7CCSMRJ

Individual Project Submission

2015/16

Name: Felix Santiago Anda Basabe

Student Number: 1567560

Degree Programme: MSc in Computing & Security

Project Title: Forensic Data Recovery from Android Smart Watches

Supervisor: Dr Nishanth Sastry

Word Count: 8482

RELEASE OF PRODUCT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I agree to the release of my project

☐ I do not agree to the release of my project

Signature: Date: August 26, 2016
Forensic Data Recovery from Android Smart Watches

Felix Santiago Anda Basabe
Student Number: 1567560
Course: MSc in Computing & Security

Supervisor: Dr Nishanth Sastry

Thesis submitted as part of the requirements for the award of the MSc in Computing & Security.

7CCSMPRJ - MSc Individual Project - 2016
Abstract

Forensically sound methods for data acquisition in Android based devices are discussable because of the complexity of obtaining data without modifying any state of the wearable. Obtaining relevant data from a smart watch that could be used as digital evidence due to its probative value will lead to the emerge of guidelines for good practices. The recollection of data and analysis of potential evidence is limited to rooted and non-rooted Smartwatches with Android Wear. The different approaches to extract data depend on the level of intrusion and destruction. Intrusive, Non-Intrusive and Destructive methodologies were applied and according to the results regarding forensic soundness, the non-intrusive data extraction was better than the Intrusive data extraction. The “reverse engineering” of AFLogical allowed the creation of a wearable forensic application that is able to extract data by making queries on the available content providers for the wearable device. Rooting the watch would require to unlock the bootloader in most cases and wipe the data due to android security measures concerning privacy.

The usage of wearable gadgets are prone to different attacks that can leak our information and invade our privacy. Several attacker capabilities are identified in the threat model and are addressed to raise awareness. With the coming of disposable computers, the risk of being attacked will notably increase. Bluetooth packet sniffing was accomplished both with kismet and wireshark and plain text messages of different messaging platforms were retrieved.
Contents

1 Introduction ........................................ 1
   1.1 Project Aims, Objectives and Introduction ............. 1
   1.2 Background and Literature Survey .................... 3

2 Background Theories .................................. 7
   2.1 Wearable Computing ................................ 7
      2.1.1 Introduction ................................ 7
      2.1.2 Android Wear ................................ 7
      2.1.3 Wearable Architecture ......................... 7
      2.1.4 Android Architecture ......................... 8
         2.1.4.1 Content Providers ......................... 9
      2.1.5 Android Wear Partitions ....................... 9
   2.2 Wearable Forensics ................................ 10
      2.2.1 Definition ................................ 10
      2.2.2 Forensic Soundness ........................... 10
      2.2.3 Triage ................................ 11
      2.2.4 Crime Scene ................................ 11
      2.2.5 File Signatures ............................... 12
      2.2.6 File Carving ................................ 12
   2.3 Security Analysis ................................ 13
      2.3.1 Introduction ................................ 13
      2.3.2 Security Models ............................... 13
      2.3.3 Android Security Model ....................... 14
      2.3.4 Permissions ................................. 14
      2.3.5 SmartWatch Security Mechanisms ............... 14
      2.3.6 Rooting ................................ 15
   2.4 Bluetooth sniffing ................................ 16
      2.4.1 Introduction ................................ 16
3 Main Result

3.1 Wearable Forensic Acquisition Methodology
3.1.1 Manual Acquisition
3.1.2 Non-rooted Logical Acquisition
3.1.2.1 ADB
3.1.2.2 AFL0gical Open Source Software
3.1.3 Intrusive Forensics
3.1.3.1 Sqlite data browsing
3.1.3.2 ADB partition commands
3.1.3.3 Images
3.1.3.4 Forensic Tools
3.1.4 Destructive Forensics
3.2 Android Wearable Forensic Application Development
3.2.1 Methodology
3.2.2 Architecture
3.2.3 Interface
3.2.4 Tests and Implementation
3.3 Threat Model
3.3.1 Definition
3.3.2 Assets
3.3.3 Attacker Capabilities
3.4 Interception of messages using the ubertooth sniffer

4 Conclusion

References
List of Figures

1  Wearable evolution.[1] ................................................................. 4
2  Wearable Architecture.[2] ............................................................ 8
3  Android Architecture. ................................................................. 9
4  Asus ZenWatch Partitions. ............................................................ 10
5  Fragment of a recovered PNG from the Asus ZenWatch .................... 13
6  Capture of BLE packets .............................................................. 17
7  Asus ZenWatch properties Report ................................................. 20
8  MAC Address Regular Expression and dumpsys command .................. 21
9  Map found on unallocated partition .............................................. 22
10 Process to obtain URI for step counts .......................................... 23
11 Asus Zenwatch X-Ray - MPS ....................................................... 24
12 Wearable Forensics GUI ............................................................. 25
13 Application Permissions ............................................................. 26
14 Console Output ................................................................. 26
15 Data Import in Excel .............................................................. 27
16 Ubertooth Specan ................................................................. 28
17 Spectool ................................................................. 29
18 Wireshark pcapbtbb lecture ....................................................... 30
19 Bluetooth Payload capture ....................................................... 30

List of Tables

1  Essentials for Wear - Applications to get started ............................ 5
2  Common File Signatures ............................................................ 12
3  Plain text analysis .............................................................. 30
Acknowledgements

I would like to thank my supervisor Dr. Nishanth Sastry for his guidance, support, great lectures and for giving me the opportunity to develop this project. A profound gratitude to Adrian O’Leary Higher Digital Forensic Specialist of the Metropolitan Police for his expertise, guidance and for letting me use the equipment that was necessary for my dissertation to be accomplished. Last but not least, I would like to express my sincere gratitude to God and my Family.
1 Introduction

1.1 Project Aims, Objectives and Introduction

Wearables seen in films like Dick Tracy, Star Trek and Back to the Future are not Science Fiction anymore. The arrival of fashionable technology has marked a significant era in computing where both size and cost are decreasing, whereas package flow and data collection are increasing.

Nowadays, the Internet is not only used for consuming and interchanging information but there are broadened possibilities of controlling capabilities of any tangible connected to a network node from any part of the world.

Gradually, with the herald of the Internet of Things, computers are not only shrinking but getting powerful and cheaper, objects are turning smarter and studies are being conducted based on disposable innovations.

An example of today’s mini-computers with ARM\(^1\) architecture are the Raspberry PI and the BBC Micro:bit. The former has a complete operating system and according to a review performed by the time the dissertation was in progress, in Amazon, the cost was no more than £15. The latter has an embedded software platform that works with a web based interface and is free for Year 7 children in the UK.

Watches of the current epoch allow the monitoring of wellness through the amount of steps taken daily, the possibility to obtain the heart rate, exercises, etc. Moreover, a user of smart-watch technology has the ability to attend instant notifications of vast applications in the market discussed further.

The recent release of smart devices has led to the birth of myriad electronic components that interact direct or indirectly with the cloud. For the former, with a connection which doesn’t rely on Blue-tooth like WI-FI digital devices and for the latter which we will review in detail, that interplays with Blue-tooth connections and therefore requires a hand-held equipment like a smart-phone.

The Internet of Things (IoT), IPv6 and reduced size ARM architecture processors are

\(^{1}\)Known as one of the most popular architectures in embedded systems. [3]
pushing hi-tech machinery to a new era in which ubiquitous wearables will improve our daily life in different aspects like health, fitness, home automation and also in the industry with a variety of employee wellness programs mentioned in Duval and Hashizume’s article[4].

As technology evolves, so does cybercrime and it is plausible to find digital evidence linked to an Android based smartwatch in a crime scene that could potentially contain forensic data with probative value. The project will focus on data extraction in an environment where only a watch is left. However, the scenario covers rooted and non-rooted devices for the investigation. If a smartphone is encountered with its correspondent paired device, then the forensic analysis should be carried out on the smartphone in a non-intrusive manner. A rooted smartwatch can be used by an attacker to gain access to private data belonging to the victims handheld device by pairing both appliances.

