SciDoc Publishers
Infer, Interpret & Inspire Science

International Journal of Forensic Science & Pathology (IJFP)
ISSN 2332-287X

Extraction and Forensic Analysis of Artifacts on Wearables

Research Article

Rongen J[1], Geradts Z[1,2*]

[1] Netherlands Forensic Institute, Laan van Ypenburg 6, The Hague, Netherlands.
[2] University of Amsterdam, Spui 21, Amsterdam, Netherlands.

Abstract

Wearables are an increasingly big item in mobile forensics, in large part due to the ever increasing popularity of social media. A device that falls into this category is Google Glass. A big part of the Google Glass interface is dedicated to social media functions. A side-effect of these functions is that in many cases a lot of data is generated that is interesting for forensic research. Therefore, it is imperative that more research is done into these types of devices.

This paper will focus on extracting data from Google Glass and the different possible ways of extraction. Following this, the extracted data will be analyzed for any possible artifacts that were left behind from normal use.

Keywords: Google; Glass; Mobile Forensics; Android; Glassware.

Introduction

Google Glass was one of Google's newest additions to a class of devices often referred to as "wearables" [12]. These devices can be of great value in digital research, considering they might be on a user for a long time. In many cases, they also generate a large amount of data. This is no different for Google Glass, as a large part of the user interface is dedicated to social media functions, which have a great probability of creating valuable digital traces.

A problem however is the lack of knowledge about the device. It was developed by Google X, Google's semi-secret research facility, leading to very little documentation about the inner workings of the devices. This might pose a problem when imaging the device or interpreting results from the extracted image.

Materials and Methods

This section can be divided in 2 subsections, namely the tools and methods used for the imaging of the device and the ones used for analyzing the extracted data. A good start when contemplating the extraction of data can be found in the NIST Guidelines for mobile device forensics [1].

Previous work

There has already been some research of a forensic nature into Google Glass, some results of which are used in this research. The main sources of this research include the work of Julie Desautels' "Google Glass Timeline Forensics" [2] and the "Computer and Digital Forensics Blog" [3].

Imaging

Extraction of data from the Google Glass can then again be split in two major methods, namely extraction via software-based methods and extraction via hardware-based methods.

Software

When following the software route, a couple of methods and their associated tools were used:

- Standard Linux command line tools, with addition of The Sleuth Kit and the Android Debug Bridge. These tools were used in several different methods, mainly the method of dumping the eMMC chip over the Android Debug Bridge via the dd binary.

***Corresponding Author**:
Zeno Geradts,
Netherlands Forensic Institute, Laan van Ypenburg 6, The Hague 2497GB, Netherlands.
Email: z.geradts@nfi.minvenj.nl

Received: December 30, 2016
Accepted: January 23, 2017
Published: January 25, 2017

Citation: Rongen J, Geradts Z (2017) Extraction and Forensic Analysis of Artifacts on Wearables. *Int J Forensic Sci Pathol.* 5(1), 312-318.
doi: http://dx.doi.org/10.19070/2332-287X-1700070

- The Shattered Google Glass Forensic Tool [4]: a python script made specifically for creating a logical image of a Google Glass and dumping all available information from the dumpsys tool to get information of system services.
- Android Debug Bridge Backup: a method built-in to the Android operating system to make backups of the data on a device running Android. This can be used to generate a partial logical image of a device.

Hardware

For imaging via hardware based methods, three methods were attempted:

- The "5-Wire method": this method gets its name from the fact that only five wires are needed for the extraction of data from the eMMC chip. In this method, an attempt is made to find the five necessary lines on the circuit board in an accessible place and tap into them. The necessary lines for this are:

- Command
- Clock
- VDD
- VSS
- Data0

If these lines are found and it is possible to tap into them, the next step is to put the eMMC into 1-bit bus Single Data Rate transmission mode. The data from the eMMC can then be read out using only one data line(data0) [5].

