2013

DroidSpotter: A Forensic Tool for Android Location Data Collection and Analysis

Jeffrey Alan Kramer
Iowa State University

DroidSpotter: A forensic tool for Android location data collection and analysis

by

Jeff Kramer

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

Master of Science

Major: Information Assurance

Program of Study Committee:
Yong Guan, Major Professor
Doug Jacobson
Leslie Miller

Iowa State University
Ames, Iowa
2013

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

AOSP	Android Open Source Project
VM	Virtual Machine
APK	Application Package File
OS	Operating System
IDE	Integrated Development Environment

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Yong Guan, and my committee members, Dr. Doug Jacobson and Dr. Leslie Miller, for their guidance and support throughout the course of this research.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience. I want to also offer my appreciation to those who were willing to listen and help problem solve with me while working on my research especially Benjamin Holland because, without them, this thesis would not have been possible.

Finally, thanks to my family for their encouragement especially my mother for her hours of patience, respect and love.

ABSTRACT

Since the surge in popularity of Android smartphones, creating Android applications and using location data in these applications have soared. Due to how quickly Android applications are being created, it is difficult for companies who have created forensics tools to keep updating their programs in a timely manner for investigators to use. This has created a need for a forensics tool which allows forensic investigators to analyze a new Android application themselves since, in most cases, criminal investigations are a time sensitive matter. The purpose of this research is to introduce a prototype for a tool which would allow investigators to identify where important data is being stored on an Android device and update the forensics tool themselves in order to assist other investigators in the future.

CHAPTER I

INTRODUCTION

In the United States, 136.7 million people owned a smartphone (approximately fifty-eight percent of the mobile market).[1] The major platforms currently in the smartphone market are Google's Android with fifty-two percent of the market, Apple's iOS with thirty-nine percent, BlackBerry's BlackBerry OS with roughly five percent, and Microsoft's Windows Mobile OS with three percent. This translates to 71.1 million Android mobile devices, 53.3 million iOS devices, 7.1 million BlackBerry devices, and 4.1 million Windows Mobile devices. As smartphones rise in popularity, it becomes more important to be able to accurately analyze these devices for forensic evidence.

Using location-based services has become increasingly popular in mobile applications. Social media applications like Facebook, Twitter, and FourSquare use location data to let users tell their friends where they are and what they are doing. Fandango, Yelp, and Google Local are being used to find nearby attractions like restaurants, movie theaters, hair salons, and more. MapQuest, Google Navigation, and Google Maps help users navigate to a destination of their choice. All of these applications use the GPS device in Android smartphones and have a possibility of storing location data on the mobile device for future analysis by a forensics scientist.

New applications for the Android platform are being released every day. Google has a minimal screening process before they put applications on the Google Play market meaning anyone can create and release an application on the Google Play marketplace. According to AppBrain and Google, the Google Play market now has over 700,000 applications. Due to this large number of applications, it would take an extremely long amount of time to manually search all of these applications for location data. Not only are there a large number of applications currently on the market, the number of applications are growing every day. There can be anywhere from 15,000 to 37,500 new applications on the Android Market every month.[2] The mobile forensic community needs a toolset to keep track of where location data is being stored and to analyze new applications when they are released for new files containing location data – these are the problems this thesis hopes to address.

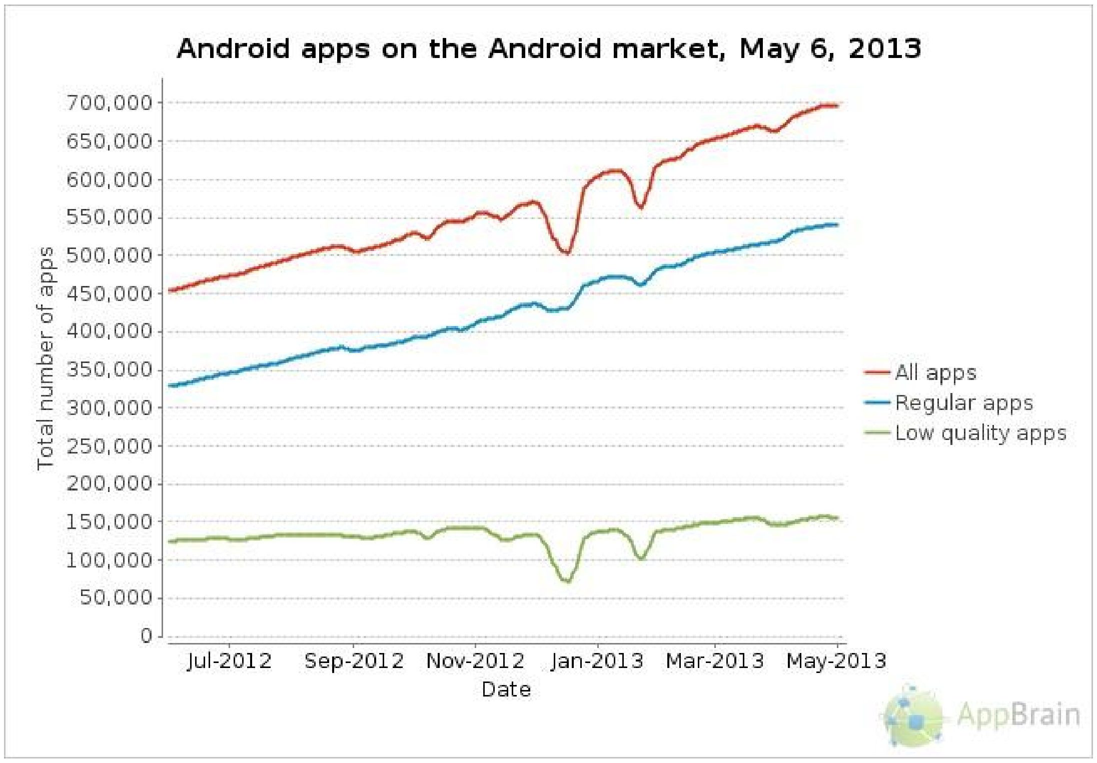


Figure 1. *Growth of Android applications between June 2012 and May 2013*

Background

The Android System Architecture

The Android System Architecture Framework demonstrates how the different components of the Android system interact with one another. There are four different layers to the Android architecture: the Applications, Application Framework, Libraries, and Linux Kernel.[4] Each section relies on another to carry out an operation on the Android device. Android applications, shown in the first layer, are written in Java and run within a Dalvik virtual machine on the mobile device. The application frameworks are skeletons for Android developers to utilize. Through these frameworks, a developer