The study will target mainly on what forensic data is recoverable from an Android based SmartWatch using the Asus ZenWatch 9A76 and the Sony SmartWatch 3 which both run over Android Wear. To achieve the acquisition, we will focus on three different stages:

- Non-intrusive logical forensic analysis by examining the AFLogical OSE functionality, taking advantage of the accessibility to the source code. ADB pull commands and shell commands like `pm`, `dumpsys`, `df` and `getprop`. And finally developing an android SmartWatch application that can retrieve the information from the content provider.

- Intrusive forensic analysis gaining root access via exploit and generating several images associated to relevant partitions like cache, data and system which can be processed with FTK Imager or Autopsy. A rooted devices would typically have its data wiped when the Bootloader is unlocked. However, there are certain devices or operating systems that do not require unlocking the bootloader. The Asus ZenWatch was rooted but the data was lost.

- Destructive Forensic Chip-Off technique to acquire data directly from the NAND
1.2 Background and Literature Survey

Over the past years, people have adopted gadgets as wearables to improve their lifestyle. It is said that most wearables are interconnected with different devices through a smartphone and still can be considered a subset of the Internet of Things[5].

Smart watches are becoming more popular and embraced by society. The amount of garment component applications is increasing. According to [6], IDC\(^2\) has forecasted that the global market for electronic wearables will reach 110 million units by the end of 2016. This study is not far from the prediction of the Juniper Research that states that the expected growth will be up to 130 million wearable devices shipped by 2018. [5]

Leading companies like Apple, Google and Samsung are releasing operating systems focused on the user interface and that scale to the needs of both smartphones and smartwatches. WatchOS, Android Wear and Tizen belong to the mentioned firms respectively.

The Samsung Galaxy Gear which is an android based device was released at the end of 2013 and the Android Wear Operating System was announced on the first quarter of 2014 then further released on the last quarter of the same year[7, 8]. The timeline of wearable tech can be seen in figure 1.

At the moment, there isn’t a Top Chart for Android Wearable applications from the Google Play Store. However, the Essentials for Wear applications have been considered

\(^2\)International Data Corporation is a market research firm
due to their popularity by the time the project outline was written from the Google applications repository.

The analysis of stored data in wearable applications with a high number of downloads between one million and fifty million will be considered as part of potential information that can be relevant in a triage. Data on table 1 refers to a compilation of wearable applications from the Google Play Store that are categorized as essentials to start with; nevertheless, are subject to changes during the development of the dissertation due to the increasing popularity of software for gadgets.

---

Triage in computer forensic is to find only relevant data between a big set of information.
<table>
<thead>
<tr>
<th>Application</th>
<th>User Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android Wear</td>
<td>1,000,000 - 5,000,000</td>
</tr>
<tr>
<td>Strava Running and Cycling</td>
<td>5,000,000 - 10,000,000</td>
</tr>
<tr>
<td>Google Fit</td>
<td>10,000,000 - 50,000,000</td>
</tr>
<tr>
<td>Water Drink Reminder</td>
<td>10,000,000 - 50,000,000</td>
</tr>
<tr>
<td>Glide</td>
<td>10,000,000 - 50,000,000</td>
</tr>
<tr>
<td>Todoist</td>
<td>5,000,000 - 10,000,000</td>
</tr>
<tr>
<td>Musixmatch</td>
<td>10,000,000 - 50,000,000</td>
</tr>
</tbody>
</table>

Table 1: Essentials for Wear - Applications to get started

Faheem et al\[9\] after gaining root access on a smartphone device, use ADB shell commands, generate an image with the DD command and finally utilize a commercial tool from Cellebrite called UFED physical analyzer.

Lessard and Kessler[10] have a similar approach. They clearly state the importance of obtaining root access from a smartphone. Additionally, FTK imager is the forensic tool used to process the bit-by-bit copy.

Commercial tools like Oxygen and EnCase are used in the study carried out by Shortal and Azhar[11]. The analysis also points out that WhatsApp decryption was feasible either with the data extraction tool or with a third party tool. By the time the project outline was written, WhatsApp added an end-to-end encryption capability to secure messages.[12] This can affect current tools that provide data extraction from WhatsApp sources.

One of the most important related work is Omeleze and Venter’s research[13] that mentions two forensic techniques:

- Logical Acquisition
- Physical Acquisition

Even though the techniques are only mentioned, the study will cover the acquisitions both logically and physically in an intrusive, non-intrusive and destructive manner.
Consequently, the determination of which methodology produces digital evidence that is forensically-sound\(^4\) will be accountable.

In a study about Building Wearables for Geology\(^{[15]}\), the authors talk about wearable system architecture, types of wearable operating systems and the comparison between Internet of Things and Wearable OS. This discussion is important for the awareness of the current challenges and wearable operating system influence.

Parikh et al\(^{[16]}\) extend the analysis on Android Smart Watch Artifacts with Bluetooth packet analysis which is considered relevant to the study.

Furthermore, Baggili et al\(^{[17]}\) have conducted a research study in which they accomplish a forensically intrusive method for the data acquisition of a smartwatch and which is rooted prior the extraction. These scenarios are unlikely to be found in a conventional crime scene; however, they document the process of gaining root access and later retrieve data with ADB pull commands.

It is worth mentioning that all the related work on forensic acquisition has based the data extraction on devices that have been previously rooted.

According to the related work, the study has determined to follow a path which would allow us to obtain all the data with probative value regardless to privative or open source tools.

The first step will be the acquisition of logical data in a non-intrusive fashion. The data extracted from the watch should be documented and analyzed.

The next step is to root the watch. Despite related work about rooting devices, there are several communities like xdadevelopers\(^{[18]}\) where this information is updated and available. Android based brands are know to have exploits which will be used to gain root access on the Asus ZenWatch.

Later, Android Data extraction is possible by using ADB which is a command line tool that allows a communication with the device via usb. Sqlite Browser would enable us to search through the different tables, create views and execute SQL Queries.

---

\(^4\)Forensic Soundness is “The application of a transparent digital forensic process that preserves the original meaning of the data for production in a court of law.”\(^{[14]}\)
2 Background Theories

2.1 Wearable Computing

2.1.1 Introduction

According to figure 1, wearables originated in 1975 with the appearance of the pulsar calculator watch. However, in the mid 90’s, energy powered always-on devices were introduced. [16]. Body-worn computers provide mechanisms to interact with data with less distraction or obstruction. For example, a voice command can be dictated and a message can be sent while the user is cycling, running or driving. This task would be unfeasible with a conventional handheld device.

2.1.2 Android Wear

Android Wear is a flavor of Android specific for wearable gadgets. The operating system connects with the Google services and uses the smart personal assistant called Google Now to make voice commands and handle notifications with cards.

2.1.3 Wearable Architecture

As mentioned before, Android Wear utilizes Google Now which requires an Internet connection. Although the smaller device typically connects to a Smartphone via Bluetooth, it can also connect directly to the cloud with WI-FI. The paired devices interact through three application programming interfaces seen in figure 2.

The API of interest is the Data API which is in charge of synchronising the data between the paired gadgets. Different tables are copied from the phone to the watch and stored on the data partition. This data is only accessible to rooted smartwatches or non-rooted devices that can interact with the different content providers discussed further.
2.1.4 Android Architecture

According to a post in http://stackoverflow.com regarding Android wear architecture; Raffaelle Garofolo, author of the books Windows Presentation Foundation and the Model View ViewModel Pattern and Applied WPF 4 in Context, states that the difference between the architectures of Android Wear and Android reside in the Activity. He claims that the android wear activity contains a different set of events and methods which are plugged into the operating system that are triggered when time is changed, notifications are delivered, luminosity is changed, etc.