- The NFI Memory Toolkit II (MTKII): The NFI Memory Toolkit is a combination of hardware and software. The hardware makes a physical connection, generates signals and supplies power to a memory chip, while the software runs the necessary command-sets to access data in the various types of memory chips [6, 13].
- For this method, the eMMC memory will be extracted from the mainboard of the Google Glass (chip-off) and the data it contains will be extracted from it using the MTKII.
- JTAG, a method for testing of the CPU and its peripherals. It is inherent to the main goal of this method that the user has access to the memory and storage peripherals normally available to the main processing unit.

Analysis of extracted data

For the analysis of extracted data, a basic installation of Ubuntu 14.04 LTS x64 is used. A couple of tools need to be added to the default installation however, these include:

- SQLite DB Browser, a graphical browser for SQLite databases.
- The Sleuth Kit, for analyzing the structure of extracted images (e.g. used filesystem, consistency)
- Android Debug Bridge, for easy communication with the Google Glass.
- Android Backup Extractor [7], for extracting backups made with ADB.

Theory

Scope

Seeing as there was a limited time available for this research, a scope had to be established. Things that fall outside the scope of this project include:

- Searching for new exploits in the Android operating system
- Analysis of traces outside of the internal storage (e.g. cloud-based, like Google+, Google Hangouts etc.)
- Developing new tools for extraction of data from Google Glass

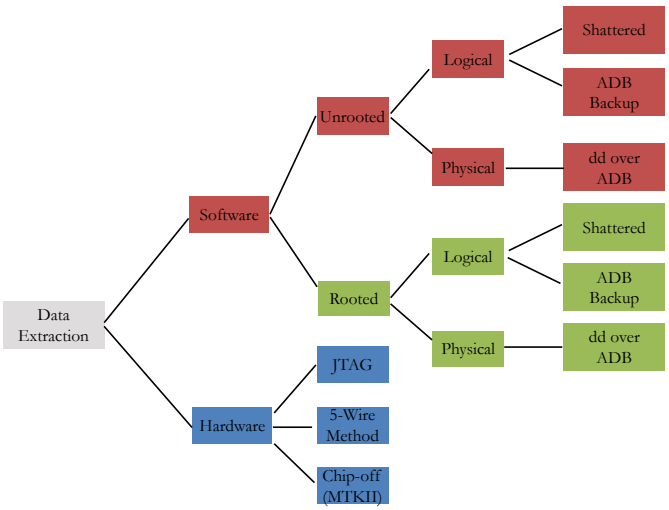
Extraction of data

Extraction of data was attempted using a collection of different methods. The methods can be divided into a number of categories, with associated subcategories. The model used for classification of data extraction methods is visible in Figure 1. Branch Diagram of Data Extraction Methods.

Logical

Shattered: The execution of this method was as simple as running the python script of which the tool consisted. The script

Figure 1. Branch Diagram of Data Extraction Methods.



then asks for a case name, which it uses to generate a directory named after the case and the date and time of the creation. The results of this method are quite extensive when it comes to information about the running system. A lot of information is available with regards to running services and the status of hardware and peripherals.

Looking at the collected data from the internal storage however, the results are quite disappointing. Most files available on the emulated internal memory card are available, as are some of the files from the /sys, /res and /acct directory. Most of the latter have low forensic value however, as they do not contain user related traces for the most part and only provide data about the running system, only useful for live investigations.

ADB Backup: This is a standardized method for creating partial backups of Android devices, executed by running ADB binary with the following switches: ***adb backup -apk -shared -all –f GlassNFI.ab***. While this would normally create an Android backup of the device named "GlassNFI.ab", there was the problem of confirming the backup via the dialog on the device.

The trouble with confirming the backup on the Google Glass itself, was that the Google Glass does not use the standard Android user interface. Rather it uses the "timeline" interface, specifically created for the Google Glass. The normal Android user interface does seem to be present in the device however, as it does popup the confirmation dialog for ADB Backup in this case. The problem however is that the Google Glass is not a touchscreen device. This means that, while it does have a touchpad, this can't be used for user input in the same way a touchscreen is used. As a result of this, there was no way to press the "Confirm" button on the dialog.