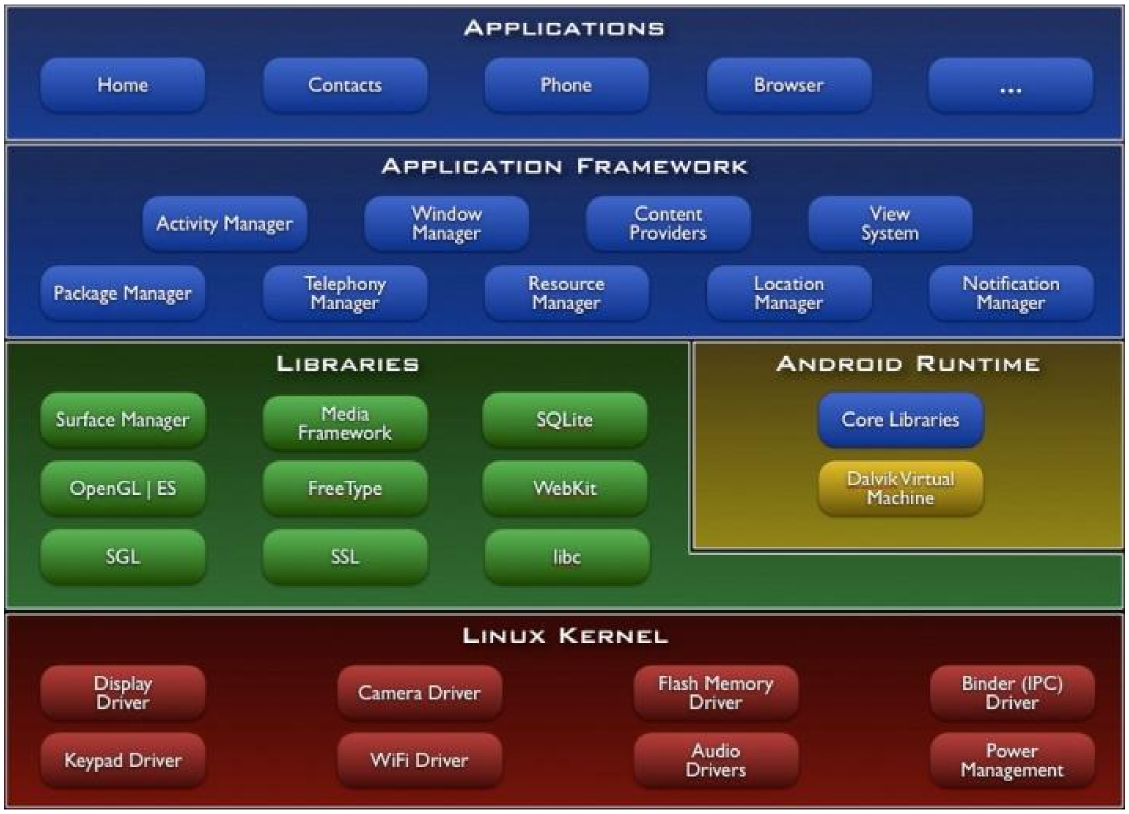


Figure 2. *Diagram of the Android Architecture*

can access all the framework APIs to manage the phone's basic functions like tracking the phone's location, switching between processes, and more. In our case, we will be focusing on the Location Manager since it is in charge of the device's GPS and wireless location data. The library layer consists of different Android libraries written in C and C++. These library allow the device to properly handle certain types of data like SQLite, graphics, and secure sockets layer (SSL). The Linux kernel provides all of the hardware drivers and provides Android's memory management, power management, and other low level functionality.

Android partitions

Another aspect of Android we must understand is how the data is partitioned. There are eight partitions in total: boot, system, recovery, data, cache, misc, sdcard, and sd-ext.[5] The boot partition allows the Android device to boot and includes the kernel and ramdisk. There is no valuable data which should be retrievable from this partition since applications will not be saving to it.

The system partition contains the entirety of the operating system including components like the user interface and system applications which come pre-installed from the manufacturer. Since the system partition contains Android applications on it, it will be of interest to us and should be investigated. The recovery partition is basically a backup partition which allows you to boot into the recovery console to perform recovery and maintenance on the device.

The data partition, also called userdata, contains any user data stored on the device such as contacts, messages, and any installed applications from the Google Play market. Applications will use this partition to store any data needed for future use. This partition will be of interest when we are looking for location data. The cache partition contains data Android frequently accesses and certain app components. This partition could potentially have location data if a user is frequently in the same place or the data was recently created. The misc partition contains miscellaneous system settings like CID, USB configuration, and other hardware settings. The misc partition should not contain any location data and will not be analyzed.

The sdcard and sd-ext are both partitions pertaining to the SD card if the device has one. Android devices can have an internal and external SD card. If this is the case, the sdcard partition will always refer to the internal SD card. The sdcard partition can store anything the user wishes such as documents, media, custom ROMs, and more. The sd-ext partition is used mainly by custom ROMs as another data partition. This can be used to install more applications than normally possible with only the memory available internally. The sdcard partitions will be analyzed in case an application develop decides to store data on these partitions. In conclusion, the data, system, cache, and sdcard partitions will be of interest during our search for location data being saved on an Android device.