As shown in figure 3 the following layers can be identified:

- Applications
- Application Framework
- Libraries
- Android Runtime
- Linux Kernel

A forensic wearable application will be developed, its GUI will reside in the Applications layer. It will use content providers from the Application Framework to access data with SQLite libraries in the Library tier. We will focus on these layers.
2.1.4.1 Content Providers  The application Framework tier contains the content providers that are interfaces which allow the read or write access to certain sqlite databases or files, depending if the application deliberately shares data. This allows the centralization of content in a single place so that different applications can consume freely the data.[19]

Typical open interfaces for android wear are the Contacts and the CallLogs. Calendar, SMS and Browser content providers aren’t available or have no data for wearable devices. Whereas handheld device populate the tables and the data is queryable.

2.1.5 Android Wear Partitions

There are certain number of partitions depending on the brand of the electronic garment (Refer to figure 4). The most important ones are the system partition, the user data partition, cache partition and the sdcard partition.

In the system’s partition the whole android operating system is saved. The user data partition is the most important partition and contains all the persistent data which is of great relevance for data collection. The cache partition stores frequently accessed data but is wiped constantly. And the sdcard partition can contain potential evidence or files like media, documents, etc or records that where created by third party software.
2.2 Wearable Forensics

2.2.1 Definition

Wearable Forensics is a new branch of Digital Forensics that refers to the preservation and forensic analysis of wearable devices by using techniques to investigate any relevant evidence that has to be presented before a forum.

2.2.2 Forensic Soundness

Is a term used in the Forensic realm to describe immutable states of the examined device. For wearable devices, it is almost impossible to not alter states leading to a challenging methodology to acquire data with probative value. Still with a manual data acquisition, the states can be modified in the moment the device is manually searched for evidence. Non-intrusive methodologies require a state change to enable developer mode. Physical data acquisition would be the best alternative to classify the acquisition as forensically sound. However, it can lead to the destruction of the gadget.

To achieve forensic soundness it is recommended to follow the four principles of digital evidence in the ACPO Good Practice Guide for Digital Evidence [20]. The first statement indicates that the data shouldn’t be tampered by anybody. The second principle states
that in case the original data is needed, an expert should manage the case and be able to explain all the implications. The third principle requires the documentation of the procedures so that in a future they could be reproduced. And finally the last principle denotes that the person in charge of the investigation must ensure that the rest of the principles are complied.

2.2.3 Triage

According to Overill et al.[21], the London Metropolitan Police Service Digital Electronics and Forensics Service receives more than 38,000 digital devices per year. This value is outdated and with the herald of wearable devices and the growth of storage capacity, the amount of devices and workload will increase considerably. To optimize the forensic analysis of myriad wearables, a Triage must be performed which implies a complex process to accomplish sifting. For the case of wearable devices, if the paired mobile device is present, the digital garment could be dismissed, unless the investigators require data regarding heart rates, sleeping habits or other information related to specific features of smartwatches. Another consideration is to filter android data partitions and only create images of the relevant partitions discussed in section 2.1.5

2.2.4 Crime Scene

In order to preserve data and manage the best evidence, the wearable devices must be seized appropriately. The guides are based on the ACPO publication [20], but customized for android wearables.

1. Create a safe environment and restrict physical access to the wearable device encountered.

2. Photograph the gadget and document the state of the device and all the notification found on the smartwatch.

3. It is important to isolate the device from any networks including bluetooth connections. This can be achieved by introducing the device to a Faraday cage.
4. To reduce any delays on examinations, seize all relevant accessories related to the wearable device like paired smartphone, manuals, etc.

5. Submit items as soon as possible due to the possibility of data loss generated by jobs created by the manufacturers that wipe information.

2.2.5 File Signatures

“A file signature is a unique sequence of identifying bytes written to a file’s header.” [22]

The importance of distinguishing type of files by reviewing the hexadecimal signature is fundamental for forensic analysis due to the plausibility of file corruption. When a file browse is performed, the headers must be identified.

<table>
<thead>
<tr>
<th>File Signatures</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>JPG</td>
<td>FF D8 FF E0</td>
<td>FF D8 FF E1</td>
<td>FF D8 FF E8</td>
<td></td>
</tr>
<tr>
<td>GIF</td>
<td>47 49 46 38</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PNG</td>
<td>89 50 4E 47 0D 0A 1A 0A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDF</td>
<td>25 50 44 46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3GP</td>
<td>00 00 00 14 66 74 79 70</td>
<td>00 00 00 20 66 74 79 70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOC</td>
<td>D0 CF 11 E0 A1 B1 1A E1 0D 44 4F 43</td>
<td>CF 11 E0 A1 B1 1A E1 00</td>
<td>DB A5 2D 00</td>
<td>EC A5 C1 00</td>
</tr>
</tbody>
</table>

Table 2: Common File Signatures

2.2.6 File Carving

According to Pal et al.[23], File carving is a forensic technique for recovering files without metadata. It is performed in unallocated clusters and file types are extracted by identifying file signatures.
2.3 Security Analysis

2.3.1 Introduction

Wearable devices should use platforms that ensure the security of the data. A model must be used to determine the appropriate protection against adversaries. Moreover, a threat model\(^5\) must be defined and it should identify the assets and all the attacker’s capabilities.

2.3.2 Security Models

The CIA triad is a security model designed to develop security policies that can preserve confidentiality, Integrity and Availability. Confidentiality assures that in the channel of communication between at least two parties, information doesn’t get leak. Integrity will ensure that the data transmitted will not be tampered and Availability denotes the accessibility to the service at any moment.

\(^5\)Threat modeling is a representation of all the information that could potentially affect the security of an application and its environment.[24]
2.3 Security Analysis

2.3.3 Android Security Model

Android Wear relies on its multi-layered security open platform. The Android Security Model obliges the user to pass through several checks starting with the examination of digital signatures pertaining to authors of the developed third party applications. After the control has been passed, the user must grant any permissions the application requires. If the user doesn’t feel comfortable with the permissions that are required, then she is free to not install the application. Finally each application is associated to a different system identification and runs its own process and Dalvik Machine\(^6\). This denies the possibility to interact between applications, share memory, users and resources. Only when the developer deliberately wants to share data, the application can be open to interoperability.

2.3.4 Permissions

Wearable applications sometimes require the user to grant permissions to certain components like contacts, messages, calendars, camera, media, etc. In Android 6.0 and later, the permissions are managed not in the installation but in runtime. System permissions are divided into Normal and Dangerous Permissions. For the former, the permissions are granted automatically. For the latter, the user must authorize the access for example, read or write permissions to contacts.

Protected features of wearable applications that involve dangerous permissions, must be declared in the app manifest by adding a tag denoted by \texttt{uses-permission}

2.3.5 SmartWatch Security Mechanisms

The screen protection of different brands of android based devices aren’t suggested to be used by default therefore leaving the user to disavow the functionality. This security measure could avoid an attacker to enable ADB debugging and further compromise data. If the ADB debugging option is already enabled, the attacker can push and pull files allowing him to either retrieve information with the getprop command, dumpsys, pm,

\(^{6}\)The Dalvik Machine is a secure sandbox for mobile and wearable applications
etc. The attacker could also install applications but dangerous permissions would be restricted. Finally, the malicious user can access fastboot and recovery and restore the device to factory settings.

Erasing the userdata partition when unlocking the bootloader\(^7\) is another security measure that guarantees the deletion of any existing data. However, root access gained by exploiting a flaw could bypass this measure.

### 2.3.6 Rooting

“We have the right to devices that can be opened” is one of the several statement of the repair manifesto of Ifixit.[25].

This phrase applies to wearable devices but in a security context, it isn’t the best idea to have a rooted smartwatch. Android Rooting is a procedure to gain privilege escalation and own the device.