One possible solution to this is connecting the Google Glass to a bluetooth keyboard. The only problem with this method, was that Google Glass only natively supports bluetooth connections to mobile phones using the MyGlass [8] application.

Keyboard input can however also be simulated using ADB in combination with the input binary. Using this method, the backup was confirmed and created.

The created backup was then converted to a tar archive, using Android Debug Extractor [7]. After unpacking this tar archive, the contents of the backup were visible. The backup contained data from 2 directories, namely /storage/emulated/0/, which is where the emulated internal memory card is mounted and /data/data, where application data is normally stored. The data found from these two directories is unfortunately not a complete backup, as the /apps directory in the backup where the /data/data content is stored, is missing all data from the system applications. The /shared/0 folder inside the backup, where the /storage/emulated/0/ content is normally housed, also misses the "Android" folder, where external application data is normally stored.

Physical

dd over ADB: This method failed at the first step of execution, which is to use the dd binary to read all data from the /dev/block/mmcblk0 node.

To read from this node, elevated privileges are needed. However, since this method is used in an unrooted environment, these privileges are not available. For further information of the workings of this method, see the equally named section further on, under "Software - Rooted".

Gaining access to the root account: To further continue the search for data extraction methods, elevated privileges are needed. Seeing as this is a consumer device, these privileges are not granted by default. In order to access these privileges, access to the root account of the device is required.

Three methods were attempted to gain access to the root account, they are as follows:

- ADB Backup and symlink traversal [9], a method in which a malicious backup file is crafted and restored via ADB.
- Towelroot , an exploit based on CVE-2014-3153 which exploits a bug in the Linux kernel via the futex_requeue function, to gain elevated privileges.
- Google Dev Images [10], flashable developer images distributed by Google for gaining root access on a Google Glass.

ADB Backup and symlink traversal: This method relies on backing up an application with access to the root or system account, setting up a small bash script that relies on a race condition to create a symlink and restoring a specially crafted backup file via ADB. When the method was successful, the original backup can be restored.

When trying to execute this method on the version of the Google Glass software used for this research (XE22), it seemed that this method did not work anymore. The reason for this, is that in this version of the software, it was impossible to backup and restore applications with access to the root or system account. Another issue was that the Android version used for this research had an SELinux enabled kernel, which meant that if this method had worked, the resulting privilege escalation would immediately be undone.

Towelroot: After downloading the apk and running the application via ADB commands, no root access was available. After further research the cause seemed to be that the used version of the Google Glass software(XE22) used the xrx13b kernel, while the futex_requeue bug was patched in an earlier kernel, namely xrv85b [11].

Google Dev Images: While these images provide access to the root account by easily flashing the bootloader with the fastboot utility, there is one big disadvantage of using these images. When flashing these images, the entire user data partition is erased. This makes them useless for a real forensic investigation, but not for this research. After flashing the image and executing a su command over ADB, access to the root account was confirmed.

Software – Rooted

Logical

Logical: Contrary to the last time, after running the shattered script with root privileges, a full logical image of the internal filesystem was generated. This means that when access to the root

account is a possibility, the shattered script can be of substantial value in a forensic investigation.

ADB Backup: As expected, the results of an ADB backup are exactly the same with root access as without root access.

Physical

dd over ADB: Now that elevated privileges are available, the dd binary can be used to read from the /dev/block/mmcblk0 node. The only issue that remains is piping the output to a usable location. This location cannot be on the device itself, as the resulting file will be exactly as large as the internal flash chip. The solution to this problem is to pipe the output of the dd binary over ADB. Another problem that arose, is that ADB adds windows style line endings to the end of each line. These line endings ended up in the image, rendering it unusable. The solution to this was to first encode all the data with the base64 binary, and decode it at the receiving side. A schematic representation of the used method can be found in Figure 2.