CHAPTER II

LITERATURE SURVEY

Commercial Products

Due to the popularity of Android, professional tools have been created to do forensic analysis for police work and courtroom evidence. Mobile Phone Examiner PLUS (MPE+) created by AccessData is one such toolkit.[6] The MPE+ has a large number of useful features like searching for emails, contacts, and phone numbers. It can automatically recognize certain applications like Facebook, Twitter, LinkedIn, etc. and grab data from them. However, it does not have the ability to do analysis on new applications and requires investigators to wait for the company to update their software or go through the analysis process themselves.

viaExtract is another tool created by viaForensics.[7] This tool also has a large number of features like being able to bypass the lock (if adb is enabled on the device), recovery of call logs, contacts, browser history, etc., and more. viaExtract allows an investigator to save all of their data onto an external SD card similar to MPE+. However, there is no tool to allow the investigator to examine new applications.

Academic Work

There have been other efforts in the academic world concerning Android forensics. One study, by Chen, Yang, and Liu [8], created a tool placed on an SD card and would insert the card into the Android device they wanted to investigate. This tool would copy forensic data from the phone onto the SD card. However, this tool requires usage of Fastboot (HBoot) mode and an SD card slot on the phone, an SD card, and a computer with an SD card reader to do analysis. DroidSpotter only requires the device to have the Android operating system and a computer capable of running a virtual machine.

Rochester Institute of Technology's Justin Grover created a tool called DroidWatch as a result of his research on "Automated Data Collection and Reporting from a Mobile Device".[9] This research involved creating an Android application which would send data on the device to another system via HTTP. This research was focused on monitoring a device. DroidSpotter wants to retrieve data from a phone which is currently in custody of a forensics examiner and be able to determine the location history of a suspect.

CHAPTER III

DROIDSPOTTER: DESIGN AND IMPLEMENTATION DETAILS

Overview of DroidSpotter

The main goal of DroidSpotter is to allow forensic scientists the ability to narrow down the number of possible locations of location data from unanalyzed applications on an image of an Android partition copied from its original hard drive. This analysis is made possible by the Android Open Source Project's emulator. This functionality will be combined with a Java program backed by a MySQL database which will keep track of where data has been found and allow scientists to update the database for future use. A virtual machine will be used to hold all of the portions of this project in order to keep researchers safe from malware. Figure 3 shows a visual representation of the expected forensic process an investigator would go through when using DroidSpotter.

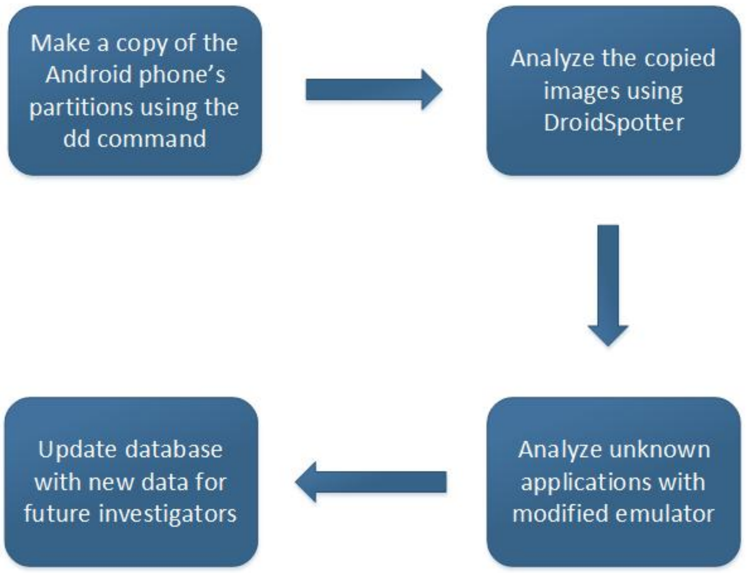

Make a copy of the Android phone's partitions using the dd command
Analyze the copied images using DroidSpotter
Analyze unknown applications with modified emulator
Update database with new data for future investigators

Figure 3. *The expected forensic process when using DroidSpotter*

Foundation of DroidSpotter

Several steps were taken before beginning to create the forensic tool DroidSpotter. The first step involved creating sample location data on an Android device in order to ensure the data being found in the research was valid. The second step was creating a virtual machine to host the Java program and emulator. The third and final step was downloading and getting the Android Open Source emulator to compile.

Creating sample location data

In order to know the process was capable of gathering data from an Android device, sample data was created using a Motorola RAZR MAXX smartphone. Applications like Google Latitude, Facebook, Twitter, and more were used to report its location in various ways on the device. The phone was then rooted via a process detailed on the XDA Forums. Applications like SSHDroid for Android and WinSCP for Windows were used in order to transfer the files to a desktop computer for analysis. Once the files were transferred to a desktop computer, they were manually analyzed the files for GPS data. After the existence of location data being saved in files on the mobile device was confirmed, the next stage of the implementation of the tool could start: creating a tool to identify the location of GPS data files on Android devices.

Virtual machine

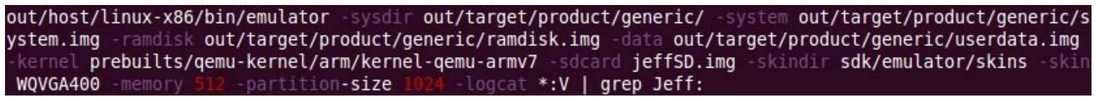
Since the goal of this project was to create a professional forensics tool, the decision to host the tool on a virtual machine using VMWare's vSphere client was beneficial for many reasons. Virtual machines are usable on most operating systems which means any organization or company who decides to use the tool or further build upon it will not have to buy new equipment in order to do so. Virtual machines use very little resources on the host operating system which means an organization can host the VM on a server and access the VM with most modern computers. A virtual machine also allows room for error. If a developer is unsure about the ramifications of a certain action or code change they are about to make, they can simply take a snapshot and revert to it if the results are unfavorable.

