Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Computer Engineering Department
Degree in Computer Engineering

GBOARD FORENSIC ANALYZER WITH AUTOPSY

STUDENT JOÃO MIGUEL LAVOS LOURENÇO
STUDENT LUÍS JORGE MONTEIRO FERREIRA

Leiria, July 2021

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Computer Engineering Department
Degree in Computer Engineering

GBOARD FORENSIC ANALYZER WITH AUTOPSY

STUDENT JOÃO MIGUEL LAVOS LOURENÇO (2182207)
STUDENT LUÍS JORGE MONTEIRO FERREIRA (2181140)

Project carried out under the guidance of: Professor Miguel Cerdeira Marreiros Negrão, Professor Miguel Monteiro de Sousa Frade, and Professor Patrício Rodrigues Domingues.

Leiria, July 2021

DECLARATION

I declare, under honor, that the work presented in this report, with the title *"GBoard Forensic Analyzer with Autopsy"*, is original and was made by the Student João Miguel Lavos Lourenço (2182207) and the Student Luís Jorge Monteiro Ferreira (2181140) under guidance of the Professor Miguel Cerdeira Marreiros Negrão, the Professor Miguel Monteiro de Sousa Fradeand the Professor Patrício Rodrigues Domingues.

Leiria, July 2021

Student João Miguel Lavos Lourenço

Student Luís Jorge Monteiro Ferreira

ACKNOWLEDGMENTS

We thank our teachers and mentors for the opportunity of experiencing the area of digital forensics. Without their help we wouldn't have grown so much in the past semester. Using and learning various tools has helped to deepen our knowledge and understanding of this area as well as the right steps needed to follow the most beneficial path in relation to what is needed to do proper research.

ABSTRACT

Technology has been rapidly growing and improving exponentially throughout the years, and this progress has made communicating with each other rather more simple and exciting. Smartphone keyboards have improved immensely in the past years, and now they are at a point where they are not just simple keyboards with little keys just to write simple sentences that can be replicated by hand. They are full-grown programs with multiple settings to tweak and customize the way the user sees and interacts with them. Some even have suggestions to go along with, to give the user a better user experience and to save time writing. In this project we study the Google Keyboard (GBoard), because of its popularity, the company that created it, and what it promises to do. GBoard came out with some incredible features, but the one that got our attention was the ability to learn and study the patterns of the writing of a user to give increasingly better suggestions with the passing time. However, for it to be able to accomplish this goal, a lot of information has to be analyzed. And, in this case, all data is driven from the user's input. What they write, the way they write, where they write, etc. By this, we can assume a lot of information is being collected to be studied, and thus there might be a way to get a hold of that information. From a point of view of a digital forensics analysis worker, this could be a way of obtaining much-needed information from encrypted or even deleted apps.

This document details the process of the examination of GBoard's internal cache data. This ranges from critical user's private information, such as all the keyboard's input data (every keypress, word written or chosen from the suggestions, every word deleted), GBoard's Personal Dictionary where a user can define words and shortcuts, translation suggestions, emojis and other non-words used in GBoard to the clipboard's information. It also details the process of development of data sources and report modules for Autopsy (an open-source application for digital forensics), the tools used to research GBoard and its databases, and their process of extraction.

Keywords: Digital Forensics Analysis, GBoard, Autopsy, Mobile devices, Virtual Keyboards, Android.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

ACRONYMS

ADB	Android Debug Bridge.
API	Application Programming Interface.
APK	Android Package.
AUR	Arch User Repository.
CD	Continuous Delivery.
CI	Continuous Integration.
CLI	Command-line Interface.
CSV	Comma-separated values.
DTO	Data Transfer Object.
FIFO	First In, First Out.
GIF	Graphics Interchange Format.
GMS	Google Mobile Services.
GNU	GNU's not UNIX.
GUI	Graphical User Interface.
HAML	HTML abstraction markup language.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
JSON	JavaScript Object Notation.