This method yields a fully usable physical image of the internal flash memory. However, because this image has been made with mounted partitions, the filesystems contained in it are marked as "dirty". *They can however still be mounted by using the noload* option, which mounts the filesystem without loading the associated journal.

Hardware

JTAG: For using the JTAG method, a JTAG header on the mainboard is needed. This is normally a 10 or 20 pin header on the mainboard of the device. After carefully disassembling the Google Glass and examining the mainboard for these headers or footprints of these types of headers. Although a lot of test pads and via's were available at the surface of the motherboard of the device, none seemed to indicate the presence of a JTAG header. As such, it is not possible to use this method.

5-Wire Method: The first step in using this method is finding a place on the motherboard were the five necessary lines are accessible for tapping into. To achieve this, the datasheet of the used eMMC was checked for the location of the contacts associated with these lines. Following that, pictures of the front and back of the mainboard were taken.

From these pictures a semi-transparent overlay of the front of the board (containing the eMMC) over the back of the board was made. This overlay is visible in Figure 3.

Using this overlay, possible locations for the needed lines can be seen on the backside of the board. From this overlay, an immediate problem can be determined. The datasheet clearly specifies that the needed lines are located on the upper and lower left side of the eMMC, which are both blocked by the square gray chip containing the WiFi and bluetooth hardware. On the topside of the board, the chip marked Elpida (Figure 3) contains the RAM and is stacked on top of the main processor.

Combining this knowledge with the fact the mainboard is multilayered, it is very probable that the needed lines run through layers on the inside of the board, between two ground planes. This is a technique commonly used to minimize outside noise to these lines and keep them as short as possible.

Chip-off: In this method a hot air rework station is utilized to remove the memory chip from the mainboard. The extracted memory chip is then inserted into the NFI Memory Toolkit II to extract the data and make a physical image of the chip.

Normally, when this is correctly executed, the chip can be placed back on the main board and the device is usable again. In this case however, all chips were sealed to the mainboard with resin. Luckily, this did not interfere with the extraction of the memory chip from the mainboard, but did damage the solder balls connecting the CPU and the RAM, rendering the device unusable. Given that all data was extracted via different methods, this did not interfere with the progress of this research.

Analysis of extracted data: Since there was a waiting period

Figure 2. Data Stream for Extraction via dd over ADB.

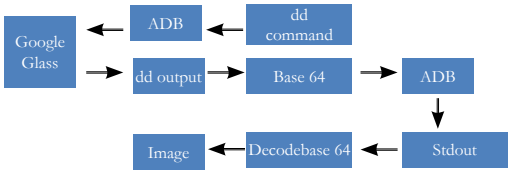
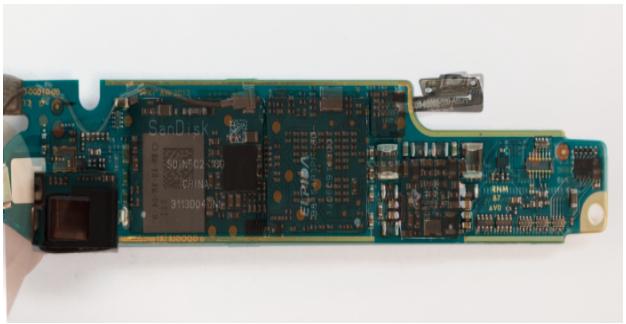


Figure 3. Semi-Transparent Overlay of the Front Over the Back of the Mainboard.



for the chip-off method, the following research was based on the image acquired with the **dd over ADB** method. The result of this analysis was verified with the images acquired from the chip-off method, to verify the conclusions of this research.

Image structure: The first step in researching the extracted data, is determining the structure of the acquired image. Using **fdisk**, the partitions in the image can be listed.

Taking into account that this research is bound to a limited amount of time, a scope of partitions had to be established. Because this research focuses on user-generated artifacts, partitions 1 to 7 are excluded from further research, seeing as they do not contain a writable filesystem and are not accessible to the user during normal operation.

