recent outgoing SMS messages could use manual transfer methods or AFDroid to move the messages; however, upon message restores, DroidWatch would detect these as new events and log them.

Larger amounts of a data set's history could be hidden if an automated, continuous hiding process was developed and implemented onto a device. Without the enforcement of strong app

installation policies for smartphones deployed with DroidWatch on an enterprise, all DroidWatch's data collection components are susceptible to this type of subversion. The aforementioned ability for apps to register custom intent-filter priorities can enable apps to hide data through the interception and rerouting of broadcasts. For example, GoSMS,¹⁴ a legitimate third-party SMS app, registers for the maximum possible intent-filter priority to eliminate the duplication of messages (Kovacevic, 2011). Like GoSMS, malicious apps may also register for high priorities on a phone and cause broadcasts to be rerouted. As mentioned previously, DroidWatch uses the default intent-filter priority, although future work (see Section 6.3) is proposed to mitigate this issue.

5.4.3 Altering Evidence Sources

Altering evidence sources, according to Distefano et al., involves modifying a data set to thwart a collection process. This anti-forensics category is another area of concern for DroidWatch. Data collected by alarms (the specific data sets listed in in Section 5.4.1) are susceptible because the utilized API methods rely on certain values in a data set to be present. For example, when looking for outgoing messages through the MMS content provider, the “msg_box” column indicates the message's direction—with “2” representing a sent/outgoing MMS message. If this value is altered to “5,” the data collection component ignores the message. Such an anti-forensics technique may affect a data set's entire history if implemented into a continuously running, automated system; otherwise, only individual events may be affected, due to the persistent nature of DroidWatch.

Another relevant, anti-forensics concern for DroidWatch is the ability for malicious users to reinstall a modified version of DroidWatch. Assuming that one has an app's Android Package

¹⁴ More information about the GoSMS app can be retrieved from <https://play.google.com/store/apps/details?id=com.jb.gosms>.

file, previous work has shown that code for an app like DroidWatch can be decompiled and modified (Pan, 2012). The malicious user could strip the AndroidManifest of its permission request statements and reinstall the app on a smartphone.¹⁵ If successful, this process would cause targeted data set collections to fail on the Android device.

To help combat the altering of evidence sources, strong app policies must be enforced to prevent the installation of apps that may cause security lapses on an enterprise. Tamper-resistance mechanisms can also be incorporated into DroidWatch and are proposed as future work (see Section 6.3).

5.4.4 Counterfeiting Evidence

Counterfeiting evidence on mobile devices involves adding fictitious data to existing data sets to confuse or evade investigators. All of DroidWatch's collection components are vulnerable to this, because no checks are performed on newly detected events to differentiate fake entries from real ones. An automated counterfeiting anti-forensics mechanism that continuously adds fake data over time may cause some confusion; however, depending on the complexity of the fake data, it may be relatively easy to spot and filter the real events on the server.

Adding massive amounts of fictitious data over a short time span may also result in a denial of service attack, another concern for DroidWatch. Should enough data be injected into a phone's internal memory, the app (and other apps) may cease to function properly. Future work is proposed (see Section 6.3) to implement a "keep-alive" mechanism that would indicate disruptions in the DroidWatch service.

¹⁵ During the code repackaging process of a previously decompiled app, a malicious user would be required to re-sign the app. This would most likely be with a key other than that used by the original author of the app.

5.4.5 Detecting Forensics Tools

Considering the work done by Azadegan et al., which involved the detection of forensic processes on Android smartphones, it was determined that their research was not directly applicable to DroidWatch. They focused on listening for initial connection signatures of some well-known forensics tools on Android phones; if found, one of the anti-forensics methods discussed in Sections 5.4.1 through 5.4.4 was triggered. DroidWatch performs monitoring, unlike traditional forensics tools, and has no initial connection signatures or requirement for physical access to a device.

The ideas presented by Azadegan et al., however, can be extended to detect DroidWatch's scheduled collection and transfer process. For example, if DroidWatch's pattern or "signature" of transfers to the enterprise server is determined, an automated tool could disable networking on a device just before the transfer occurs, therefore causing a denial of service on the system. Future work to DroidWatch may mitigate this through the randomization of the collection intervals, possibly through event-based triggers (i.e., data set "A" is collected when "B" happens).

6 Future Work

Future research on DroidWatch includes work on the app and the enterprise server. The upcoming sections will cover proposed improvements for future researchers to investigate and cover resolving current issues (Section 6.1), collecting additional data sets (Section 6.2), and implementing anti-tampering mechanisms (Section 6.3). Ideas for longer-term research are also proposed for potential integration (Section 6.4).

6.1 Resolve Current Issues

DroidWatch warrants future work for several data sets currently used. The duplication of reported events by a phone is an issue; this may be resolved with improved server handling of events.

Additionally, Logcat output, even with filters in place, continues to generate the most events of all the collections. A comprehensive study on the usefulness of Logcat's reported logs to internal investigations could prove beneficial for the Android forensics community. Improvements to DroidWatch's current thread usage, location retrieval methods, and configuration settings can also be developed.

6.2 Additional Data Collections

Android forensics is an evolving field that changes with each new operating system release from Google. As new data sets and features become available, their value to an investigation must be assessed for possible inclusion into a monitoring system. Future work includes the following additional data sets to be monitored and collected by DroidWatch:

- USB Debugging Settings
- Voicemail Logs
- Phone Restarts
- Logs of `dumpsys`, `dumpstate`, and `dmesg`.

A user attempting to subvert security apps or policies may enable USB debugging on a phone to obtain command line access from a laptop or desktop computer, thus providing additional options to explore and possibly exploit a device. Catching the generated broadcast as the setting is enabled or disabled can be beneficial for the overall security of an enterprise, including internal investigations. Detecting phone restarts may also be useful for the overall security and awareness of a phone's status on an enterprise. Investigators may use this in determining usage trends.

Voicemail log detection can also be a helpful element to internal investigations. Knowledge of a voicemail following a missed call increases the chance that a number was called intentionally. Programmatic access to the voicemail logs was added with Ice Cream Sandwich (API 14).

Third-party app and system logs include those of the `dumpsys`, `dumpstate`, and `dmesg` commands, which may provide additional clues to how a phone is used. Filtering mechanisms similar to those used in Logcat will be required to lessen the many logs outputted by each command and improve the usefulness to investigators.

6.3 Anti-Tampering Mechanisms

Data collected and stored by DroidWatch on a phone currently relies on the Android built-in security model (Section 3.3) to prevent users and apps from malicious viewing or tampering.

Some proposed capabilities to harden DroidWatch include:

- Encryption of database events
- High intent-filter priorities
- Code obfuscation
- Event-based collections and transfers
- Keep-alive logging
- Database hashing.