MARISA	Matching Algorithm with Recursively Implemented StorAge.
MobSF	Mobile Security Framework.
PHP	PHP: Hypertext Preprocessor.
PNG	Portable Network Graphics.
PR	Pull Request.
RDBMS	Relational Database Management System.
RPM	RPM Package Manager.
SQL	Structured Query Language.
SSL	Secure Sockets Layer.
SVG	Scalable Vector Graphics.
TLS	Transport Layer Security.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
XLS	Excel Spreadsheet.
XML	Extensible Markup Language.
YAML	YAML Ain't Markup Language.

1

INTRODUCTION

Over the years, technology improved exponentially. This growth helped to facilitate the work in multiple industry sectors, and likewise, in the everyday life of each individual. Things like ordering food and groceries, shopping, or communicating with someone, are all tasks that are now possible to do online, and with ease. However, these improvements can be seen as a double-edge sword, in which the same can be applied to the ability to perform cybercrime. Malicious websites, catfishing, fraud, terrorism, extortion, obscene content, harassment, are some of the online crimes conducted daily. One of the perfect weapons to use for these is a mobile device.

Almost all computing tasks can now be done with a mobile phone. More than 87 thousand apps are released every month in the google app store (Department, 2021), varying from games to business to finance to entertainment apps, and with more than 90% of the market being free (Curry, 2021), all this can be found and downloaded with just a few clicks. Of these applications, some have the purpose of sharing and saving the personal data and private information of a user. Tik-Tok[1], Facebook[2], Twitter[3], Instagram[4], WhatsApp[5], and other social media applications are filled with conversations and posts of users. It's safe to say, with more than 3 billion active Android devices (Cranz, 2021), more than 1 billion active IOS devices (Kastrenakes, 2021), and with more than 5 billion people that currently have a mobile device (Turner, 2021), which means more than 60% of the world population has a mobile device, that means if someone commits a crime they probably have a mobile phone. Furthermore, with almost 4 billion people actively using social media, we can assume there is a high percentage that the same individual uses a social network (Dean, 2021).

From a forensics point of view, this information is crucial because the same data that is shared and stored in a device or a mobile application can be collected

[1] https://www.tiktok.com/
[2] https://www.facebook.com/
[3] https://twitter.com/
[4] https://www.instagram.com/
[5] https://www.whatsapp.com

and analyzed for the good of an investigation. Forensic science plays a big part in criminal investigations. This field helped to improve the success of cases and trials by examining and finding evidence which without science wouldn't be possible. Fingerprint scans, DNA samples, toxicology, ballistics, are some of the methods used to gather valuable information and move a step forward towards the end goal. Digital Forensics, a sub-branch of Forensic Science, takes advantage of this and other digital sources to collect valuable information. With the everyday growth of new digital applications and updates from existing ones, Digital Forensic scientists must update regularly their tools to extract information from current applications and create new tools for the upcoming and most used ones.

The goal of this report is to research the Google Keyboard application and find relevant data that can be used at a forensics level. The Google Keyboard (GBoard[6]) application was first released in May 2016 and launched with some outstanding features and has now more than 1,000,000,000 installations (Google, 2021). Its predictive typing engine was what caught our attention. What makes a keyboard's data so important is that everyone uses it for everything. Doing a simple google search, writing notes, messaging, chatting with friends in an app, and all the other activities that involve typing must be done with the help of a keyboard. So, not only could we potentially extract full conversations with a particular individual but also extract private information that was never made public, was already deleted or modified, or even see the thought process when writing a full sentence. The GBoard's predictive typing engine could give all of this. As it's able to suggest words depending on the context and the way a person writes as well as analyzing the written text to improve itself, all this information and data must be stored somewhere. As such, the objective is to research the GBoard application, find if GBoard processes data from other apps and while incognito mode, find where all data is being stored, if it can be accessed and extracted, analyze the information, filter what is most important in terms of relevant information, and create a tool that does this whole process automatically.