The first step to root a device is to unlock the bootloader. This step is critical due to its risk of wiping the user’s data. Some devices have security flaws that permit the bypass of unlocking the bootloader. The next step is to flash the device with a TWRP\(^8\) custom recovery ROM that corresponds to its brand. For example, an anthias TWRP Custom ROM was used for the ZenWatch. The last step is performed in the TWRP recovery mode. A correct version of SuperSU must be installed via ADB sideload. Finally we check in an ADB shell if the SU command works properly.

If anything goes wrong with the rooting procedure, there is always the option to flash a Stock ROM so keep in mind that it is always better to have a Stock Rom before the rooting process is carried out.

---

\(^7\) The bootloader refers to code that is executed before the Android operating system is running. Depending on the manufacturer or the service provider, the bootloader can be locked or unlocked

\(^8\) TWRP is an open source community project that involves the creation of custom roms for different brands of digital devices
2.4 Bluetooth sniffig

2.4.1 Introduction

Sniffing is the procedure of eavesdropping a communication and capturing all the packets that are in the network. Bluetooth sniffing takes place in a Bluetooth network and its security relies on the linked key which is created when the devices (SmartWatch and SmartPhone) are paired. This key can be intercepted by listening to incoming and outgoing traffic.

2.4.2 Bluetooth snoop logging

Android devices can generate a Bluetooth snoop log by enabling the option with the developer menu. Once it is enabled, the file btsnoop_hci.log is created in the sdcard folder and all the bluetooth host controller interface traffic from the wearable device is captured. To stop the capture, the Bluetooth snoop logging option must be disabled. This file can be considered critical because it can disclose unencrypted conversations of Whatsapp, SMS, Glide, Hangouts and other messaging applications.

2.4.3 Project Ubertooth

Cheap Passive promiscuous Bluetooth sniffing is possible with the open source project Ubertooth, which is a wireless development platform used for bluetooth experimentation. The installation requires certain libraries like libusb, cmake, libbluetooth pkg-config libpcap python-numpy python-pyside and python-qt4. After these prerequisites are installed, the library libbtbb must be built. The installation was accomplished in a Kali Linux Virtual Machine, following the guide https://github.com/greatscottgadgets/ubertooth/wiki. Finally the ubertooth tools must be installed as stated in the URL provided above.
2.4 Bluetooth snifing

2.4.3.1 Wireshark  The current version of Wireshark is 2.0.1. It has included certain plugins (BTBB and BR/EDR) that enable the lecture of Kismet\(^9\) packets. However, Bluetooth Low Energy packets can be captured directly in Wireshark by using the pipe functionality together with the `ubertooth-btle` command:

1. In a terminal type `mkfifo /tmp/pipe`
2. Open Wireshark and in capture-options Press the button Manage Interfaces
3. In the pipe tab add an entry with the value `/tmp/pipe`, save and press start
4. In a terminal type `ubertooth-btle -f -c /tmp/pipe`

![Figure 6: Capture of BLE packets](image_url)

\(^9\)Kismet is not only an IDS for WLAN's but it also works as a packet sniffer and a network detector in a terminal environment.
3 Main Result

3.1 Wearable Forensic Acquisition Methodology

At the beginning of the dissertation, three acquisition methods were defined based on the literature review. Nevertheless, a Manual acquisition was not considered but documented further on. It was important to consider any methodology that would contribute with more information for the data collection in a forensically sound manner. Rooting the smartwatch was an option for the Intrusive Forensic Methodology but was questioned due to its nature of seeking a security breach to gain root access. Additionally, the rooting process would lead to the deletion of data and wipe the forensic evidence.

3.1.1 Manual Acquisition

A Manual Acquisition is the documentation of the state of the wearable device with the possibility to browse stack notifications that were stored on the device, hangout and glide conversations, the agenda, contacts, steps by date, Google information about the city location and weather. This technique is tedious when it comes to document the findings and also the information is incomplete. The stack notifications where the most important findings due to the unavailability of this information sought both in files and databases in a rooted environment.

3.1.2 Non-rooted Logical Acquisition

This technique is used to obtain the most information of the device with ADB commands and Content Providers. For the former, the ADB program had to be installed and enabled on the SmartWatch. For the latter, a wearable application was developed to access databases through provider URI’s and by using cursors the raw information would be captured and consequently queried. The software created was able to extract relevant records like contacts, call logs, step records, profile and photos. The development is discussed in section 3.2
3.1.2.1 ADB  Is a program that connects a gadget to the computer via USB. It allows to cast a set of commands on the wearable. The commands used were the `adb forward` command to forward a socket connection. This instruction was useful for pairing emulated wear devices with the correspondent handheld device which was either virtual or real. The shell order was also utilized to create a socket connection with NC\textsuperscript{10} and send files directly from the digital garment to a computer without the ADB pull command. This required the installation of BusyBox\textsuperscript{11} in the wearable. The `adb shell` command spawns a remote shell to interact with the device. Adb install is capable of installing a package with the extension .apk. Adb sideload is used to send files from the PC to a watch that is in a recovery mode state.

3.1.2.1.1 RO Properties  Inside the ADB shell it was possible to retrieve relevant data with the getprop command. A shell script was generated and can be seen in Appendix A and the output in figure 7. For an evidence case we should document the following data:

- `ro.product.device`
- `ro.product.manufacturer`
- `ro.product.model`
- `persist.sys.asus.btmac` (*command may vary according to brand of watch.*)
- `persist.sys.timezone`
- `ro.boot.serialno`
- `ro.build.id`

\textsuperscript{10}NC stands for Netcat and is considered the digital swiss army knife.
\textsuperscript{11}BusyBox is the swiss army knife of embedded Linux
3.1.2.1.2 PM  In a triage the command `pm list packages` could be executed to obtain a list of packages. The command `pm dump all — grep dataDir` outputs the path of the SQLite databases.

3.1.2.1.3 DUMPSYS  This command outputs information about system services. It obtains the most relevant information that allows us to identify the owner of the device by the email account and telephone number. It also outputs the MAC Address of the Bluetooth interface belonging to the paired smartwatch.

In a crime scene where only a watch is left, this information is of vital importance. It was found that the MAC Address of the Wireless device of a handheld gadget is sequential to its Bluetooth MAC Address.

The following regular expression returns a valid MAC Addresses:

```
grep -o -E '([[:xdigit:]][1,2]):{6}([[:xdigit:]][1,2])'
```

By executing the next command, different MAC Addresses are listed (figure 8):

```
dumpsys | grep -o -E '([[:xdigit:]][1,2]):{6}([[:xdigit:]][1,2])'
```

Once the MAC Address belonging to the smartphone is identified, the following command could give us more information:

```
dumpsys | grep -i -B 10 XX:XX:XX:XX:XX:XX
```
3.1 Wearable Forensic Acquisition Methodology

Figure 8: MAC Address Regular Expression and dumpsys command

*Please replace XX:XX:XX:XX:XX:XX with a valid MAC Address*

With the above command, the name of the service is granted. The dump of service is called `bluetooth_manager` and the command to obtain the information traduces to:

```bash
dumpsys bluetooth_manager
```

Another utility of the dumpsys command is to obtain the content providers by executing the following code:

```bash
dumpsys | grep Provider
```

### 3.1.2.2 AFLogical Open Source Software

AFLogical OSE is an open source mobile application for forensic Analysis that allows an examiner to extract CallLog Calls, Contacts Phones, MMS messages, MMSParts, and SMS messages from Android devices. It was compatible with wearable devices but didn’t extract all the data expected.

How does the application obtain all the information it claims to extract? This question leads to a meticulous analysis of the open source software available on github: [https://github.com/nowsecure/android-forensics](https://github.com/nowsecure/android-forensics) And furthermore, how can we personalize the code to work with wearable devices? The answers were implicit in the source code and the only way to understand the functionality was to create a developer environment with Android Studio and execute a step by step debug.