Encrypting events in the local phone database is a mechanism to deter users from viewing or tampering with previously collected events. Each event will be paired with a checksum and encrypted using a public key infrastructure with the private key stored on the enterprise server. Registering for maximum intent-filter priority values in the AndroidManifest may help prevent other apps from suppressing a system broadcast, although further research is needed to determine what happens when two apps register with the same priority value (PVS, 2012). Obfuscating the DroidWatch source code (when integrated into other systems) may act as a deterrent because it can make decompiled code difficult to read and understand (Mackenzie, 2012). Event-based

triggers would provide a more random collection and transfer pattern and may prevent time-based thwarting attempts that are possible against scheduled operations; however, further research on the effectiveness of this capability is required. Periodic logging of “keep-alive” messages to the DroidWatch phone database would show disruptions in the DroidWatch service. Indications of tampering would be evident if it were found that a large amount of time elapsed between logged keep-alive messages. The final, future anti-tampering capability listed above is the hashing of transferred databases. Comparing hash values is a solid way to discover digital evidence tampering, assuming that the originating device is not compromised.

6.4 Future Considerations

The following list contains longer-term considerations for DroidWatch that require additional research or system redesigns:

- GPS proximity alerting
- Third-party admin APIs
- Remote screen watching
- Additional device identifiers
- Password change detection
- Remote imaging of SDCard
- Detecting sizes of read-only partitions
- Integration into a law enforcement tool
- Integration into an MDM system

GPS proximity alerts may be configured on Android phones. These will trigger if a user’s phone reports its location to be near a pre-programmed set of coordinates. Support for this feature may require system architecture changes to DroidWatch, as communications currently travel in one direction: phone to server. A revised system should support server-side options relayed to mobile devices in order to target individual devices on an enterprise.

Proprietary third-party APIs created by manufacturers, such as the Samsung for Enterprise (SAFE) API¹⁶ for Samsung phones, may be worth researching to identify additional monitoring capabilities. The AirWatch MDM currently implements the Samsung API to gain more capabilities for managing devices.¹⁷

Other data set collections for future consideration are remote screen watching (mentioned in Section 4.2), additional device identifiers (e.g., model, firmware version, Internet Protocol address, and connected Bluetooth devices), and password changes. Some may require use of the built-in Android Admin API.

DroidWatch currently monitors device activity stemming from the `/data` partition of a phone; however, it may be possible to implement mechanisms that reach out to other partitions.

Detecting size changes to the free space within the read-only `/system` partition or imaging the `/sdcard` partition remotely may be helpful for investigators.

Future DroidWatch considerations include the possibility of extracting some techniques used in this app and applying them to a tool used by law enforcement to perform stealth monitoring of a subject's device. A field survey of available tools will need to be conducted and techniques to mask the presence of the monitoring service will need to be researched and implemented.

Integration of DroidWatch into an MDM solution would be very valuable for the Android security communities; however, it would involve a large integration effort as most current MDM systems lack user monitoring features that could aid internal investigations. An overall enterprise security system that provides a combination of solid policy enforcement, remote device

¹⁶ More information about SAFE can be retrieved from <http://www.samsung.com/us/safe>.

¹⁷ More information about AirWatch's use of SAFE can be retrieved from <http://www.air-watch.com/solutions/android/samsung-for-enterprise>.

management, and comprehensive user monitoring on Android devices would help close security gap concerns among government and industry organizations considering Android smartphone deployments.

7 Conclusion

The system introduced in this thesis serves as a prototype for the Android community to design and implement Android smartphone monitoring in enterprise environments without root privileges. The DroidWatch app is the first open-source solution of its kind; however, it requires further research and improvements to broaden its number of monitored data sets and capabilities. In addition, anti-tampering mechanisms will need to be implemented to increase security. As noted, the data collected by DroidWatch would be useful for many types of internal investigations for a variety of reasons. This research also provides a novel development design strategy, which can be used as a guide for developers needing to prioritize Android app components for monitoring. Finally, this thesis serves as a comprehensive list of data sets available through the default Android API for researchers and investigators to reference.

Appendix A: Android Manifest

The following code block displays the contents of the DroidWatch `AndroidManifest.xml` file. Within it, the app's declared permissions and registered activities, services, content providers, and broadcast receivers can be seen. With the exception of the `StartupIntentReceiver` broadcast receiver, which declares an intent-filter to handle the `android.intent.action.BOOT_COMPLETED` broadcast, all intent-filters associated with broadcast receivers are declared directly within the code base (not in the `AndroidManifest`).

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.droidwatch"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="10" />

    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_WAP_PUSH" />
    <uses-permission android:name="android.permission.RECEIVE_MMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS" />
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.READ_LOGS" />
    <uses-permission android:name="android.permission.READ_CALENDAR" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"
        android:allowBackup="false" >
        <activity
            android:name=".DroidWatch"
            android:label="@string/title_activity_droidwatch" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".ConsentBanner"
            android:theme="@android:style/Theme.Translucent"
            android:launchMode="singleInstance" >
        </activity>

        <service
            android:name=".WatcherService"
            android:exported="false" >
            <intent-filter>
                <action android:name="com.droidwatch.WatcherService" />
            </intent-filter>
        </service>
    </application>
</manifest>
```

```

    </intent-filter>
</service>

<receiver android:name=".StartupIntentReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
        <category android:name="android.intent.category.HOME" />
    </intent-filter>
</receiver>
<receiver android:name=".CalendarWatcher"/>
<receiver android:name=".LocationWatcher"/>
<receiver android:name=".BrowserHistoryWatcher"/>
<receiver android:name=".LogcatWatcher"/>
<receiver android:name=".MMSOutgoingWatcher"/>
<receiver android:name=".AppWatcher"/>
<receiver android:name=".SMSIncomingWatcher"/>
<receiver android:name=".MMSIncomingWatcher"/>
<receiver android:name=".LocationProviderWatcher"/>
<receiver android:name=".Transfer"/>

<provider
    android:authorities="com.droidwatch.DroidWatchProvider"
        android:multiprocess="true"
        android:name=".DroidWatchProvider"
        android:exported="false">
</provider>