To help us reach our goal, we will use Autopsy, a digital forensics software that makes it easier to work with The Sleuth Kit[7] plugins. The Sleuth Kit is a library of utilities to extract data from storages to facilitate forensic analysis. With this library, creating a tool to perform some type of forensic analysis is easier because much of the groundwork is already performed under the hood by the library itself,

[6] https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en_US&gl=US

[7] https://www.sleuthkit.org/

and we just have to worry about the implementation at a higher level. This saves a lot of time to think and work on other important issues like research, planning, and providing a better final product. Autopsy is a graphical user interface and is bundled with The Sleuth Kit. This tool serves the purpose of displaying the results of an analysis, making it easier to flag the data collected.

1.1 MOTIVATION

In the digital forensic analysis world, gathering information is everything. Every bit of information counts, and sometimes even one that can be considered insignificant by many can turn the tables on an ongoing investigation. However, analyzing information about an individual's personal and private data has a higher chance of achieving the end goal. The Google Keyboard (GBoard) provides this kind of data even after its been deleted from its origins. Access to private conversations, google searches and notes is some of the information that can be extracted from this keyboard. This is accompanied by the date and time the action was performed, in what application it was performed, and, with the case of special words such as emojis, how many times they were used. These were the reasons and motivations behind the birth of this project. Furthermore to add weight to this investigation and to give credibility to what it was believed (the data could be extracted), a group of people in the forensics analysis world had released an article on their blog talking about the contents stored within GBoard (Patel, 2016).

1.2 OBJECTIVES

The main goal of this project is to build a module for Autopsy, an open source application to forensically process digital devices, to analyze interesting GBoard artifacts for this purpose. To acheive this, the project is subdivided into three objectives:

1. **Analyze and explore GBoard components and functionalities on an Android system:** GBoard is composed by various components such as basic functionality of manual texting, translation, voice-to-text, integrated search and clipboard management. It is important to evaluate which component is more important and explore them in terms of digital forensics value.

2. **Identify the artifacts created by using GBoard on an Android system:** Even if a certain component is very valuable for forensic analysis, it is important to identify which artifacts are generated by different versions of GBoard, identify in which capability they correspond to and how important they are compared to its functionality in terms of its outcome for forensics analysis.
3. **Development and integration of various technologies to facilitate investigation of forensic cases:** It is very important to have tools and integrate them with existing ones to facilitate the investigation of forensic cases by automating certain tasks of extraction and selection.

1.3 CONTRIBUTIONS

Within the scope of this project, some relevant contributions were made, such as:

1. **Repository to install Autopsy debian package:** To provide more flexibility for the Autopsy end-users an automated repository was created to build Autopsy packages for Debian linux distribution.
2. **Repository to use Android Emulator with Docker:** To provide more easy access to an Android machine, a docker image was created with the Android emulator configured and ready to use.
3. **Deepening existing knowledge about GBoard:** Available information about GBoard internal structure is very superficial as it only covers a few internal parts.
4. **Development of GBoard Forensics standalone application:** A standalone application was created to help extracting forensic information from GBoard application data.
5. **Creation of an Autopsy data source module:** With cooperation with the standalone application, an Autopsy data source module was created to integrate the data fetched.
6. **Creation of an Autopsy report module:** With cooperation with the standalone application, an Autopsy data source module was created to report and visualized the data fetched.

1.4 DOCUMENT STRUCTURE

In Chapter 2 the concept of Digital Forensics is briefly described and presented the used technologies such as D programming language, SQLite and Android.

In Chapter 3 GBoard is briefly described with a user's perspective analysis and comparison with other applications. Some tested versions for use in this project are listed and the internal structure is described in detail, such as internal databases and other relevant files. Some rearranged data was also described.

In Chapter 4 the Autopsy digital forensic software is briefly presented, focusing on its main characteristics and functionalities. It also details on one of our contribution to Autopsy with a Linux package for both Arch Linux and Debian distributions.

In Chapter 5 it is described the developed GBoard standalone application called GBoard4A, the integration with Autopsy digital forensics software and the development of two modules for Autopsy. Some results were also presented and discussed.

In Chpater 6 some conclusions were made and presented some aspects to improve and work in the future.

2

CONCEPTS AND TECHNOLOGIES

In this chapter we present the main concepts needed to understand the developed work. We also outline the main technologies used throughout this work.