Debugging led to the discovery of the Java file DataCaptureDetailProvider which is a class that gathers information from the device using an Android SDK library called Build.java. Refer to the Appendix B for more information.
Eventually, the Content Providers would arise and the panorama would be clearer. The class `CSVForensicsProvider` had a function `process` which would take an URI as one of its parameters and execute a query based on the URI input. The result would be stored in a Cursor and later the column names would be extracted. Similar functions for obtaining data from the content providers were also present like `queryContent` and `queryForNumberOfRecords`. For more information, refer to Appendix C.

After an exhaustive analysis on the source code, an android wearable forensic application was developed because the code was abstract and there was information that was missing.

### 3.1.3 Intrusive Forensics

This methodology is based on Android Rooting and is categorized as the worst forensic sound technique. It would be difficult to convince a Jury that the data extracted has probative value. Several states of the smart watch have been changed and possibly all the evidence was wiped. Despite being able to create images of partitions and Analyzing them with FTK Imager, this methodology would introduce reasonable doubt.

Rooting a smartwatch was useful to obtain the content provider URI for diverse SQLite tables. Without gaining root access this wouldn’t have been possible. Assuming that an exploit would be found for bypassing the data wipe, the intrusive methodology would enable a quick image creation and analysis. Another reason for rooting is to analyze a replica of the original when a brand hasn’t been forensically examined before; this way we could retrieve the name of the tables and craft the content provider URI’s.

![Figure 9: Map found on unallocated partition](image-url)
3.1.3.1 SQLite data browsing  SQLiteBrowser was used to read the SQLite databases. The process of data browsing was tedious and there wasn’t any relevant data. However, the tool was useful to obtain table names. This would allow to configure the correct content provider URI.

With the content provider retrieve command stated in 3.1.2.1.3, The Asus wellness content provider can be known as `com.asus.wellness`. But the table name is still unknown; the database path can be recovered with the command stated in 3.1.2.1.2 and later browsed. Once the table name is known, the URI is crafted for example:

```
URI = com.asus.wellness/step_count
```

![Figure 10: Process to obtain URI for step counts](image)

3.1.3.2 ADB partition commands  To view the partitions, first run the following command:

```
ls -laR | grep by-name
```

Then change the directory to the path found in the previous command and list the content for example:

```
cd /dev/block/platform/msm_sdcc.1/by-name && ls -la
```
3.1.3.3 Images  The creation of images is accomplished with the `dd` command. The images can be bigger than the sdcard partition therefore, a solution must be used to solve problems related with storage space. As mentioned before, Busybox was installed and the following commands were executed:

```
adb forward tcp:{5555} tcp:5555
```

```
dd if=/dev/block/mmcblk0p33 | nc -l -p 5555
```

```
nc 127.0.0.1 5555 > image.dd
```

3.1.3.4 Forensic Tools  When the images are ready, they must be processed by tools that allow the image mounting and file browsing. Two tools were used FTK Imager and Autopsy. The former is a windows tool that was available at the KCL Computer Lab. The latter is an open source linux tool with a web interface. FTK was more user friendly.

3.1.4 Destructive Forensics

The Physical Data Acquisition was considered as a technique. Nevertheless, the methodology wasn’t able to be accomplished due to the lack of documentation about the Datasheets corresponding to the MCP which is a multi chip package that packs 512 MB LPDDR2 plus 4 GB eMMC used by the Asus Zenwatch.

![Asus Zenwatch X-Ray - MPS](image.jpg)

Figure 11: Asus Zenwatch X-Ray - MPS
3.2 Android Wearable Forensic Application Development

A wearable application was developed from scratch using Android Studio, the requirement was to develop an application that can connect to the content providers and download contacts, profiles, call logs, emails, photos and phones and store the information in different files separated by pipes. The files must be stored in the sdcard/forensics folder and each time the application desires to download the data, the content of the folder must be deleted. The file must be readable in excel.

3.2.1 Methodology

Due to the nature of the project, the Cowboy coding methodology would be implemented. This enables the coder to do whatever he feels correct, work alone with minimal discipline but high experimentation. This is suitable for a students.

3.2.2 Architecture

The Architecture is MVP that derives from MVC; it stands for Model View Presenter and is common on Android Development. It allows the decouple of the presentation layer with the business logic.

3.2.3 Interface

The application uses the console to output the data encountered. The application is developed for a square smartwatch with a single button and a label.

![Wearable Forensics GUI](image)

Figure 12: Wearable Forensics GUI
The first time a user opens the application, it requests 5 permissions: Read SMS, Read Calendar, Read Call Log, Read Contacts and Write External Storage. This security is based on Android’s runtime permissions.

![Application Permissions](image)

**Figure 13: Application Permissions**

### 3.2.4 Tests and Implementation

The console returns the number of records found for each table accessed through the Content Provider (SMS, Calendar, Call Log, Contacts).

![Console Output](image)

**Figure 14: Console Output**

The files are created in the forensics folder and it is possible to import the file to
Excel with the Excel’s wizard. The delimiter is the pipe |.

![Excel Data Import](image)

Figure 15: Data Import in Excel

3.3 Threat Model

3.3.1 Definition

According to Snyder et al [26], A Threat model is a detailed description of all the threats that can affect a system. It’s the analysis of factors that can affect the CIA triad.

3.3.2 Assets

Personal Data, Accounts, Messaging

3.3.3 Attacker Capabilities


1. Simple Eavesdropping can lead to the disclosure of information affecting the confidentiality security principle.

2. Packet injection is plausible

3. Temporary Physical Access can cause data leakage by using Non-rooted logical Acquisition techniques.
4. An attacker when able to gain physical access for a longer period could root the device and install malware or Surveillance software.

5. Bluetooth snoop logging could be deliberately enabled for further revision and disclosure of plain text messages.

3.4 Interception of messages using the ubertooth sniffer

The following command in Linux generates a spectrum analyzer graph:

```
ubertooth-specan-ui
```

![Figure 16: Ubertooth Specan](image)

Moreover, there is a tool that creates other graphs related to spectrum analysis (See figure 17) the command is as follows:

```
spectools-gtk
```

In Kismet it is possible to generate a pcapbtbb file that is readable with Wireshark. Nevertheless, only the LAP\(^\text{12}\) of the Bluetooth MAC Address is readable. The payload doesn’t reveal any relevant information. Refer to figure 18

\(^\text{12}\)LAP is the lower address part
Finally, the file btsnoophci.log was analyzed with wireshark and revealed plain text messages. The table 3 below is regarding personalized messages sent from the device and captured as plain text with the btsnoophci.log. An example of a packet capture is seen in figure 19.

The plain text is revealed by searching for packet bytes that contain a string of any predefined messages defined in the table 3 above.
3.4 Interception of messages using the ubertooth sniffer

Figure 18: Wireshark pcapbtbb lecture

Table 3: Plain text analysis

```
<table>
<thead>
<tr>
<th>Messaging</th>
<th>On my way home</th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHATSAPP</td>
<td>On my way home</td>
<td>OK</td>
</tr>
<tr>
<td>SMS</td>
<td>Leaving now</td>
<td>OK</td>
</tr>
<tr>
<td>HANGOUTS</td>
<td>Give me a moment</td>
<td>OK</td>
</tr>
</tbody>
</table>
```

Figure 19: Bluetooth Payload capture
4 Conclusion

In an environment where only a watch is left, to assure forensic soundness, the Non-rooted Logical data acquisition methodology must be applied. According to the 2nd principle of digital evidence it would be plausible to enable developer mode, enable ADB debugging and install the wearable forensic application on the digital watch as long as the executer is competent to do the job and the compliance of the 3rd principle takes place by documenting each step.

To obtain more information, it is true that the Intrusive Forensics methodology would help when rooting is applied to a device that is a replica model of the original when such device hasn’t been analyzed before in history.

It is concluded that unlike other studies of forensic analysis of wearables that restrict their scope to rooted devices, this dissertation uses another approach with a more realistic scenario, software included and useful findings.