This leaves 3 partitions, 8, 9, and 10. Partition 10 (/system) does contain a known and writable filesystem, however it is always mounted read-only in normal operation. This means the user cannot write to this partition and it will most likely contain little user artifacts.

This can be verified by downloading a factory system image [10], unpacking it and comparing (diff) it against the system partition in the image acquired from the device. Based on this information, the decision was made to exclude this partition from further research. The /cache partition seems to be an interesting place for forensic artifacts at first. Looking closer however, it seems to only have a couple of uses:

- Temporary storage for OTA(over-the-air) updates
- Temporary storage for Play-store downloads
- Logs from the recovery mode
- The "lost+found" directory which contains artifacts from partition recovery.

Since the chance of discovering new artifacts from further research seems unlikely, this partition will be excluded from further research.

This leaves only one partition for further investigation, the /userdata partition, which is also by far the biggest partition.

Results

The search for artifacts in this partition was split into four categories, depending on the type of data:

- Databases
- Log files
- Cache
- Media

Databases

Given that Google Glass runs on a slightly modified version of Android 4.4.4, a lot of databases were expected in this investigation. This proved to be true, and as such only the most important Glass-specific databases will be mentioned in this article.

One of the first things that can be remarked, is the naming schema for Google's own "Glassware". These applications all shared the same prefix, namely "com.google.glass". Another thing to note is that all databases found were SQLite 3 databases.

One of the most important databases is timeline.db, found at /data/data/com.google.glass.sync/databases/timeline.db. To understand what's contained in this database, a little more information about the user interface of the Google Glass is needed. This user interface is different from the normal Android launchers. It is built up as a timeline, where for each event (eg. taking a picture, receiving a text message and so on) a new "card" is added to the timeline. The information contained in these cards is stored in the timeline.db and it describes every event extensively.

Another important database is found at /data/data/com.google.glass.home/databases/entity.db, which stores all saved contacts, groups and associated metadata. This metadata includes the last action performed with this contact and the amount of times something was "shared" with it.

In timeline.db and other Glass-specific databases, events are coupled with api numbers, rather than application names. The /data/data/com.google.glass.boutique/databases/ boutique.db, can be used to derive application names from api numbers. The records in the glassware table of this database contain the api number and the associated application name, making it possible to link the events in the timeline.db to a certain application. Caution has to be taken however, as the api numbers are stored as unsigned 64-bit integers.

The /data/data/com.google.glass.camera/databases/compan-

Table 1: Partitions in Extracted Image.

Number	Start	Size	Filesystem	Name
1	131kB	131kB	xloader	msftdata
2	262kB	262kB	bootloader	msftdata
3	524kB	524kB	fpga	msftdata
4	1049kB	262kB	bootconfig	msftdata
5	1311kB	262kB	misc	msftdata
6	1573kB	8389kB	recovery	msftdata
7	9961kB	8389kB	boot	msftdata
8	18.4MB	1074MB	ext4	system
9	1092MB	805MB	ext4	cache
10	1897MB	13.9GB	ext4	userdata

ion_photo_sync database, can be of forensic importance. It contains all metadata pertaining to the syncing of pictures with Google+. This might give some insight as to whether photos were deleted.

All data associated with custom "hotwords" (words that the glass can recognize in speech), such as contact and group names is stored phonetically in /data/data/com.google.glass.voice/databases/prons.

Logfiles

All logfiles generated by the system (specific app logs might differ from this) are stored in two locations, namely /data/system/dropbox and /data/media/0/logs.

The /data/system/dropbox directory does not have any association with Dropbox the cloud storage service, contrary to what the name might imply. This directory houses logs pertaining to battery discharge, dumpsys data, hardware events, kernel messages, boot messages and a couple other miscellaneous logs.

The /data/media/0/logs directory contains all the syslogs, with data from the boot logs from the /data/system/dropbox directory, it can be determined to which session these syslogs belong.