2.1 DIGITAL FORENSICS

Digital Forensics is an area of interest for recovery and research of material and information found in any digital device capable of storing digital data to help discover information crucial for the investigation of various types of crimes. Various applications of digital forensics investigations range from supporting criminal cases to private internal corporate intrusion. In terms of technical aspects, digital forensics is divided into multiple branches involving various types of devices, including computers, network devices and mobile devices (Reith et al., 2002).

Shreds of evidence found in mobile devices can range from phone calls, messages, emails and interesting application data and depending on the case, some information can be more interesting to explore. This project is specifically focused on mobile devices capable of storing messages, conversations and other information by the GBoard mobile application.

2.2 D PROGRAMMING LANGUAGE

D is a general-purpose multi-paradigm system programming language with a C-like syntax focused on writing fast, safe and readable code while keeping low-level access power. In contrast to some system languages, D emphasizes type-safe and memory-safe code, minimizing undefined behaviours and therefore maintaining the program correctness (Çehreli, 2015).

In this project, the D programming language was used to develop a standalone application to help Autopsy extracting relevant information. Using its capabilities, it was possible to create a cross-platform application with a low-level approach to

achieve better performance and still maintain the same readability that Jython offers, which is Python within the Java Virtual Machine, being one of the language that can be used to write Autopsy modules.

2.2.1 *Diet Templating Language*

Diet is a generic templating language for D programming language to generate Extensible Markup Language (XML)-based structures at compile-time. This lancuage is highly influenced by pug[1] and HTML abstraction markup language (HAML)[2] languages (rejectedsoftware, 2021). In this project, Diet was used to easily generate HyperText Markup Language (HTML) reports due to the easy integration with D code.

```
doctype html
- auto title = "Hello, <World>";
html
        head
                title #{title} - example page
        body
                h1= title
                p.
                        Simple paragraph
```

Listing 1: Example in Diet templating language

```
<!DOCTYPE html>
<html>
        <head>
                <title>Hello, &lt;World&gt; - example page</title>
        </head>
        <body>
                <h1>Hello, &lt;World&gt;</h1>
                <p>Simple paragraph</p>
        </body>
</html>
```

Listing 2: Generated HTML example from a Diet template

Listing 2 shows an example of the generated output of the Listing 1 code.

[1] https://pugjs.org/
[2] https://haml.info/

2.3 SQL QUERY LANGUAGE AND SQLITE

The Structured Query Language (SQL) is a query language designed to gather, define, control or manipulate structured data managed and stored by a Relational Database Management System (RDBMS). It is useful to provide an Application Programming Interface (API) for the application to access data from a database. SQLite is an implementation of an RDBMS written in C, intended to be used as a library (Shafranovich, 2013). This RDBMS is a popular solution for many software to store information in the local storage because it does not require a client-server connection. SQLite is one of the most used database engines in the Android operation system with a mature and stable API[3] (SQLite, n.d.).

Particularly, GBoard uses SQLite to store a personal dictionary, temporary training cache, clipboard data, and emojis expression history. To fetch all that data in the standalone application, we used a wrapper library written in D called `d2sqlite3`[4] that uses the official SQLite C library under the hood. This wrapper library provides a safer API and can handle built-in D types.

2.4 ANDROID OPERATING SYSTEM

Android is one of the most widely used mobile operating system, developed by Google, based on the Linux kernel. According to statistica[5], Android has around 1.6 billion active users worldwide with about 70% of the market share. (O'Dea, 2021) Even though Android is considered free and open-source software, most Android devices have proprietary software out of the box including Google Mobile Services (GMS) and some applications such as GBoard, Chrome and other vendor-specific applications depending on the device. Because most people use the stock Android distribution, most likely they will have GBoard installed.