</application>
</manifest>

```

Appendix B: Juniper Product Evaluation

Evaluation Description

This appendix details the results an evaluation of Juniper's Junos Pulse Mobile Security Gateway (MSG) v3.0.3-45 performed by the author during a 30-day free trial in March 2012. MSG is part of Juniper's mobile device management (MDM) system named Junos Pulse Mobile Security Suite (MSS) that performs remote management and some monitoring of Android, iPhone Operating System (iOS), and RIM devices.

Purpose and Scope

While researching various leading MDM products, Juniper's MDM documentation listed the most user monitoring capabilities when compared to its competitors. This assessment was required to ensure that the work proposed in the thesis was original. With an understanding of the current state of the MDM market in respect to user monitoring, it was possible to identify gaps to focus on. This evaluation assesses the status of user monitoring capabilities within the MSS and tests features available in the MSG. Analysis and results of each test are provided.

This work will not provide recommendations. It is meant to present the technology's capabilities in its current state. Note that not all features of the MSG were covered—only those relevant to user monitoring.

Product Information

The Junos Pulse Mobile Security Suite is a commercial product made by Juniper Networks that includes two components:

1. *Junos Pulse Mobile Security Gateway*¹⁸

The MSG is the central server and repository to which data sent. It has a rudimentary, Web-based console that allows basic admin controls of mobile device policy settings and viewing of some collected data. Access to the MSG was provided by Juniper as part of a free-trial evaluation.

2. *Junos Pulse MSS Dashboard*¹⁹

The MSS Dashboard was not available for this evaluation as Juniper does not include it in free-trials; however, documentation for it is available on Juniper's website. The dashboard displays data reported to the MSG in a user-friendly format. It also includes the ability to view all reported data sets.

As of March 2012, the cost for deploying Juniper MSS agents on up to 5,000 mobile devices is \$291,479 per year. MSS is offered by Juniper as a Software as a Service (SaaS) product. It is not available for private network hosting. Packages and prices differ, based on the number of reporting devices.

Testing Environment

- Samsung Galaxy S II Epic 4G Touch smartphone (Gingerbread v2.3.6)
- Server hosted by Juniper (<https://mss.junospulse.juniper.net/smobile/console/login.htm>).

¹⁸ More information about the MSG can be retrieved from <http://www.juniper.net/techpubs/software/pulse/guides/j-pulse-mobile-3.0R1-security-admin.pdf>.

¹⁹ More information about the MSS Dashboard can be retrieved from <http://www.juniper.net/techpubs/software/pulse/guides/j-pulse-mobile-3.0R2-dashboard-ug.pdf>.

MSG Configuration Settings

The screenshot shows a configuration window titled "Default Monitor & Control Settings". It contains several settings, each with a label and a control element (checkbox or dropdown menu). The settings are as follows:

Setting	Value
Active:	<input checked="" type="checkbox"/>
Log Event Limit:	1
Log Size Limit:	1K
Log Email:	<input checked="" type="checkbox"/>
Log SMS:	<input checked="" type="checkbox"/>
Log MMS:	<input checked="" type="checkbox"/>
Log Voice:	<input checked="" type="checkbox"/>
Disable Voice:	<input type="checkbox"/>
Log Images:	<input checked="" type="checkbox"/>
Log Web Images:	<input type="checkbox"/>
GPS Update Period:	Update every 30 minutes

Figure 15. Mobile Security Gateway Settings

Events including SMS, call logs, and images are configured to be sent to the MSG via SSL after every log is recorded. Transmissions could be configured to update less frequently in real enterprise settings to decrease network traffic but were left “as is” for testing purposes.

According to the product’s administration guide, the following data sets are not supported for monitoring on Android devices:

- Email
- MMS messages
- Web images.

This product also does not support the collection of a phone’s browser history or searches, among other data sets, on Android devices. These features are not present in the MSG and are not

mentioned in the MSS Dashboard documentation. Call logs, referenced by the “Log Voice” option in the settings above, can only be accessed through the MSS Dashboard, which was not available for testing in this evaluation (since Juniper does not include the MSS Dashboard in free-trials). GPS coordinates are set to be retrieved from devices every 30 minutes.