Emulator

Creating an Android device on a desktop is possible through the use of the Android Open Source Project (AOSP).[10] Google has made the Android source code publically available for developers to use and modify for their own purposes. During my first attempt at compiling the Android source code, the Ubuntu 12.04 operating system was being used. However, it is only experimentally supported. The code compilation was consistently giving errors so Ubuntu 10.04 was used instead. There are two options you must choose for building an Android emulator: the build and the build type. For the build, there are three options – full, full_maguro, and full_panda. The decision to use the full option was made because it is configured to work with all languages, apps, and input methods. The buildtype options are user, userdebug, and eng. The eng option is a development configuration which allows for additional debugging tools. The eng option seemed most fitting since additional debugging tools may be needed during the development process.

In order to get the Android source code to compile and run, several steps must be taken. First, the envsetup.sh script is run to setup the environment variables for the emulator. Second, the lunch command is run with the desired build and build types specified. Third, the make command is run to recompile any code which has been modified since the last run of the emulator. Finally, the emulator command is run.

There are several different options for the emulator command. Here are the options in the emulator used for this tool:



```
out/host/linux-x86/bin/emulator -sysdir out/target/product/generic/ -system out/target/product/generic/s
ystem.img -ramdisk out/target/product/generic/ramdisk.img -data out/target/product/generic/userdata.img
-kernel prebuilts/qemu-kernel/arm/kernel-qemu-armv7 -sdcard jeffSD.img -skindir sdk/emulator/skins -skin
 WQVGA400 -memory 512 -partition-size 1024 -logcat *:V | grep Jeff:
```

Figure 4. *Emulator command used for DroidSpotter*

The emulator portion is simply the name of the command itself. Any of the other options which contain locations are relative to the /home/jeff/WORKING_DIRECTORY folder. The –sysdir option specifies the location of the system directory which is out/target/product/generic in this case. The –system option specifies the location of the system.img file which is created after the source code is compiled and made. The –ramdisk, -data, and –sdcard options specify the location of their respective image (.img) files. –Kernel specifies the location of the kernel for the emulator. –Skindir specifies where the skins for the emulators are located which make the emulator to look like specific devices. The –skin option is a specific example of one of these skins. The –memory option specifies how much RAM the emulated device should receive and the –

partition-size option states how much hard drive space the emulator partition will be allocated. The above options is the configuration used to run the emulator on the virtual machine.

The –logcat option tells the emulator the user wants to view the output from logcat as the emulator is running. There are seven different priorities for logcat and each priority has its own letter code: Verbose (V), Debug (D), Info (I), Warning (W), Error (E), Fatal (F), and Silent (S). Verbose is the lowest priority meaning every log message will be outputted. Since the messages output by this tool are not considered high priority, the Verbose tag is used when outputting information about when the Location API is called. These modifications will be shown in more detail later. The output of logcat is piped into the grep command so only the relevant output will be displayed.

Modifying the Android source code

Once the code was compiling properly, it was time to modify the Android source code to have it report when the Location API calls are being made. In the Android source code, there is a Location package with several classes. The most important ones are the Location, LocationManager, and LocationProvider classes.

The Location class is an object used to represent a geographical spot on the globe. A Location consists of a latitude, longitude, timestamp, and other information such as bearing, altitude, and velocity. This is good since this means every Location object will have coordinates and an associated timestamp. This will allow us to know where the phone was and when the Location object was created.

The LocationManager class provides access to the system location services. The LocationManager class is used by applications to obtain updates of the device's geographic location and it is used to access the location providers. The LocationManager is used to register location update listeners and notifies the device when the location of the device changes.

The LocationProvider class is a superclass of different location providers which deliver information about the current location. There are three different types of location providers: network, GPS, and passive. The network provider uses the mobile network signal or Wi-Fi to determine the device's location. The GPS provider uses the GPS receiver in the mobile device to determine its location via global positioning satellites. The passive provider is a special location provider for receiving locations without actually initiating a location fix by providing locations generated by other providers.

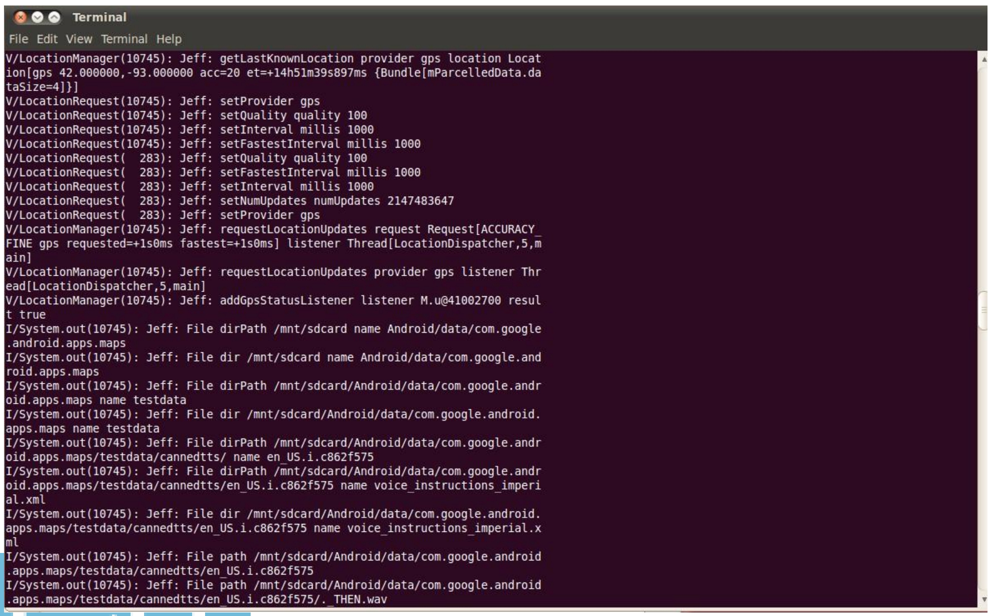
In addition to modifying the Android Location classes, the File Java library was modified as well. Whenever a file is created, a message will be sent to logcat via the System.out.println call of Java. The filename will be printed along with the filepath if it is known. This allows the analyst to see any files created when an action is taken on the emulator.