[3] https://developer.android.com/reference/android/database/sqlite/package-summary

[4] https://github.com/dlang-community/d2sqlite3

[5] https://www.statista.com/

Figure 1: Screenshot of a Pixel 4A running Android 11 (Megamanfan3, 2020)

2.4.1 *Android Debug Bridge and Android Emulator*

Android Debug Bridge (ADB) is a tool to make communication possible with an Android device and facilitate the process of installing and debugging an application, access a shell to run various commands and access logs. There are three main components to make communication via ADB possible: a command-line tool that acts as a client and runs on the host, a daemon, running on the Android device and a server running on the host machine that manages the communication between the client and the daemon (Google Developers, 2021).

The Android Emulator is an application that simulates an Android device. It does so by running a Virtual Machine with the Android operating system. One of the best functionalities it provides is the ability to test an application on a vast number of devices with different Android API levels, different screen density and resolution and the ability to simulate a vast number of hardware sensors.

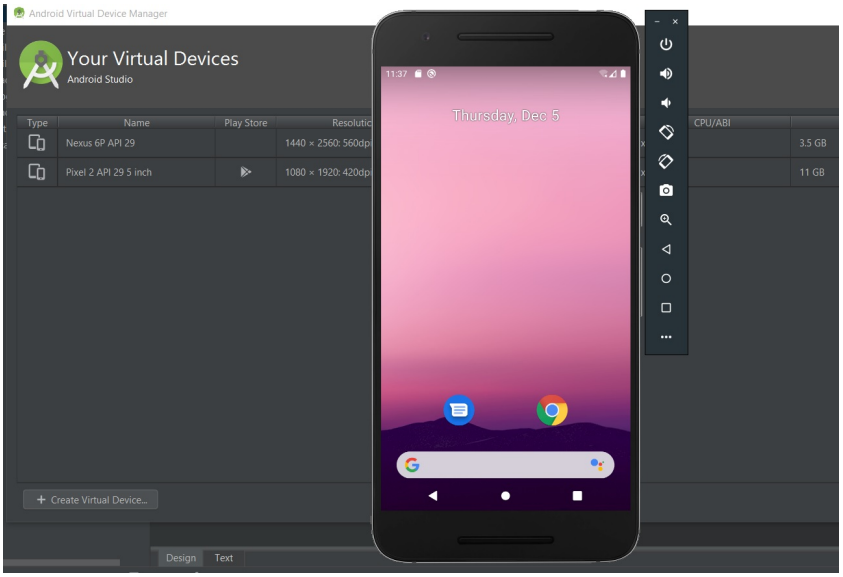

Figure 2: Screenshot of a running virtual device on Android Emulator (Chung, 2019)

This project used both technologies to either acquire the required forensics data and test various versions of GBoards in different environments with both virtual and real-world devices. Although, acquiring forensics data via ADB could be very limiting, from the forensics analyzer perspective, because Android permissions are becoming more aggressive and those can somehow prevent that gathering. That being said, some alternatives such as data image dump via recovery or fastboot and even direct flash dump can be more beneficial, depending on the case.

In the next chapter we presented the main concepts needed to understand the developed work. We also outlined the main technologies used throughout this work.

3

GOOGLE KEYBOARD (GBOARD)

In this chapter, we briefly describe the user's and the forensics analyzer's perspective of GBoard, compare it with other popular applications, list the tested versions, present a walkthrough of the internal structure of GBoard and the rearranged data performed.

GBoard is one of the most known virtual keyboard applications for mobile devices. It is closed-source, developed by Google and targets both Android and iOS devices. (Patel, 2016) Because GBoard is often found pre-installed in Android devices, most of the people use it as their primary keyboard, and it is one of the reasons why the application has more than 1 billion installations. (Davenport, n.d.)

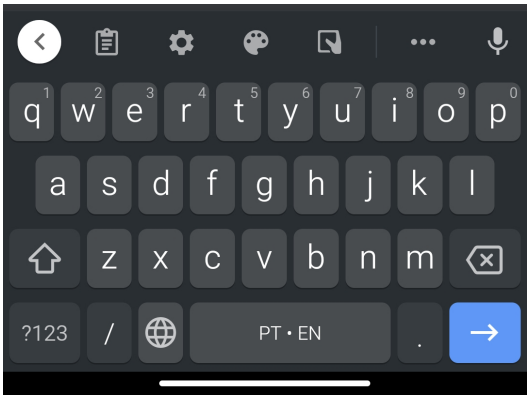

Figure 3: Screenshot of the main view of Google Keyboard (GBoard)

3.1 COMPARISON WITH OTHER APPLICATIONS

To validate the popularity of this keyboard, a comparison was made with other popular keyboards that are available in the Google Play store[1].