MSS Monitoring Capabilities

The available MSS user monitoring capabilities, as tested and researched during the evaluation, are listed in Table 2. The table gives researchers an idea of features available in the MSS system compared with those available in DroidWatch. Each data set is highlighted in a different color, with the data sets available within MSS listed first. Note that “No Information Available” means that no details were found about the presence of a feature in the MSS. “Unable to test” means that a feature is listed in the MSS Dashboard documentation, but could not be tested as part of the free trial.

Table 2. Junos Pulse Mobile Security Suite - Capability Chart

Data Set	Test	Result		Note
		Success	Fail	
SMS	SMS Collected Regularly	✓		
	View Deleted SMS	✓		
	Export SMS		✓	
	SMS Storage Limits			No information available
	View SMS Contact Name	✓		
	View SMS Contact Number		✓	No number if name is present
	View SMS Direction		✓	
	View SMS Body	✓		
	View SMS Timestamp	✓		
Call Log	Call Logs Collected Regularly	?		Unable to test
	View Deleted Calls	?		Unable to test
	Export Call Logs			No information available
	Call Log Storage Limits			No information available
	View Call Timestamp	?		Unable to test
	View Call Contact Number	?		Unable to test
	View Call Contact Name			No information available
	View Call Duration			No information available
	View Call Direction			No information available
GPS	Location Collected Regularly	✓		Most locations report as null
	Export GPS Coordinates	✓		
	View GPS Storage Limits			No information available
	View GPS Latitude / Longitude	✓		
	View GPS Timestamp	✓		
Gallery	Images/Pics Collected Regularly	✓		
	View Deleted Images/Pics	✓		

	Export Images/Pics		✓	
	Image/Pic Storage Limits			No information available
	View Image/Pic Filename		✓	
	View Image/Pic Timestamp		✓	
App Installs	Apps Collected Regularly	✓		
	View Deleted Apps		✓	No indication of app removals; current app list displayed
	View Added Apps		✓	No indication of added apps; current app list displayed
	Export Installed Apps		✓	
	App List Storage Limits			No information available
	View App Install/Removal Timestamp		✓	
	View App Package Name	✓		
Contacts	Contacts Collected Regularly	✓		
	View Deleted Contacts		✓	No indication of deleted contacts; current list displayed
	View Added Contacts		✓	No indication of added contacts; current list displayed
	Export Contacts		✓	
	Contacts Storage Limit			No information available
	View Contact Name	✓		
	View Contact Number	✓		
	View Contact Number Type (e.g., Home, Work, Cell)		✓	
Device ID	View Contact Date Added		✓	
	Device ID Collected Regularly	✓		
	View Device ID Collection Timestamp	✓		
Calendar	Calendar Collected		✓	
MMS	MMS Collected		✓	
Browser Searches	Browser Searches Collected		✓	
Browser History	Browser Websites Collected		✓	
Screen Locks	Screen Lock Statuses Collected		✓	
GPS Setting	GPS Settings Collected		✓	
Device Accounts	Device Accounts Collected		✓	
Logcat	Third-Party App Logs Collected		✓	

Capability Analysis

SMS Capture

Every SMS message sent and received on the phone appears to be recorded on the server. As of the last day of the evaluation, the message counter within the MSG's "Messages—View Detail" screen lists 832 messages captured; however, due to a bug in the interface, only the latest 20 messages could be viewed.

SMS messages are pushed to the server as they are received on the phone, as specified in the settings. The interface lists a timestamp, a contact name/number, a subject, and a body for each

SMS message. A contact's name replaces the number if matching contact information is available. The subject field appears to mimic the body field for each SMS message. There also appears to be no ability to export content from the MSG interface. Figure 16 is a snapshot of the interface listing SMS messages.

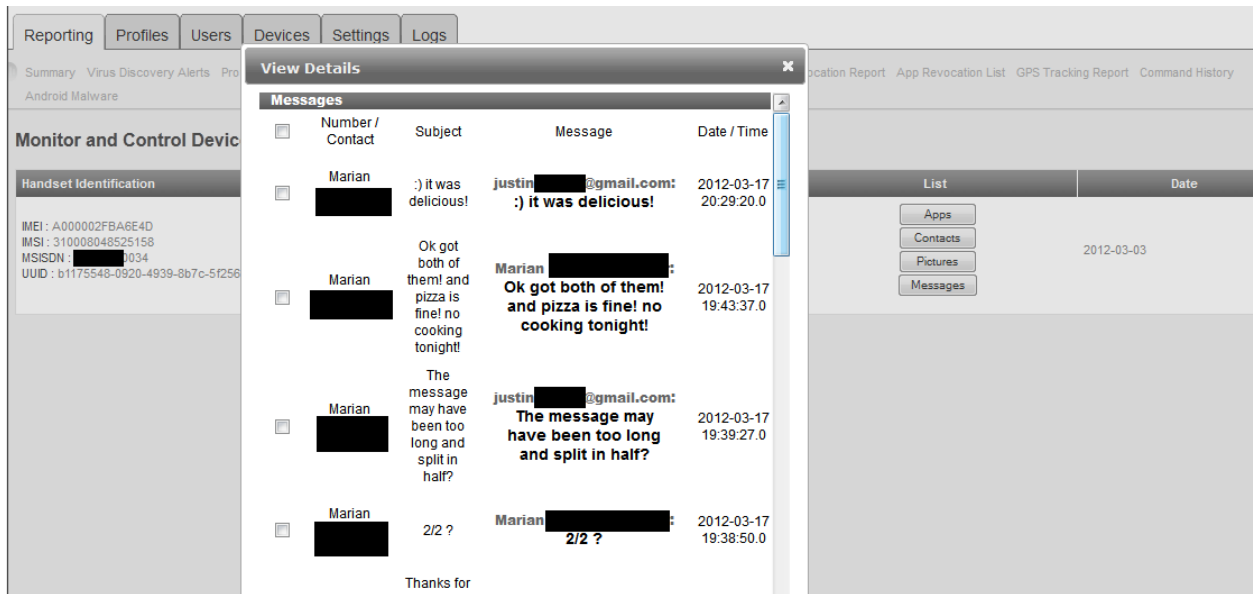


Figure 16. MSG SMS Interface

Call Log Capture

According to product documentation, call logs are only available for viewing from the MSS Dashboard. Since the dashboard interface was not available to be tested for this evaluation, all tests related to this category were marked “Unable to test.” Documentation notes that call timestamps and contact numbers are available for viewing.

GPS Capture