Finding location data

The process to find location data is fairly simple using this tool. Once you have made the emulator, start the emulator and unlock the device. Next, you must install the desired APK to the emulator. To install an application, simply type './adb install

<location of APK>' without the quotes and brackets. A terminal window will be brought up and tell you the results of the installation process. If the installation has completed successfully, the next step is creating a fake location for the emulated GPS on the emulator. This is accomplished by using telnet to connect to the emulator by typing 'telnet localhost <port number>'. The port number is displayed at the top of the window of the emulator and is typically run on port 5554. When the telnet connection has been established, the command 'geo fix <longitude> <latitude>' can be entered to give the GPS a fake location for the device where longitude and latitude are the coordinate values the user desires. Once this has finished, go to the emulator and start up the application. The user should then interact with the application in such a way to cause the location API to be called.

Anytime the GPS module is used by an application, a message will be output to the emulator's terminal via logcat. The most important message comes from the LocationManager stating what the provider and current coordinates of the phone is according to the device reporting the location. Any subsequent files created shortly after



```
V/LocationManager(10745): Jeff: getLastKnownLocation provider gps location Locat
ion[gps 42.000000,-93.000000 acc=20 et=+14h51m39s897ms {Bundle[mParcelledData.da
taSize=4]}]
V/LocationRequest(10745): Jeff: setProvider gps
V/LocationRequest(10745): Jeff: setQuality quality 100
V/LocationRequest(10745): Jeff: setInterval millis 1000
V/LocationRequest(10745): Jeff: setFastestInterval millis 1000
V/LocationRequest(  283): Jeff: setQuality quality 100
V/LocationRequest(  283): Jeff: setFastestInterval millis 1000
V/LocationRequest(  283): Jeff: setInterval millis 1000
V/LocationRequest(  283): Jeff: setNumUpdates numUpdates 2147483647
V/LocationRequest(  283): Jeff: setProvider gps
V/LocationManager(10745): Jeff: requestLocationUpdates request Request[ACCURACY_
FINE gps requested=+1s0ms fastest=+1s0ms] listener Thread[LocationDispatcher,5,m
ain]
V/LocationManager(10745): Jeff: requestLocationUpdates provider gps listener Thr
ead[LocationDispatcher,5,main]
V/LocationManager(10745): Jeff: addGpsStatusListener listener M.u@41002700 resul
t true
I/System.out(10745): Jeff: File dirPath /mnt/sdcard name Android/data/com.google
.android.apps.maps
I/System.out(10745): Jeff: File dir /mnt/sdcard name Android/data/com.google.and
roid.apps.maps
I/System.out(10745): Jeff: File dirPath /mnt/sdcard/Android/data/com.google.andr
oid.apps.maps name testdata
I/System.out(10745): Jeff: File dir /mnt/sdcard/Android/data/com.google.android.
apps.maps name testdata
I/System.out(10745): Jeff: File dirPath /mnt/sdcard/Android/data/com.google.andr
oid.apps.maps/testdata/cannedtts/ name en_US.i.c862f575
I/System.out(10745): Jeff: File dirPath /mnt/sdcard/Android/data/com.google.andr
oid.apps.maps/testdata/cannedtts/en_US.i.c862f575 name voice_instructions_imperi
al.xml
I/System.out(10745): Jeff: File dir /mnt/sdcard/Android/data/com.google.android.
apps.maps/testdata/cannedtts/en_US.i.c862f575 name voice_instructions_imperial.x
ml
I/System.out(10745): Jeff: File path /mnt/sdcard/Android/data/com.google.android
.apps.maps/testdata/cannedtts/en_US.i.c862f575
I/System.out(10745): Jeff: File path /mnt/sdcard/Android/data/com.google.android
.apps.maps/testdata/cannedtts/en_US.i.c862f575/._THEN.wav
```

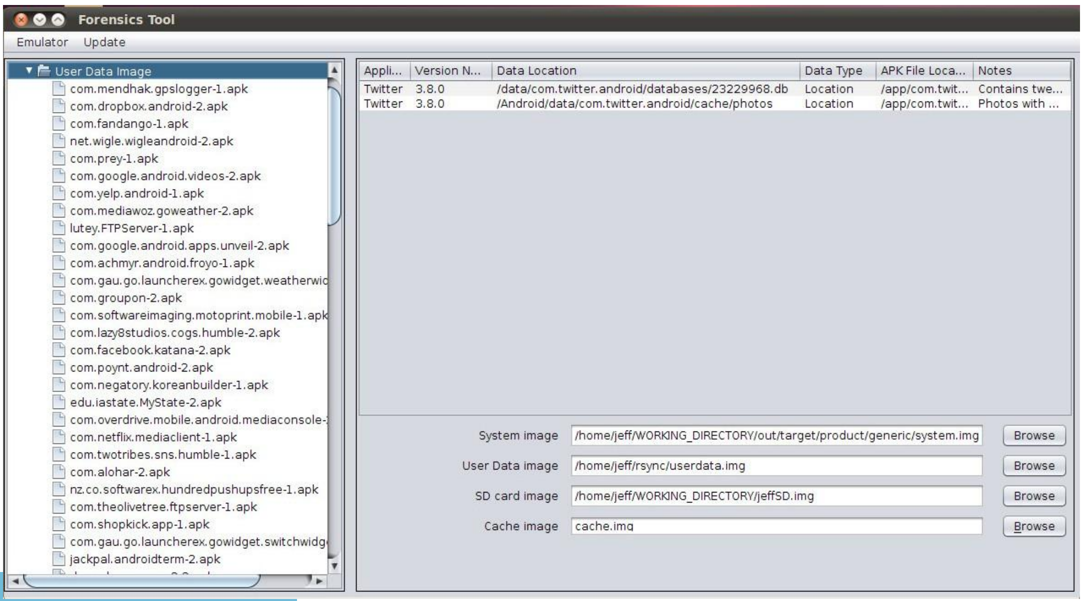
Figure 5. *Modified logcat output from the Android emulator for Google Navigation*

the GPS has been updated has a potential to contain location data. In most cases, files which actually contain location data is contained within SQLite databases within a database folder commonly found in Android applications.