In summary, using a smartwatch increases the risk of being attacked. Confidentiality, Integrity and Availability could be compromised.

As future work, the open source code could be improved to a more complex system that can automatize script execution, partition recovery, rooting assistance, database management and reporting. Extensible to different brands of smartwatches. Another desirable research would be how to transmit the enabled btisnoop_hci.log from the watch through messaging services. This could generate a surveillance software based on wearable devices.
References


[8] “Motorola, LG announce upcoming Android Wear smartwatches — The Verge.”


A Bash Script (Properties of SmartWatch)

!/bin/bash
adb shell "
echo -n device: `getprop ro.product.device`
echo -n manufacturer: `getprop ro.product.manufacturer`
echo -n model: `getprop ro.product.model`
echo -n MAC: `getprop persist.sys.asus.btmac`
echo -n timezone: `getprop persist.sys.timezone`
echo -n serialno: `getprop ro.boot.serialno`
echo -n id: `getprop ro.build.id`
"
B AFL-OSE (DataCaptureDetailProvider.java)

/*
   This file is part of AFL-OSE (AFLogical – Open Source Edition).

   AFL-OSE is a framework for the forensic logical extraction of data from Android devices.

   Copyright 2011 viaForensics LLC

   AFL-OSE is available under the terms of multiple licenses.

   For academic, research, and experimental purposes, AFL-OSE is free software: you can redistribute it and / or modify it under
   the terms of the GNU General Public License as published by the Free Software Foundation, version 3. The source code must be made
   available and this license must be retained. It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
   even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License at http://www.gnu.org/licenses/
   for more details.

   For any other purposes, this file may not be used except under the terms of a commercial license granted from viaForensics. For
   commercial license details, contact viaForensics at http://viaforensics.com/contact-us/.
*/

package com.viaforensics.android;
public class DataCaptureDetailProvider {

    public static void createDetailsFile(final Context ctx, File forensicsDir, String timeStampAsString, PackageManager packageManager) {
        try {
            TelephonyManager telephonyManager = (TelephonyManager) ctx.getSystemService(Context.TELEPHONY_SERVICE);
            BufferedWriter writer = new BufferedWriter(new FileWriter(new File(forensicsDir, "info.xml")));
            writer.write("<android-forensics>
" + timeStampAsString + "</date-time>" + time StampAsString + "</date-time>" + telephonyManager.getSubscriberId() + "</IMSI>" + telephonyManager.getDeviceId() + "</IMEI-MEID>" + telephonyManager.getDeviceId() + "</IMEI-MEID>");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
// indicates the device phone type (CDMA, GSM, None, SIP)
writer.write("<phone-type>") + telephonyManager.getPhoneType();

// Returns the MDN / MSISDN for the primary line.
writer.write("<MSISDN-MDN>") + telephonyManager.getLine1Number();

// Returns the ICCID
writer.write("<ICCID>") + telephonyManager.getSimSerialNumber();

writer.write("<build>");

writer.write("t<version.release>") + Build.VERSION.RELEASE + 
writer.write("t<version.sdk>") + Build.VERSION.SDK + 
writer.write("t<board>") + Build.BOARD + "</board>
writer.write("t<brand>") + Build.BRAND + "</brand>
writer.write("t<device>") + Build.DEVICE + "</device>
writer.write("t<display>") + Build.DISPLAY + "</display>
writer.write("t<fingerprint>") + Build.FINGERPRINT + 
writer.write("t<host>") + Build.HOST + "</host>
writer.write("t<id>") + Build.ID + "</id>");

writer.write("t<model>") + Build.MODEL + "</model>
writer.write("t<product>") + Build.PRODUCT + "</product>
writer.write("t<tags>") + Build.TAGS + "</tags>
writer.write("t<time>") + Build.TIME + "</time>
writer.write("t<type>") + Build.TYPE + "</type>
writer.write("t<user>") + Build.USER + "</user>

writer.write("</build>");
writer.write("<applications>
");
List<ApplicationInfo> apps = packageManager.getInstalledApplications(PackageManager.GET_UNINSTALLED_PACKAGES);
for (ApplicationInfo appInfo : apps) {
    writer.write("t<app>
");
    writer.write("t\t<label>" + appInfo.label + "\n");
    writer.write("t\t<className>" + appInfo.className + "\n");
    writer.write("t\t<dataDir>" + appInfo.dataDir + "\n");
    writer.write("t\t<descriptionRes>" + appInfo.descriptionRes + "\n");
    writer.write("t\t<flags>" + appInfo.flags + "\n");
    writer.write("t\t<manageSpaceActivityName>" + appInfo.manageSpaceActivityName + "\n");
    writer.write("t\t<name>" + appInfo.name + "\n");
    writer.write("t\t<packageName>" + appInfo.packageName + "\n");
    writer.write("t\t<permission>" + appInfo.permission + "\n");
    writer.write("t\t<processName>" + appInfo.processName + "\n");
    writer.write("t\t<publicSourceDir>" + appInfo.publicSourceDir + "\n");
    writer.write("t\t<sourceDir>" + appInfo.sourceDir + "\n");
    writer.write("t\t<taskAffinity>" + appInfo.taskAffinity + "\n");
    writer.write("t\t<uid>" + appInfo.uid + "\n");
    writer.write("t\t<enabled>" + appInfo.enabled + "\n");
    writer.write("t\t<description>" + appInfo.loadDescription(packageManager) + "\n");
    try {
        PackageInfo pkgInfo = packageManager.getPackageInfo(appInfo.packageName, PackageManager.GET_META_DATA);
        writer.write("t\t\n");
        writer.write("t\t\n");
        writer.write("t\t<versionCode>" + pkgInfo.versionCode + "\n");
        writer.write("t\t<versionName>" + pkgInfo.versionName + "\n");
        writer.write("t\t</packageinfo>\n");
    }
    catch (NameNotFoundException e) {}
}
writer.write("t</app>
");
writer.write("<applications>");
writer.write("<android-forensics>");
writer.close();
}
catch (IOException e) {
    e.printStackTrace();
}
}
C AFL-OSE (CSVForensicsProvider.java)

/*
 * This file is part of AFL-OSE (AFLogical – Open Source Edition).

AFL-OSE is a framework for the forensic logical extraction of data from Android devices.

Copyright 2011 viaForensics LLC

AFL-OSE is available under the terms of multiple licenses.

For academic, research, and experimental purposes, AFL-OSE is free software: you can redistribute it and / or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3. The source code must be made available and this license must be retained. It is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License at http://www.gnu.org/licenses/ for more details.

For any other purposes, this file may not be used except under the terms of a commercial license granted from viaForensics. For commercial license details, contact viaForensics at http://viaforensics.com/contact-us/.

*/
package com.viaforensics.android.providers;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Locale;
import java.util.TimeZone;

import android.content.ContentResolver;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.text.format.DateFormat;
import android.util.Log;

import com.viaforensics.android.ForensicsException;
import com.viaforensics.android.logs.DebugLogger;

public class CSVForensicsProvider extends ForensicsProvider {

    private static final String TAG = "AndroidForensics";

    public CSVForensicsProvider(String displayName, Uri uri) {
        super(displayName, uri);
    }
}
public void process(Context context, File forensicsDir) throws ForensicsException {
    ContentResolver resolver = context.getContentResolver();
    DebugLogger.d(TAG, "processing " + displayName);
    try {
        Cursor idsOnlyCursor = null;
        try {
            idsOnlyCursor = resolver.query(uri, new String[] {"id"}, null, null, "id ASC");
            if (idsOnlyCursor != null) {

                // consider opening csv here for all writing (normal and chunks)
                BufferedWriter writer = new BufferedWriter(new FileWriter(new File(forensicsDir, displayName + "\.csv")));

                try {
                    final int numRecords = idsOnlyCursor.getCount();
                    {
                        queryContent(resolver, writer, true, null);
                    }
                } finally {
                    if (writer != null) {
                        try {
                            writer.close();
                        } catch (IOException ex) {