Between March 3, 2012 (start of the evaluation) and March 9, 2012, GPS coordinates were reported every 30 minutes, as specified in the settings. If the phone's GPS provider setting was switched off, the latitude and longitude were reported as "0.0." After March 9, 2012, the phone failed to report its GPS coordinates to the server for unknown reasons.

During this period, 264 GPS submissions were found, all of which could be exported to a Microsoft Excel spreadsheet or a tab-separated format file from the MSG. The "GPS Tracking Report" interface, as seen in Figure 17, includes timestamps and an option to map the coordinates on Google Maps.








<div>Add FilterRun ReportExport TSVExport Excel</div>					
Device ID	GPS Type	Latitude	Longitude	Captured Date/Time	Map It
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 16:50:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 16:20:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 15:20:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 15:50:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 14:50:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 14:20:32	
IMEI: A000002FBA6E4D IMSI: 310008048525158 UUID: b1175548-0920-4939-8b7c-5f25635efbfb MSISDN: [REDACTED]034	N/A	0.0	0.0	2012-03-09 13:50:32	

Figure 17. GPS Capture Interface

Image Capture

The image collections rarely appeared to function properly during the MSS evaluation. Six pictures were taken on the phone; only one was found on the server. The sole reported picture appeared after it was uploaded to a social networking site, and the interface provided no

mechanism to export it or get more information. A screenshot of the interface that lists the reported images can be seen in Figure 18.

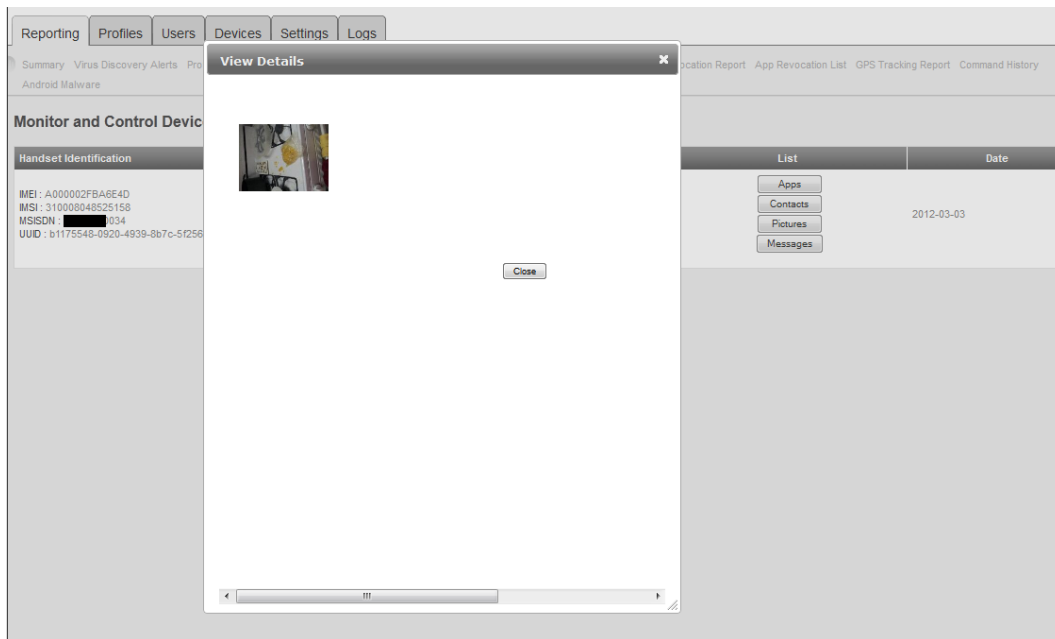


Figure 18. Image Capture Interface

App Capture

The MSG's "Apps—View Details" screen, as seen in Figure 19, shows a listing of all apps currently installed on a phone. Each app is listed alphabetically with its associated package name. Since no timestamps are reported and the interface only displays the current list of installed apps, it is infeasible to identify when an app was installed or removed from a phone. There was also no ability to export the app list from the interface.

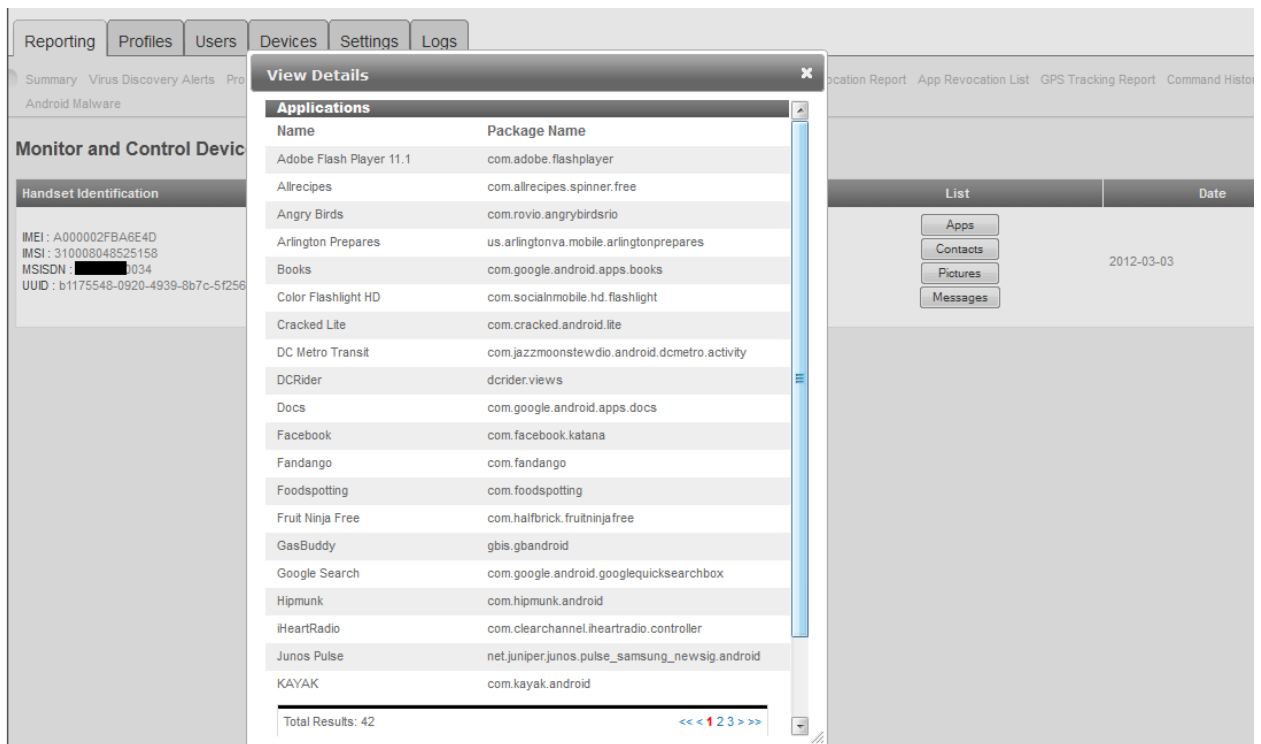


Figure 19. App Capture Interface

Contacts Capture

The “Contacts—View Details” screen, as seen in Figure 20, is a listing of all contacts currently maintained on the phone. The contacts are listed alphabetically by first name and are associated with a phone number. Since no timestamps were reported and the interface only displays the current list of contacts, it is infeasible to detect when a contact was added. There was also no ability to export the contacts list.

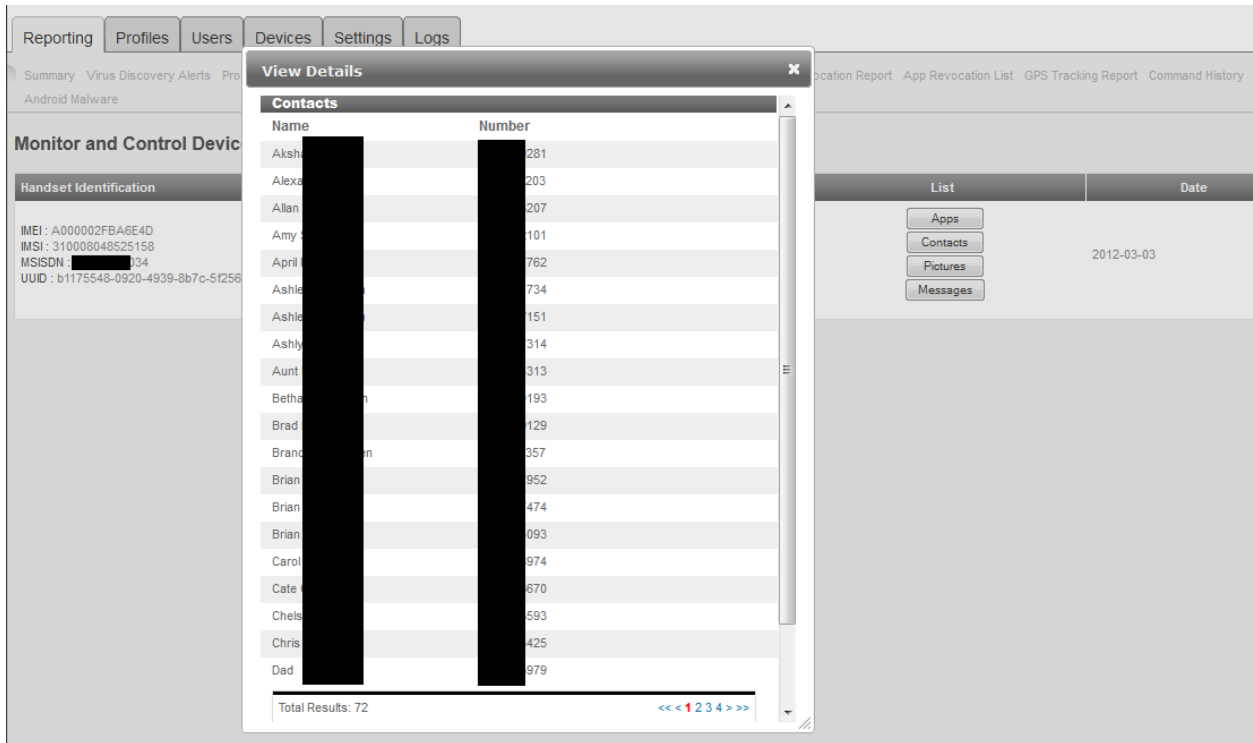


Figure 20. Contacts Capture Interface

Device ID Capture

Under the “Reporting” tab seen in the backgrounds of Figure 18, Figure 19, and Figure 20, several device ID specifications are displayed. Among them are the device’s IMEI, the International Mobile Subscriber Identity, the Mobile Subscriber ISDN, and a unique user ID (UUID). These fields were all reported with the latest associated timestamp, also seen in the interface.

Final Comments

The Junos Pulse MSS is advertised as a SaaS MDM from Juniper. According to a Juniper sales representative, it is not currently possible to buy an MSS server and host it on private infrastructure. This may present problems for organizations that want data stored within their enterprise.

After evaluating the MSS's current user monitoring capabilities, results found that of the fifteen data sets collected by DroidWatch, only seven were addressed in MSS. Of the seven MSS data sets, several were considered incomplete when their usefulness to internal investigations was taken into account. Items such as message directions, call durations, app install timestamps, and contact added timestamps are critical fields for analysts and investigators not found in the MSS. The work performed in this thesis and evaluation highlights the needs for improvement in the MDM user monitoring realm and presents an opportunity for technology transfer.

Appendix C: PHP Server Code

The following PHP code is found on the central webserver to handle the SQLite database files uploaded through HTTPS POST. The files are saved and ingested into the enterprise server MySQL database, which is used by Splunk.