Creating DroidSpotter

DroidSpotter was written entirely in Java. To run the tool, simply run Eclipse and run the program. The NetBeans IDE was used to create the user interface. This IDE allowed for quick implementation of the main window, all of the buttons, JTree, and FileChoosers for the tool. The design for the user interface is fairly simple. The list of APKs are in the far left in a file directory structure and only contain APKs on the selected image. Any location data on the APK will be loaded in the center portion of the screen when the APK is selected in the JTree on the left.

This forensics tool stores all of its data in a basic SQL database. This database has many columns and certain columns are not currently necessary but are placeholders for future work. The first column is a unique integer identifier, created for a primary key



User Data Image
- com.mendhak.gpslogger-1.apk
- com.dropbox.android-2.apk
- com.fandango-1.apk
- net.wigle.wigleandroid-2.apk
- com.prey-1.apk
- com.google.android.videos-2.apk
- com.yelp.android-1.apk
- com.mediawoz.goweather-2.apk
- lutey.FTPServer-1.apk
- com.google.android.apps.unveil-2.apk
- com.achmyr.android.froyo-1.apk
- com.gau.go.launcherex.gowidget.weatherwic
- com.groupon-2.apk
- com.softwareimaging.motoprint.mobile-1.apk
- com.lazy8studios.cogs.humble-2.apk
- com.facebook.katana-2.apk
- com.poynt.android-2.apk
- com.negatory.koreanbuilder-1.apk
- edu.iastate.MyState-2.apk
- com.overdrive.mobile.android.mediaconsole-
- com.netflix.mediaclient-1.apk
- com.twotribes.sns.humble-1.apk
- com.alohar-2.apk
- nz.co.softwarex.hundredpushupsfree-1.apk
- com.theolivetree.ftpserver-1.apk
- com.shopkick.app-1.apk
- com.gau.go.launcherex.gowidget.switchwidg
- jackpal.androidterm-2.apk

Appli...	Version N...	Data Location	Data Type	APK File Loca...	Notes
Twitter	3.8.0	/data/com.twitter.android/databases/23229968.db	Location	/app/com.twit...	Contains twe...
Twitter	3.8.0	/Android/data/com.twitter.android/cache/photos	Location	/app/com.twit...	Photos with ...

System image /home/jeff/WORKING_DIRECTORY/out/target/product/generic/system.img

User Data image /home/jeff/rsync/userdata.img

SD card image /home/jeff/WORKING_DIRECTORY/jeffSD.img

Cache image cache.img

Figure 6. *GUI for DroidSpotter created with NetBeans*

and to ensure each row is unique. The next column is the application name – the real name of the application as it would appear on the device. The application version is stored in the next column, in case future versions of an application behave differently or store location data in a different file. The data type is another column included in the database. This column is used to denote what type of information is stored in the file. Currently, this column will always be "Location" since this research is only concerned with location data. In the future, there may be other kinds of data stored in this database. Data location is where on the image the information is stored relative to the image.

Partition name column states what image the data is stored on. Notes is a column used by the creator of the entry to add details the analyst may need to know about the data. The app location column is used to maintain where the application file is stored relative to the data partition. APK location is a column denoting where the APK is being stored for the purpose of the database. All of the APKs analyzed during this research are including on the hard drive of the virtual machine. These APKs can be installed to the emulator and can be of use to those who may not possess an Android device. The apkName is the column that states what the name of the APK file is when it is found natively on an Android device.

In order to mount the images specified in the UI of the tool, the sudoers file was edited to make it so the mount command does not require a password. This is done by using the visudo command to open the sudoers file. The NOPASSWD option is then added to the commands needed, in this case the mount and find commands. The find command is added to ensure when searching for APKs in the images that no files are excluded due to permission issues.

Once the sudoers file was modified, bash scripts were created for each image to assist in mounting the image. Each partition has a dedicated folder to be mounted on: sysdir for system, userdir for userdata, sdcarddir for the SD card, and cachedir for the cache image. The script tests to see if an image is currently mounted in the directory for the partition. If there is an image currently mounted, it is unmounted and the new image is mounted to the directory. Otherwise, the image is simply mounted to the directory. The names of these scripts correspond to the image being mounted: system.sh, userdata.sh, sdcard.sh, and cache.sh.

With the image now mounted, the tool recursively searches for any application in the subsequent folders. A file is determined to be an application based on its .apk extension. All APKs are placed into the image's folder in the JTree on the left side of the tool's UI. The APKs are represented by their filename, also known as the APK name in the database. When an APK is selected in the JTree, the APK file is found in the hard drive via the find command. The SQL database is then queried based on the APK name and version. If the database contains any results for the selected APK, they are populate in the table in the middle of the screen. No data for the APK in the database will result in the table being cleared. Once the results are populated in the table, the user can double click on the entry in the database and the file will be opened with its default application as determined by the operating system of the virtual machine.

CHAPTER IV
CASE STUDIES

The main point of this research was to create DroidSpotter and have it be able to successfully identify new data for forensic investigators. In order to ensure this was accomplished, several different Android applications were analyzed. Applications were examined by attempting to use them in a way which caused the Location API in the Android operating system to be called. If the application caused the Location API to be used, any subsequent files output by the emulator were investigated for saved location data. The more interesting applications will be discussed in detail here.

Twitter

To analyze the Twitter application, a tweet was created with a photo attached to it taken by the camera provided by the Android operating system. This tweet was then sent out using the application. My personal Twitter account was used to create and send out the tweet. The database file 23229968.db was found after sending out the tweet. This database contains several different tables, the most interesting of which is 'statuses' as shown in Figure 6. This SQLite file shows the most recent tweets in a particular user's feed with a wealth of information about the post such as a UNIX timestamp for when it was created, what application was used to send the tweet, the unique ID for the author of the tweet, and if the user had location data enabled for Twitter, a longitude and latitude of the user's device. Inside the application's cache folder, there is a dedicated photos folder. This folder contains any photos associated with the Twitter account. If the photos are taken with a GPS camera, they may contain location data in the EXIF geolocation tags.