                        }
                    }
                }
            } else {
                idsOnlyCursor = resolver.query(uri, null, null, null, null);
                if (idsOnlyCursor != null) {

                    // consider opening csv here for all writing (normal and chunks)
                    BufferedWriter writer = new BufferedWriter(new FileWriter(new File(forensicsDir, displayName + "\.csv")));

                    try {
                        final int numRecords = idsOnlyCursor.getCount();
                        {
                            queryContent(resolver, writer, true, null);
                        }
                    } finally {
                        if (writer != null) {
                            try {
                                writer.close();
                            } catch (IOException ex) {

                            }
                        }
                    }
                }
            }
        } catch (IOException ex) {

        }
    }
}
String[] cols = idsOnlyCursor.getColumnNames();

throw new ForensicsException(
    "No 'id' column found."
) else {
    Log.w(TAG, "Unable to find data for " + displayName);
}

} finally {
    if (idsOnlyCursor != null) {
        idsOnlyCursor.close();
    }
}

} catch (Exception ex) {
    Log.e(TAG, "Unexpected error in (" + displayName + ")
    throw new ForensicsException(ex);
}

postProcess();

}

public int queryForNumberOfRecords(ContentResolver resolver) throws ForensicsException {
    Cursor cursor = null;
    try {
        cursor = resolver.query(this.uri, new String[] {"_id"})
        if (cursor != null) {
            return cursor.getCount();
        }
    } finally {
        if (cursor != null) {
            cursor.close();
        }
    }
}
```java
private void queryContent(ContentResolver resolver, final BufferedWriter writer, final boolean writeColumnNamesRow, final String selection) throws Exception {
    Cursor cursor = null;
    try {
        cursor = resolver.query(this.uri, getProviderProjection(), selection, null, null);
        if (cursor != null) {
            int numColumns = cursor.getColumnCount();
            if (numColumns > 0) {

                if (writeColumnNamesRow) {
                    if (cursor.moveToFirst()) {
                        do {
                            this.writeValue(cursor, numColumns);
                        } while (cursor.moveToNext());
                    }
                }
            }
        } else {
            Log.w(TAG, "Invalid cursor detected. skipping.");
        }
    } finally {
        if (cursor != null) {
            cursor.close();
        }
    }
}
```
protected String[] getProviderProjection() {
    return null;
}

protected void postProcess() {
    DebugLogger.d(TAG, "completed " + displayName);
}

protected void writeColumnNames(BufferedWriter writer, Cursor cursor, int numColumns)
throws java.io.IOException {
    String[] columnNames = cursor.getColumnNames();
    Log.d(this.getClass().getName(), displayName + " number of column headers = " + columnNames.length + " / " + numColumns);
    for (int columnIndex = 0; columnIndex < numColumns; columnIndex++) {
        String formattedHeaderValue = this.formatStringForCSV(columnNames[columnIndex]);
        Log.d(this.getClass().getName(), displayName + " column headers columnIndex = " + columnIndex + " ( " + formattedHeaderValue + " ) ");
        writer.write(formattedHeaderValue);
        if (!isNotLastColumn(numColumns, columnIndex)) writer.write(COMMA_SEP);
    }
    writer.newLine();
}

private boolean isNotLastColumn(int numColumns, int columnIndex) {
    return columnIndex != numColumns - 1;
}
protected void writeColumnValues(BufferWriter writer, Cursor cursor, int numColumns)
    throws Exception {
    for (int columnIndex = 0; columnIndex < numColumns; columnIndex++) {
        writeColumnValue(writer, cursor, columnIndex);
        if (isNotLastColumn(numColumns, columnIndex))
            writer.write(COMMA_SEP);
    }
    writer.newLine();
onAllColumnValuesWrote(cursor, numColumns);
}

protected void onAllColumnValuesWrote(Cursor cursor, int numColumns) throws IOException {
    
protected void writeColumnValue(BufferWriter writer, Cursor cursor, int columnIndex)
    throws Exception {
    try {
        String value = cursor.getString(columnIndex);
        if (value != null) {
            writer.write(this.formatStringForCSV(value));
        }
    }
    catch (Exception e) {
        Log.d(TAG, "Error pulling String for index: " + columnIndex);
        throw e;
    }
}

protected void writeDateValue(BufferWriter writer, Cursor cursor, int columnIndex)
    throws IOException {
    String value = cursor.getString(columnIndex);
    String formattedDate = formatDate(value);
    writer.write(this.formatStringForCSV(formattedDate));
}
public String formatDate(String value) {
    if (!isValidDateValue(value)) {
        return "";
    }
    GregorianCalendar cal = new GregorianCalendar(TimeZone.getDefault(), Locale.getDefault());
    cal.setTime(new Date(Long.valueOf(value)));
    return (String) DateFormat.format("MMMM dd, yyyy hh:mm a", cal);
}

private boolean isValidDateValue(String value) {
    return value != null && !"0".equals(value);
}

protected String formatStringForCSV(String stringToFormat) {
    return CSVCharEscapeWrapper.safeEscape(stringToFormat);
}
D WearableForensics

I verify that I am the sole author of the programs contained in this folder, except where explicitly stated to the contrary.

Felix Santiago Anda Basabe, 26/08/2016

D.1 AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.felix.wearableforensics">
    <uses-feature android:name="android.hardware.type.watch" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.DeviceDefault">
        <activity
            android:name=".ForensicActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```
<manifest>
    <uses-sdk android:minSdkVersion="3"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.READ_CALL_LOG"/>
    <uses-permission android:name="android.permission.READ_CALENDAR"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
</manifest>

D.2 activity_wear.xml

<?xml version="1.0" encoding="utf-8"?>
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/watch_view_stub"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:rectLayout="@layout/rect_activity_wear"
    app:roundLayout="@layout/round_activity_wear"
    tools:context="com.example.felix.wearableforensics.ForensicActivity"
    tools:deviceIds="wear"/>

D.3 rect_activity_wear.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
android:orientation="vertical"
tools:context="com.example.felix.wearableforensics.ForensicActivity"
tools:deviceIds="wear_square">

<TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txt_title" />

<Button
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Capture"
    android:id="@+id/btn_capture"
    android:layout_gravity="center_horizontal"
    android:onClick="btnCapture" />
</LinearLayout>

D.4 ForensicActivity.java

package com.example.felix.wearableforensics;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.provider.CallLog;
import android.provider.ContactsContract;
import android.support.wearable.view.WatchViewStub;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

/**
 * Created by felix on 26/08/2016.
 */
public class ForensicActivity extends Activity implements ForensicView {

    private TextView mTextView;
    private Button bCapture;
    private ForensicPresenter presenter;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_wear);

        final WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub);
        stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener()
        {
            @Override
            public void onLayoutInflated(WatchViewStub stub)
            {
                mTextView = (TextView) stub.findViewById(R.id.text);
                bCapture = (Button) stub.findViewById(R.id.btn_capture);
            }
        });

        // Other methods and code...
    }
}
presenter = new ForensicPresenterImpl(this);

@Override public void showMessage(String message) {
}

public void btnCapture(View target) {

    Uri uriCallLog = CallLog.Calls.CONTENT_URI;
    Uri uriContacts = ContactsContract.Contacts.CONTENT_URI;
    Uri uriPhone = ContactsContract.CommonDataKinds.Phone.CONTENT_URI;
    Uri uriProfile = ContactsContract.Profile.CONTENT_URI;
    Uri uriEmail = ContactsContract.CommonDataKinds.Email.CONTENT_URI;
    Uri uriWellness = Uri.parse("content://com.asus.wear.wellness.provider/step_count");

    // guessing thanks to rooted device I can know the table name

    String providerContact = "contact";
    String providerPhone = "phone";
    String providerCallLog = "call_log";
    String providerProfile = "profile";
    String providerEmail = "email";
    String providerWellness = "step_count";

    presenter.deleteFiles();
presenteer.