```
<html>
<body>
<?php

// Specify uploaded file path
$target_path = "uploads/"
$target_path = $target_path . basename($_FILES['uploadedfile']['name'], ".db") . "_" .
date('YmdHis') . ".db";

// Move upload to target_path
if (move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target_path))
{
    // Extract deviceID from uploaded SQLite DB
    $db = new PDO("sqlite:$target_path");
    $idQuery = "SELECT device_id FROM transfers ORDER BY _id DESC LIMIT 1";
    $idResult = $db->query($idQuery);
    $idRow = $idResult->fetch(PDO::FETCH_ASSOC);
    $deviceID = $idRow['device_id'];

    // Extract events from uploaded SQLite DB
    $eventQuery = "SELECT * FROM events";
    $eventResult = $db->query($eventQuery);
    $eventRows = $eventResult->fetchall(PDO::FETCH_ASSOC);

    // Store data in MySQL for Splunk
    mysql_connect("localhost", "www-data", "");
    mysql_select_db("events");
    foreach($eventRows as $row)
    {
        $eventID = $row['_id'];
        $detector = mysql_real_escape_string($row['detector']);
        $detected = $row['detected'];
        $action = mysql_real_escape_string($row['action']);
        $occurred = $row['event_occurred'];
        $date_occurred = date('c', $occurred / 1000);
        $description = mysql_real_escape_string($row['description']);
        $additional_info = mysql_real_escape_string($row['additional_info']);
        $query = "INSERT INTO events (_id, detector, action, date_occurred, description,
additional_info, device_id) VALUES
($eventID, '$detector', '$action', '$date_occurred', '$description', '$additional_info', '$deviceID)";
        mysql_query($query);
    }
    $db = NULL;
    mysql_close();
}