Cache

Cache files can be found in a couple of places on the Glass, but by far the most extensive storage of cache is /data/private-cache. It contains a number of different cache files:

- a_[UNIX Timestamp].call, which, as the extension suggests, contains data about phone calls made/received with the Glass. The structure of these files is visible in Figure 4.

The meaning of the last two variable might seem obvious, but a consensus of their meaning could not be confirmed by reference testing, so they remain unknown.

- a_wear_[source_tag_id]_[type]_[UNIX Timestamp], cached files originating from Android Wear interactions.
- gi_[api-nummer]. SMALL, PNG files with 8 bytes prefixed to the header, these are cached icons for Glassware applications.
- h_[timeline-id], cached Google search results, in the shape of HTML files. These HTML files are the exact pages presented to the user as search results. The metadata from these results can be found in the timeline.db by searching the id in the filename.
- p_[32 alphanumerical symbols]-640-640-0, JPEG files, with differing content (and presumably origin). The two variables with a value of "640" seem to indicate the resolution of the images, this is however not the case for most images.
- The last variable "0" seems to indicate if the files were created locally or remotely. In all cases were this flag is "1", the resolution in the first two variables was correct.
- ss_[UNIX Timestamp].png, screenshots in PNG format. The timestamp seems to indicate their creation time.
- Files with a t_ prefix, these all seem to be thumbnails, with

Figure 4. Structure of a .call file.

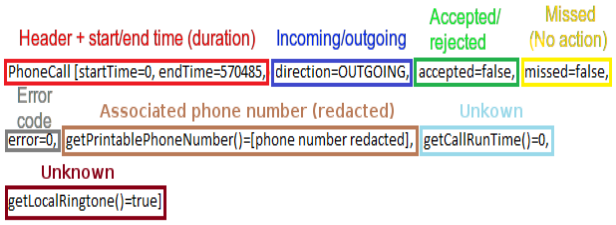


Figure 5. Header of a cache file.

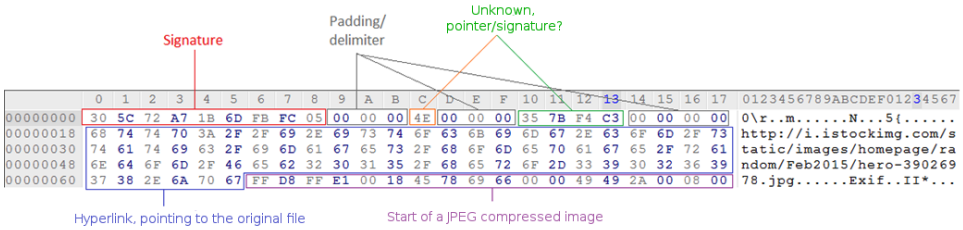
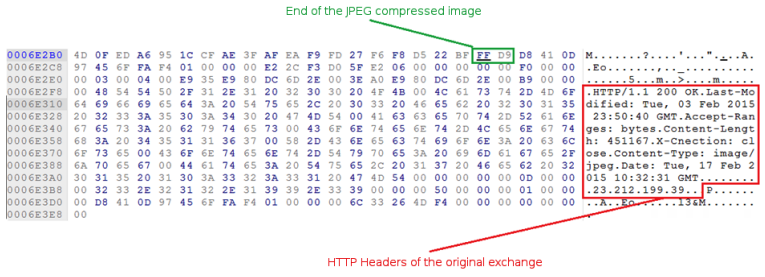


Figure 6. Footer of a cache file.



the respective location of the original image in their filename in multiple ways (path, timeline id, filename +extension)

Another forensically interesting location, when searching for cache files is /data/data/[app_name]/app_webview/Cache. This location is where cache from browsing is stored for apps that have a WebView component. These files can be very extensive, one example of a found cache file is seen in Figure 5.

Not only can the original hyperlink be extracted from the cache file, the full jpeg image which it refers is present in the cache file. As can be seen in Figure 6, the footer seems to also contain the original HTTP headers.