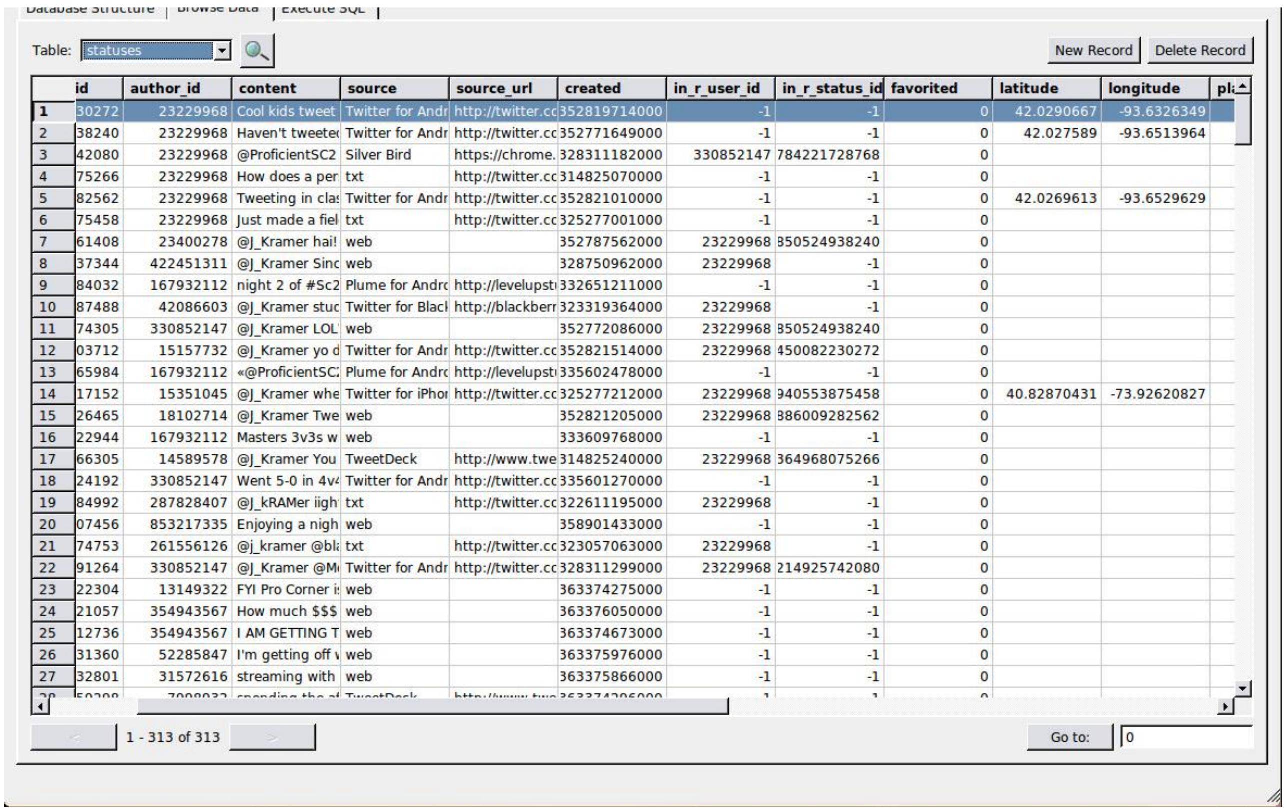


Table: statuses

	id	author_id	content	source	source_url	created	in_r_user_id	in_r_status_id	favorited	latitude	longitude	pl
1	30272	23229968	Cool kids tweet	Twitter for Andr	http://twitter.cc	352819714000	-1	-1	0	42.0290667	-93.6326349	
2	38240	23229968	Haven't tweete	Twitter for Andr	http://twitter.cc	352771649000	-1	-1	0	42.027589	-93.6513964	
3	42080	23229968	@ProficientSC2	Silver Bird	https://chrome.	328311182000	330852147	784221728768	0			
4	75266	23229968	How does a per	txt	http://twitter.cc	314825070000	-1	-1	0			
5	82562	23229968	Tweeting in clas	Twitter for Andr	http://twitter.cc	352821010000	-1	-1	0	42.0269613	-93.6529629	
6	75458	23229968	Just made a fiel	txt	http://twitter.cc	325277001000	-1	-1	0			
7	61408	23400278	@J_Kramer hai!	web		352787562000	23229968	350524938240	0			
8	37344	422451311	@J_Kramer Sinc	web		328750962000	23229968	-1	0			
9	84032	167932112	night 2 of #Sc2	Plume for Andrc	http://levelupst	332651211000	-1	-1	0			
10	87488	42086603	@J_Kramer stuc	Twitter for Blacl	http://blackberr	323319364000	23229968	-1	0			
11	74305	330852147	@J_Kramer LOL	web		352772086000	23229968	350524938240	0			
12	03712	15157732	@J_Kramer yo d	Twitter for Andr	http://twitter.cc	352821514000	23229968	450082230272	0			
13	65984	167932112	«@ProficientSC:	Plume for Andrc	http://levelupst	335602478000	-1	-1	0			
14	17152	15351045	@J_Kramer whe	Twitter for iPhor	http://twitter.cc	325277212000	23229968	940553875458	0	40.82870431	-73.92620827	
15	26465	18102714	@J_Kramer Twe	web		352821205000	23229968	386009282562	0			
16	22944	167932112	Masters 3v3s w	web		333609768000	-1	-1	0			
17	66305	14589578	@J_Kramer You	TweetDeck	http://www.twe	314825240000	23229968	364968075266	0			
18	24192	330852147	Went 5-0 in 4v4	Twitter for Andr	http://twitter.cc	335601270000	-1	-1	0			
19	84992	287828407	@J_kRAMer iigh	txt	http://twitter.cc	322611195000	23229968	-1	0			
20	07456	853217335	Enjoying a nigh	web		358901433000	-1	-1	0			
21	74753	261556126	@j_kramer @bla	txt	http://twitter.cc	323057063000	23229968	-1	0			
22	91264	330852147	@J_Kramer @M	Twitter for Andr	http://twitter.cc	328311299000	23229968	214925742080	0			
23	22304	13149322	FYI Pro Corner i	web		363374275000	-1	-1	0			
24	21057	354943567	How much $$$	web		363376050000	-1	-1	0			
25	12736	354943567	I AM GETTING T	web		363374673000	-1	-1	0			
26	31360	52285847	I'm getting off v	web		363375976000	-1	-1	0			
27	32801	31572616	streaming with	web		363375866000	-1	-1	0			