D.5 ForensicInteractor.java

package com.example.felix.wearableforensics;

import android.app.Activity;
import android.content.Context;
import java.util.List;

/**
 * Created by felix on 26/08/2016.
 */
public interface ForensicInteractor {

    interface OnForensicFinishedListener {

        void onSuccess();
    }

    boolean addPermission(Context context, Activity activity, List<String> permissionsList, String permission);
}
boolean setPermission(Context context, Activity activity);

void deleteFiles();

}
public boolean addPermission(Context context, Activity activity, List<String> permissionsList, String permission) {
    if (ActivityCompat.checkSelfPermission(context, permission) != PackageManager.PERMISSION_GRANTED) {
        permissionsList.add(permission);

        // Check for Rationale Option
        if (!ActivityCompat.shouldShowRequestPermissionRationale(activity, permission))
            return false;

    } return true;
}

@Override
public boolean setPermission(Context context, Activity activity) {

    List<String> permissionsNeeded = new ArrayList<String>();
    final List<String> permissionsList = new ArrayList<String>();

    if (!addPermission(context, activity, permissionsList, Manifest.permission.READ_CALL_LOG))
        permissionsNeeded.add("Read Call LOG");
    if (!addPermission(context, activity, permissionsList, Manifest.permission.READ_CONTACTS))
        permissionsNeeded.add("Read Contacts");
    if (!addPermission(context, activity, permissionsList, Manifest.permission.READ_SMS))
        permissionsNeeded.add("READ_SMS");
    if (!addPermission(context, activity, permissionsList, Manifest.permission.READ_CALENDAR))
        permissionsNeeded.add("READ_Calendar");
    if (!addPermission(context, activity, permissionsList, Manifest.permission.WRITE_EXTERNAL_STORAGE))
        permissionsNeeded.add("WRITE_External_Storage");

    return true;
}
if (permissionsList.size() > 0) {
    if (permissionsNeeded.size() > 0) {
        // Need Rationale
        String message = "You need to grant access to " + permissionsNeeded.get(0);
        for (int i = 1; i < permissionsNeeded.size(); i++)
            message = message + ", " + permissionsNeeded.get(i);

        ActivityCompat.requestPermissions(activity, permissionsList.toArray(new String[permissionsList.size()]), REQUEST_CODE_ASK_MULTIPLE_PERMISSIONS);

        System.out.println("Permissions required ...");
    }

    return false;
}

return true;

public void deleteFiles() {

    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {

        File directory = Environment.getExternalStoragePublicDirectory("forensics");

        if (!directory.exists()) {

directory.mkdirs();
}

for (File child : directory.listFiles()) {
    child.delete();
}

System.out.println("forensics folder emptied...");

}
}
}

D.7 ForensicPresenter.java

package com.example.felix.wearableforensics;

import android.app.Activity;
import android.content.ContentResolver;
import android.content.Context;
import android.net.Uri;

/**
 * Created by felix on 26/08/2016.
 */
public interface ForensicPresenter {

    void createPhotos(ContentResolver contentResolver);
    void deleteFiles();
    void capture(Context context, Activity activity, Uri URI, String description);
package com.example.felix.wearableforensics;

import android.app.Activity;
import android.content.ContentResolver;
import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.Environment;
import android.provider.ContactsContract;

import org.apache.commons.io.IOUtils;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;

/**
 * Created by felix on 26/08/2016.
 */
public class ForensicPresenterImpl implements ForensicPresenter {

private ForensicView forensicView;
private ForensicInteractor forensicInteractor;

public ForensicPresenterImpl(ForensicView forensicView) {
    this.forensicView = forensicView;
    this.forensicInteractor = new ForensicInteractorImpl();
}

@Override
public void createPhotos(ContentResolver contentResolver) {

    int numRecords;
    String extension = ".jpg";  //TODO put somewhere

    Cursor cursor = contentResolver.query(
        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
        new String[] { ContactsContract.CommonDataKinds.Phone.PHOTO_URI },
        //projection (I just want photo uri's)
        ContactsContract.CommonDataKinds.Phone.PHOTO_URI + " IS NOT NULL",
        null, null);

    numRecords = cursor.getCount();

    if (numRecords > 0) {

        System.out.println("Photo Records:");

        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {

            ...
File directory = Environment.getExternalStoragePublicDirectory("forensics");

if (!directory.exists()) {
    directory.mkdirs();
}

int counter = 0;
String filename = "file";

while (cursor.moveToNext()) {

    Uri imageUri = Uri.parse(cursor.getString(0));
    counter++;

    try {
        InputStream inputStream = contentResolver.openInputStream(imageUri);

        File file;
        file = new File(directory, filename + counter + extension);

        OutputStream outputStream = new FileOutputStream(file);
        IOUtils.copy(inputStream, outputStream);
        outputStream.close();
    }
    catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
try {
   摄影作品cursor.close();
} catch (IOException e) {
    e.printStackTrace();
}
System.out.println("Closing cursor photo ... ");
cursor.close();
}
else {
    System.out.println("Photo cursor is null ... ");
}

@Override
public void deleteFiles() {
}

@Override
public void capture(Context context, Activity activity, Uri URI, String description) {
    if (forensicView != null) {
        forensicView.showMessage("Processing " + description);
        forensicInteractor.setPermission(context, activity);
    }
}
public void processProvider(ContentResolver contentResolver, ForensicActivity forensicActivity, Uri URI, String description)
{

capture(forensicActivity, forensicActivity, URI, description);

int numRecords = 0;
String extension = ".txt";  //TODO put somewhere

try {

Cursor cursor = contentResolver.query(URI, null, null, null, null);

if (cursor != null) {
    numRecords = cursor.getCount();

    System.out.println(description + " Records:" + numRecords);

    //Create csv header

    String headerCSV = createStringCSV(cursor.getColumnNames());

    writeFile(headerCSV, description + extension);

    System.out.println("Processing." + description + "...");

    while (cursor.moveToNext()) {

        int size = cursor.getColumnCount();

        String[] Array = new String[size];

        for (int i = 0; i < size; i++) {
            String columnName = cursor.getColumnName(i);
            String value = cursor.getString(i);
            Array[i] = value;
        }

        // Process the data...
    }

    writeFile(headerCSV, description + extension);

    System.out.println("Processed.");

} else {
    System.out.println("No records found.");
}

}
}}
for (int i = 0; i < size; i++) {

    String property = cursor.getString(i);

    if (property != null)
        Array[i] = property;
    else
        Array[i] = "empty"; //TODO change to maybe a null

    String message = createStringCSV(Array);

    writeFile(message, description + extension);
}

System.out.println("Closing cursor " + description + "...");

cursor.close();

} else
    System.out.println(description + "cursor is null ...");
} catch (Exception ex) {
    System.out.println(ex.getMessage());
}

private void writeFile(String message, String filename) {

String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {

File directory = Environment.getExternalStoragePublicDirectory("forensics");

if (!directory.exists()) {
    directory.mkdirs();
}

File file;

try {

    file = new File(directory, filename);
    if (!file.isFile())
        file.createNewFile();

    FileOutputStream stream = new FileOutputStream(file, true);
    OutputStreamWriter writer = new OutputStreamWriter(stream);
    writer.write(message);
    writer.append("\n"); // append a new line
    writer.close();

} catch (Exception e) {

    e.printStackTrace();
    System.out.println(e.getMessage());
}
```java
public interface ForensicView {
    void showMessage(String description);
}
```

D.9 ForensicView.java

```java
package com.example.felix.wearableforensics;

/**
 * Created by felix on 26/08/2016.
 */

private String createStringCSV(String[] Array) {
    String message = "";
    boolean flag = false;

    for (String s : Array) {
        if (!flag) {
            message = s;
            flag = true;
        } else {
            message += " | " + s;
        }
    }

    return message;
}
```
}