The last important cache files were found in de data folder of the com.google.glass.maps application, present at /data/media/0/Android/data/com.google.glass.maps/cache/. In this location multiple files with a cache_ prefix exist, the content of these files could not be identified.

What could be identified however, were the contents of ._speech_nav_[#].wav files present in the directory. These are cached spoken navigation instructions, stored as regular WAV files. These might place the device at a certain location. To find the creation time of these files, the Android MediaProvider database can be checked, present at /data/data/com.android.providers.media/databases/external.db.

Media

All media created with the internal camera can be found at /storage/emulated/0/DCIM/Camera. All files use the following naming scheme:

[Date(YYYYMMDD)]_[Time(HHMMSS)]_[Time(ms)].[mp4/jpg]

There is however, another place where some media files reside, namely/data/data/com.google.glass.voice/recorded_audio. Stored in this location are spoken orders that were given to the Google Glass. Not all spoken orders are stored, this only seems to be the case for Google searches, navigation requests and dictated messages. The files are PCM encoded audio, with a sample rate of 8kHz and signed 16-bit encoding.

The naming scheme for these files is as follows:

[type]_[date(YYYYMMDD)]_[time(HHMMSS)]_[###]_8000.pcm

Type can be one of three possible types:

- DICTATION, for dictated messages.
- NAVIGATION, for navigation requests.
- VOICE_SEARCH, for Google searches.

The meaning of the 3 numbers "[###]" is unknown, but seems to be the milliseconds time.

Conclusion

As can be seen from the results, a lot of forensically interesting information can be retrieved from the Google Glass. A short summary of this information:

- Pictures
- Video's
- Contacts
- Social media activity
- Locations and destinations (navigation)
- Audio (including users voices and TTS cache)
- Interactions from various messaging platforms
- Sync information (eg. for Google+)
- Connected devices
- Android Wear interactions
- Web browsing behavior and resulting artifacts
- Search history including results
- Phone calls

This is a trend that is very visible in "wearables" and it is to be expected that these devices will play a growing role in future forensic research.

Focusing on this research however, the value of this data will largely be dependent on the eventual consumer version of Google Glass. Seeing as this research was conducted on the Explorer Edition of Google Glass, the final product may be quite different. It is to be expected however that some of the basic workings and principals of this version will still apply to the consumer version, e.g. the abundance of SQLite database, the timeline interface, the strong emphasis on social media functions etc.

A basis for any future research might be based on finding exploits, to allow for better software-based imaging and perhaps memory extraction. This might aid in circumventing the possible cryptographic functions in future Android versions.

References

[1]. Ayers Rick, Brothers Sam, Jansen Wayne (2014) NIST Special Publication 800-101 Revision 1 - Guidelines on Mobile Device Forensics. National Institute of Standards and Technology.
[2]. Desautels Julie (2014) Google Glass Timeline Forensics. Blogspot.
[3]. Champlain College. Computer & Digital Forensics blog, 2014.
[4]. Bryce Chapin (2014) Shattered - Google Glass Acquisitition an Analysis Tool. GitHub.
[5]. Zeroplus. eMMC Technology Application, June 2012.
[6]. Netherlands Forensic Institute. NFI Memory Toolkit. April, 2011.
[7]. Elenkov Nikolay (2015) Android backup extractor. Github.
[8]. Google Inc. MyGlass. Google Play Store. November, 2014.
[9]. Freeman Jay (2013) Exploiting a Bug in Google's Glass.
[10]. Google. System and Kernel Downloads. Google Developers. May, 2015.
[11]. Yick Kai (2014) android/kernel/omap/glass-omap-xrv85b. Google Git.
[12]. Steve Watson, Ali Dehghantanha (2016) Digital forensics: the missing piece of the Internet of Things promise, Computer Fraud & Security. 2016(6): 5-8.
[13]. Marcel Breeuwsma, Martien de Jongh, Coert Klaver, Ronald van der Knijff, Mark Roeloffs (2007) Forensic Data Recovery from Flash Memory. Small Scale Digital Device Forensics J. 1(1): 124-132.