1 - 313 of 313

Go to: 0

Figure 7. *My personal Twitter account's SQLite database - 23229968.db*

Google Maps

There are several applications associated with Google Maps such as Google Navigation and Google Local. The Google Navigation application was used to discover the location data files in this research. The data was created by searching for directions to a nearby location. A file named da_destination_history is a SQLite database containing the records of any locations the user requested the application for directions. The destination_history table has multiple columns which contains pertinent data. The database has a column with a UNIX timestamp for when the searches were made, the destination's longitude and latitude, the source's latitude and longitude, as well as the name of the destination.

Another database file named search_history.db contains any searches done by the user. The latitude and longitude columns for the searched location are present in the table as well as a timestamp. The device's location can be in this database too if the user decides to include it in the search.

The Google Latitude application has a database file associated with it as well. The file, google_latitude_latitudeLocations.db, has columns for a UNIX timestamp and encrypted values for latitude and longitude. However, the decryption algorithm is unknown and the purpose of this research is not to decipher Google's encryption method.

CHAPTER V

DISCUSSIONS

Due to having a finite amount of time to work on this research, there are still many features which could be added to this toolset. The current process could be automated to a greater extent by batch installing APKs located in the database or in a group of folders located on the VM to the emulator. This process would make it easier to analyze a large number of new applications in a short amount of time for location data. It should also be possible to automate the process of a user interacting with the application through UI fuzzing. UI fuzz testing has been done to test operating systems by the University of Wisconsin Madison such as Windows NT and Mac OS X. This process should be possible on the Android emulator and could generate random input to attempt to output location data. This process could be hindered by applications which require login credentials however.

This type of analysis could be extended to other types of data available through the Android API. There are several other Android and Java classes which could be of interest to a forensics analyst like contacts, SQL calls, URLs, HTTP request, calendar data, and more. However, this functionality is outside the scope of this research since we are solely concerned with location data.

Of course, more files containing location data for the database would always be an improvement. No paid applications were analyzed during this research since there was no outside funding or grants to pay for these applications. Old versions of APKs are not officially supported by Google and therefore are difficult to find. It is possible to find these APKs on third party websites around the internet but the integrity and legitimacy of the APKs is questionable at best. The best option is to crowd source the database and have the public contribute any old APKs they possess on their phones for research purposes.

Another interesting idea would be to create a visual timeline of the location data present on all of the images currently being analyzed. This would be fairly simple to implement since all of the data encountered so far have coordinates and UNIX timestamps. Assuming all the images were from the same phone, this visual timeline would give the forensics analyst an idea of the past behavior of a suspect using the data generated from this tool. Google Maps could be used to create this visual representation of the data but Google does not allow their Google Maps API to be used in applications which are not “publically accessible”.

CHAPTER VI

SUMMARY AND CONCLUSION

As shown by this research, location data is stored on Android devices by many applications. The increase in applications using location data will make finding this information on a suspect's device more valuable. It seems highly plausible a timeline of a person's whereabouts could be created from this information and used in an investigation. It is also incredibly important an investigator finds this information in a timely fashion. This is the reason why the emulator is a vital part of DroidSpotter. The emulator allows the investigator to easily find location data for newer applications which may have never been analyzed before. The forensics community could assist with finding more information with this tool and create a vast database of forensic information which would be invaluable in helping catch criminals.

REFERENCES

1. "ComScore Reports March 2013 U.S. Smartphone Subscriber Market Share." *ComScore, Inc*. ComScore, Inc, n.d. Web. 06 May 2013. <http://www.comscore.com/Insights/Press_Releases/2013/5/comScore_Reports_March_2013_U.S._Smartphone_Subscriber_Market_Share>.

2. "Android Statistics - Development over Time." *AppBrain*. AppBrain, n.d. Web. 17 May 2013. <http://www.appbrain.com/stats/number-of-android-apps>.

3. "Android Architecture." *ELinux.org*. ELinux, n.d. Web. 17 May 2013. <http://elinux.org/Android_Architecture>.

4. Raja, Haroon Q. "Android Partitions Explained: Boot, System, Recovery, Data, Cache & Misc." *AddictiveTips*. AddictiveTips, 19 May 2011. Web. 17 May 2013. <http://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/>.

5. "Mobile Phone Examiner Plus." *AccessData*. AccessData Group, Inc., n.d. Web. 17 May 2013. <http://www.accessdata.com/products/digital-forensics/mobile-phone-examiner>.

6. "ViaExtract." *ViaForensics*. ViaForensics, n.d. Web. 17 May 2013. <https://viaforensics.com/products/viaextract/>.

7. Chen, Sheng-Wen, Chung-Huang Yang, and Chien-Tsung Liu. "Design and Implementation of Live SD Acquisition Tool in Android Smart Phone." *IEEE Xplore*. IEEE, n.d. Web. 18 May 2013. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6042741>.

8. Grover, Justin. "Android Forensics: Automated Data Collection and Reporting from a Mobile Device." *Rochester Institute of Technology*. Rochester Institute of Technology, n.d. Web. 17 May 2013. <https://ritdml.rit.edu/bitstream/handle/1850/15949/JGroverThesis1-31-2013.pdf?sequence=1>.

9. "Welcome to Android." *Android Open Source Project*. Google, n.d. Web. 06 May 2013. <http://source.android.com/>.