?>
</body>
</html>
```

Appendix D: DroidWatch Detectors

Detectors are mechanisms configured to watch for specific new events that happen on an Android device. The detector names correspond with the tags stored with an event. The list of detectors used in this work are in Table 3.

Table 3. DroidWatch Detectors

Detector Name	Event Trigger
AppWatcher	App installation or removal
BrowserWatcher	URL navigations or searches in the Android browser
CalendarWatcher	Added calendar events
CallWatcher	Incoming/outgoing/missed calls
ContactWatcher	Added contacts
DeviceAccountRetriever	Account names (retrieved upon service start)
GalleryWatcher	Added images to the Android Gallery
LocationWatcher	Added last known location coordinates
LogcatWatcher	Added third-party app logs
IncomingMMSWatcher	Incoming MMS messages
OutgoingMMSWatcher	Outgoing MMS messages
ProviderChangeWatcher	Changes to the location provider settings
ScreenWatcher	Changes to the screen lock status
IncomingSMSWatcher	Incoming SMS messages
OutgoingSMSWatcher	Outgoing SMS messages

Appendix E: Content Providers

Table 4 contains a list of data sets associated with content provider URIs used during the development of DroidWatch. Each entry is marked as “undocumented” or “documented” depending on whether the URI is found in the Android API (as of API 10). Data sets that were not collected through a content provider are marked as not applicable (N/A).

Table 4. Content Provider URIs Used in this Research

Data Set	Content Provider URI	Documented?	
		Yes	No
App Installs / Removals	N/A		
Browser Navigation History	android.provider.Browser.BOOKMARKS_URI	✓	
Browser Searches	android.provider.Browser.SEARCHES_URI	✓	
Calendar Events	content://com.android.calendar/event_entities		✓
Call Logs (Incoming / Outgoing / Missed)*	android.provider.CallLog.Calls.CONTENT_URI	✓	
Contacts Added	android.provider.ContactsContract.CommonDataKinds.Phone.CONTENT_URI	✓	
Device Accounts	N/A		
Device ID (e.g., IMEI, MEID, ESN)	N/A		
GPS Location	N/A		
Location Settings (e.g., GPS turned off)	N/A		
MMS (Incoming / Outgoing)*	content://mms content://mms/part content://mms-sms/conversations content://sms		✓
Pictures Added	android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI	✓	
Screen Lock Status (e.g., Unlocked)	N/A		
SMS (Incoming / Outgoing)*	content://sms		✓
Third-Party App Logs (Logcat)	N/A		

**Data sets that use the “Contacts” content provider URI to associate phone numbers with names.*

Appendix F: Glossary

Activity	Screen within an Android application that displays a user interface
Alarm	Scheduled process that can potentially recur over time
Android	Operating system made by Google, Inc. found in many mobile devices
AndroidManifest	Configuration file necessary for all Android applications; contains information about permissions and Android component used
Anti-Forensics	Techniques used to compromise, thwart, or minimize the usefulness of forensics on a system
Apache	Software that creates or enables a Web server
API	Application Programming Interface; set of documented methods that can be used for software component communication
Broadcast Receiver	Android application component; capable of handling system notification events
BYOD	Bring Your Own Device; policy that allows an organization's personnel to use their own mobile devices on an enterprise
Content Observer	Android application component; receives notifications when changes are made to the underlying data that it is watching
Content Provider	Android application component; manages access to an underlying data set
Content Resolver	Android application component; manages access to a content provider
Data Collection Component	Android application components capable of collecting data

Data-at-rest	Data on a system that is not in use; often associated with encryption
Detector	Mechanism configured to watch for specific new events that happen on a system
DroidWatch	Prototype system created as part of this research; composed of an application and a server
Enterprise	Network or networks that comprise an organization
Enterprise App Deployment	Act of installing an app to mobile devices on an organization's network; commonly associated with Mobile Device Management
Gallery	Area on an Android device where pictures are stored
HTTPS POST	Communication protocol used to send form information (possibly containing digital files) over a network in a secure manner
Incident Responder	Person, commonly part of a larger team, who immediately handles or investigates events of concern to an organization
Intent	Internal Android communication message
Intent-Filter	Mechanism that handles internal Android communications; used with a broadcast receiver
Investigation	Systematic examination or research of someone or something; types of investigations referenced in this work include incident response, security audits, proactive monitoring, and law enforcement
Location Provider	Android mechanism that supplies a device with geo-coordinate information; types of location providers referenced in this work are network providers and Global Positioning System (GPS) providers
Logcat	Android tool; outputs application logs

MEID	Mobile Equipment Identifier; unique device identifier; commonly found on code division multiple access (CDMA) phones
MMS	Multimedia Messaging System; mobile device protocol used to send text messages with pictures or other media
MDM	Mobile Device Management; system that manages and enforces security policies on enterprise mobile devices
MySQL	Open-source database system
PDU	Protocol Description Unit; raw format used to store SMS content
PHP	PHP: Hypertext Processor; server-side scripting language
Proactive Monitoring	Type of investigation; typically performed by an analyst on an enterprise; involves the active and continuous exploration for new events of concern
Remote Wipe	Capability of erasing data from a mobile device; commonly found as a feature on Mobile Device Management (MDM) systems
Rooting	Gaining elevated privileges on a system
Security Auditor	Person who investigates past security incidents within an organization
Service	Android application component; runs in the background for long periods of time
Smartphone	Mobile device able to performing advanced computing capabilities such as Internet browsing
SMS	Short Message Service; mobile device protocol used to send and receive text messages

Splunk	Commercially available tool that ingests and displays log data
SQLite	Database management system; commonly used on Android devices
Steganography	Art of hiding messages inside unsuspected mediums
Ubuntu	Open-source Linux operating system; used to implement the server in this research
User Consent Banner	Dialog box displayed to a user that states terms and conditions for monitoring
User-Initiated Platform Modification	Actions taken by an individual to alter components of the Android operating system on a device
Version Reporting	Device identification information collected from a device; common MDM function
VNC	Virtual Network Computing; streaming the screen of a device to a remote location
VPN	Virtual Private Network; extension mechanism that can be used to access private networks across public infrastructure

Appendix G: Acronyms

API	Application Programming Interface
BYOD	Bring Your Own Device
ESN	Electronic Serial Number
MDM	Mobile Device Management
MEID	Mobile Equipment Identifier
MMS	Multimedia Message Service
MSG	Mobile Security Gateway
MSS	Mobile Security Suite
PDU	Protocol Description Unit
PHP	PHP: Hypertext Processor
PID	Process ID
RIM	Research in Motion
SaaS	Software as a Service
SCA	Stored Communications Act
SDCard	Secure Digital Card
SDK	Software Development Kit
SMS	Short Message Service
SQL	Structured Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus

Bibliography

- Android Open Source Project. (2008). *android/platform/packages/apps/Email.gitl*. Retrieved from Android Source Code Repository:
https://android.googlesource.com/platform/packages/apps/Email.git/+/android-2.3.6_r1/AndroidManifest.xml
- Azadegan, S., Yu, W., Liu, H., Sistani, M., & Acharya, S. (2012). Novel Anti-forensics Approaches for Smart Phones. *Hawaii International Conference on System Sciences* (pp. 5424-5431). Maui: IEEE.
- Barbeau, S. (2012, August 3). *LocationManager requestLocationUpdates minTime parameter not working*. Retrieved from GoogleGroups Forum:
<https://groups.google.com/forum/?fromgroups#!topic/android-developers/eyctELx19kc>
- Bray, T. (2011, October 20). *New Public APIs in ICS*. Retrieved from Android Developers Blog:
<http://android-developers.blogspot.com/2011/10/ics-and-non-public-apis.html>
- Bureau of Labor Statistics, U.S. Department of Labor. (2012, March 29). *Private Detectives and Investigators*. Retrieved from Occupational Outlook Handbook, 2012-13 Edition:
<http://www.bls.gov/ooh/Protective-Service/Private-detectives-and-investigators.htm>
- Casey, E. (2009, July 1). *Top 7 ways investigators catch criminals using Mobile Device Forensics*. Retrieved from SANS Forensics Blog: <http://computer-forensics.sans.org/blog/2009/07/01/top-7-ways-investigators-catch-criminals-using-mobile-device-forensics>
- Citrix. (2011, July 22). *IT Organizations Embrace Bring-Your-Own Devices*. Retrieved from Citrix:
http://www.citrix.com/site/resources/dynamic/additional/Citrix_BYO_Index_report.pdf

- Cohen, M., Bilby, D., & Caronni, G. (2011). Distributed forensics and incident response in the enterprise. *Digital Forensics Research Workshop* (pp. S101-S110). New Orleans, LA: Elsevier.
- CommonsWare. (2010, September 28). *Access Android Emails through Content Provider*. Retrieved from StackOverflow: <http://stackoverflow.com/questions/3811608/access-android-emails-through-content-provider>
- comScore, Inc. (2012, November 30). *comScore Reports October 2012 U.S. Mobile Subscriber Market Share*. Retrieved from comScore: http://www.comscore.com/Insights/Press_Releases/2012/11/comScore_Reports_October_2012_U.S._Mobile_Subscriber_Market_Share
- D'Arcy, J., Hovav, A., & Galletta, D. (2009). User Awareness of Security Countermeasures and Its Impact on Information Systems Misuse: A Deterrence Approach. *Information Systems Research*, 79-98.
- Davis, W. (2012, May 21). *Carrier IQ Loses Preliminary Round In Privacy Lawsuit*. Retrieved from MediaPost: <http://www.mediapost.com/publications/article/175096/carrier-iq-loses-preliminary-round-in-privacy-laws.html#axzz2FMwgPlJ6>
- Distefano, A., Me, G., & Pace, F. (2010). Android anti-forensics through a local paradigm. *Digital Forensics Research Workshop* (pp. S95-S103). Portland, Oregon: Elsevier.
- Epstein, Z. (2011, December 5). *Apple, Samsung and six more companies sued over Carrier IQ scandal*. Retrieved from BGR: <http://bgr.com/2011/12/05/apple-samsung-and-six-more-companies-sued-over-carrier-iq-scandal/>
- Garfinkel, S. L. (2011, July 11). *Android Forensics*. Retrieved from Simson Garfinkel's Website: [http://simson.net/ref/2011/2011-07-12 Android Forensics.pdf](http://simson.net/ref/2011/2011-07-12%20Android%20Forensics.pdf)

Garnett, B. (2010, June 21). *Computer Forensic Examiners: PI Licensing Requirement Revisited*.

Retrieved from SANS Computer Forensics Blog: <http://computer-forensics.sans.org/blog/2010/06/21/computer-forensic-examiners-pi-licensing-requirement-revisited/>

Google. (n.d.-a). *AccountManager*. Retrieved from Android Developer Documentation:

<http://developer.android.com/reference/android/accounts/AccountManager.html>

Google. (n.d.-b). *Application Fundamentals*. Retrieved from Android Developer Documentation:

<http://developer.android.com/guide/components/fundamentals.html>

Google. (n.d.-c). *Broadcast Receiver*. Retrieved from Android Developer Documentation:

<http://developer.android.com/reference/android/content/BroadcastReceiver.html>

Google. (n.d.-d). *ContentObserver*. Retrieved from Android Developer Documentation:

<http://developer.android.com/reference/android/database/ContentObserver.html>

Google. (n.d.-e). *Permissions*. Retrieved from Android Developer Documentation:

<http://developer.android.com/guide/topics/security/permissions.html>

Google. (n.d.-f). *TelephonyManager*. Retrieved from Android Developer Documentation:

<http://developer.android.com/reference/android/telephony/TelephonyManager.html>

Harris, R. (2006). Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital Forensics Research Workshop* (pp. S44-S49).

Lafayette, Indiana: Elsevier.

Hoog, A. (2010, March 1). *Open Source Android Digital Forensics Application*. Retrieved from

Andrew Hoog SANS Blog: <http://computer-forensics.sans.org/blog/2010/03/01/open-source-android-digital-forensics-application>

- Hoog, A. (2011). *Android Forensics: Investigation, Analysis and Mobile Security for Google Android*. Waltham, MA: Syngress.
- Jansen, W., & Ayers, R. (2007, May 31). *Guidelines on Cell Phone Forensics*. Retrieved from NIST: <http://csrc.nist.gov/publications/nistpubs/800-101/SP800-101.pdf>
- Juniper Networks. (2012, 06 28). *MTC Mobile Signatures*. Retrieved from Juniper Networks Mobile Threat Center: <http://www.juniper.net/us/en/security/mobile-threat-center/#ANDROID:A.Mobistealth>
- Kang, C. (2012, April 3). *BlackBerry remains official Washington's smartphone even as its maker's fortunes decline*. Retrieved from The Washington Post: http://articles.washingtonpost.com/2012-04-03/business/35454153_1_blackberry-products-smartphone-android-devices
- Kovacevic, N. (2011, July 15). *SMS Broadcastreceiver not called when GO SMS Pro installed*. Retrieved from StackOverflow: <http://stackoverflow.com/questions/6561297/sms-broadcastreceiver-not-called-when-go-sms-pro-installed>
- Kovacik, S., & O'Day, D. R. (2010, November 5). *A Proposed Methodology for Victim Android Phone Analysis by Law Enforcement Investigators*. Retrieved from ViaForensics: <https://viaforensics.com/?fid=Proposed-Methodology-for-Android-Forensics.pdf>
- Lai, E. (2012, February 29). *NSA Testing Motorola Android Smartphones for Top-Secret Calls*. Retrieved from Forbes: <http://www.forbes.com/sites/sap/2012/02/29/nsa-testing-motorola-android-smartphones-for-top-secret-calls/>
- Lee, X., Yang, C.-H., Chen, S., & Wu, J. (2009). Design and Implementation of Forensic System in Android Smart Phone. *The 5th Joint Workshop on Information Security*. Guangzhou,

- China. Retrieved from National Kaohsiung Normal University:
http://crypto.nknu.edu.tw/publications/2010JWIS_Android.pdf
- Lessard, J., & Kessler, G. C. (2010). Android Forensics: Simplifying Cell Phone Examinations.
Small Scale Digital Device Forensics Journal, Vol. 4, No. 1.
- Mackenzie, T. (2012, October 10). *Protect your Android apps with obfuscation*. Retrieved from
TechRepublic: <http://www.techrepublic.com/blog/app-builder/protect-your-android-apps-with-obfuscation/1724>
- MarketsandMarkets. (2012, September 21). *Bring-your-own-device (BYOD), Consumerization of IT (Co-IT) and Enterprise Mobility Market - Global Advancements, Business Models, Market Forecasts & Analysis (2012 – 2017)*. Retrieved from MarketsandMarkets:
<http://www.marketsandmarkets.com/AnalystBriefing/byod-enterprise-mobility-market.asp>
- Marks, J. (2012, October 19). *ICE drops BlackBerry in favor of iPhone*. Retrieved from NextGov
News Website: <http://www.nextgov.com/mobile/2012/10/ice-dumps-blackberry-favor-iphone/58905/>
- National Security Agency. (2012, August). *Mobile Device Management: A Risk Discussion for IT Decision Makers*. Retrieved from NSA Information Assurance:
http://www.nsa.gov/ia/_files/factsheets/mdm_decision_makers.pdf
- Pan, X. (2012, April 27). *ModifyApkWithDexTool*. Retrieved from dex2jar Project Website:
<http://code.google.com/p/dex2jar/wiki/ModifyApkWithDexTool>
- Pettey, C. (2012, October 25). *Gartner Says Two-Thirds of Enterprises Will Adopt a Mobile Device Management Solution for Corporate Liable Users Through 2017*. Retrieved from
Gartner: <http://www.gartner.com/it/page.jsp?id=2213115>

- PVS. (2012, April 10). *Suppress / Block BroadcastReceiver in another app*. Retrieved from StackOverflow: <http://stackoverflow.com/questions/6600266/suppress-block-broadcastreceiver-in-another-app>
- Rauf, D. S. (2012, February 28). *More feds ditch BlackBerrys*. Retrieved from Politico: <http://www.politico.com/news/stories/0212/73369.html>
- Richardson, R. (2010, December 2). *2010 / 2011 CSI Computer Crime and Security Survey*. Retrieved from Computer Security Institute: <http://gocsi.com/survey>
- Shields, C., Frieder, O., & Maloof, M. (2011). A system for the proactive, continuous, and efficient collection of digital forensic evidence. *Digital Forensics Research Workshop* (pp. S3-S11). New Orleans, LA: Elsevier.
- Souppaya, M., & Scarfone, K. (2012, July 9). *Guidelines for Managing and Securing Mobile Devices in the Enterprise (Draft)*. Retrieved from NIST: http://csrc.nist.gov/publications/drafts/800-124r1/draft_sp800-124-rev1.pdf
- TechGuy. (2009, December 08). *Dissection of Mobistealth, Best Android Spy Software*. Retrieved from Articlesbase: <http://www.articlesbase.com/infidelity-articles/dissection-of-mobistealth-best-android-spy-software-1555632.html>
- Townsend, B. (2010, June 20). *Integrating GPS + Android*. Retrieved from Kibilogic.com: <http://kibilogic.com/wordpress/?p=73>
- U.S. Census Bureau. (2010). *USA Quickfacts*. Retrieved from U.S Census Bureau Website: <http://quickfacts.census.gov/qfd/states/00000.html>

U.S. Court of Appeals, 5th Circuit. (2012, December 12). *Fannie Garcia v. City of Laredo, Texas*.

Retrieved from U.S. Court of Appeals Website:

<http://www.ca5.uscourts.gov/opinions%5Cpub%5C11/11-41118-CV0.wpd.pdf>

U.S. Court of Appeals, 9th Circuit. (2007, June 20). *U.S. v. Ziegler*. Retrieved from U.S. Court of Appeals Website:

<http://www.ca9.uscourts.gov/datastore/opinions/2007/06/20/0530177o.pdf>

U.S. Supreme Court. (2010, June 17). *City of Ontario, California, et al. v. Quon et al.* Retrieved from The Supreme Court Website: <http://www.supremecourt.gov/opinions/09pdf/08-1332.pdf>

Valle, S. (2013, January 7). *Android Forensics & Security Testing*. Retrieved from

OpenSecurityTraining: <http://opensecuritytraining.info/AndroidForensics.html>

Vidas, T., Zhang, C., & Christin, N. (2011). Towards a General Collection Methodology for Android Devices. *Digital Forensics Research Workshop*. New Orleans.

Villan, A. G., & Esteve, J. J. (2011, June 19). *Remote Control of Mobile Devices in Android Platform*. Retrieved from Universitat Oberta de Catalunya Institutional Repository: <http://openaccess.uoc.edu/webapps/o2/handle/10609/8131>

White, J. (2012, March 16). *Android Logging*. Retrieved from Intertech Blog:

<http://www.intertech.com/Blog/Post/Android-Logging.aspx>

Yang, C.-H., & Lai, Y.-T. (2012). Design and Implementation of Forensic Systems for Android Devices based on Cloud Computing. *Applied Mathematics & Information Sciences*, 243S-247S.