[1] https://play.google.com/store/apps/

Name	Rating (Reviews)	Downloads
Google Keyboard (Gboard)	4,5 (8M+)	1B+
Microsoft SwiftKey	4,3 (3M+)	500M+
GO Keyboard	4,5 (4M+)	100M+
GO Keyboard Lite	4,4 (2M+)	100M+
Samsung Keyboard	3,0 (3m+)	500m+

Table 1: Comparison with the most popular keyboards in Google Play store

Table 1 shows the GBoard popularity with its high download count and user rating from Google Play store. Other relevant competitors are the Microsoft SwiftKey[2] and the Samsung Keyboard[3]. Although they are worth researching, both are out of scope of this project. Additionally, the Samsung Keyboard could be taken into consideration even though the numbers are not relevant because Google Play only counts updated apps through their store and Samsung has its software manager to update system apps.

3.2 TESTED VERSIONS

For this project, six different versions were tested in three different physical and virtual devices to determine whether a certain feature is present and check if it is feasible to analyze that data. Because GBoard bundles system libraries such as Tensorflow[4], the same version can have multiple builds for different architectures. There is also different build types for applications already installed in the device emulator and the released version in the Google Play store[5].

Version	Type	Arch	Device	API Level
9.4.11.312687073	preload	x86_64	Android Emulator	30
10.0.02.338070508	release	x86	Genymotion	29
10.2.07.351353117	release	arm64-v8a	Xiaomi MI9 SE	30
10.4.04.361808908	release	arm64-v8a	Xiaomi MI9 SE	30
10.4.04.361808908	release	x86	Genymotion	29
10.5.03.367007960	release	arm64-v8a	Xiaomi MI9 SE	30

Table 2: GBoard Tested versions

[2] https://play.google.com/store/apps/details?id=com.touchtype.swiftkey
[3] https://play.google.com/store/apps/details?id=com.samsung.keyboard.themes
[4] https://www.tensorflow.org/
[5] https://play.google.com/store

Table 2 lists the tagged version associated with a type and an architecture along with the information of the device on which the application was tested on such as the device name and the Android API Level.

In our experiments, we did not detect differences caused by running a given version in different architectures, API levels or build types. Nevertheless, some versions have significant differences. For instance, version 10.0.02.338070508 introduced the expression history database, while a more recent version – 10.5.03.367007960 – seems to no longer generate a training cache, a crucial forensic data source of user information.

3.3 USER PERSPECTIVE

From the user perspective, GBoard is very comfortable to use due to a lot of features that make the user experience effortless. Examples include easy content sharing like GIFs, emojis and emoticons with smart suggestions (Ramaswamy et al., 2019), predictive typing that suggests the next word depending on its context (Hard et al., 2019a), multi-language support (Esch et al., 2019), themes, personal dictionary with the users most used shortcuts and words, voice dictation, clipboard management, embedded translation and one-hand swiping. (Patel, 2016)

3.4 FORENSICS PERSPECTIVE

Naturally, these features yield digital forensic artifacts that can be valuable to forensic analyzers. Examples of such forensic artifacts are all copied text and images captured by the background service that GBoard is running, training cache data with users input information, words used frequently by the user from the personal dictionary, translation input data and much other information captured by the virtual keyboard application. Combining pieces of this data can be crucial to discover patterns or complete criminal cases.

3.5 INTERNAL STRUCTURE

GBoard stores most of its internal files in the data partition of Android. Access requires root privileges, as a normal user/applications do not have enough permissions

to read GBoard's data. This prevents everything other than system applications to access possible private information stored by the normal user apps.

Folder Name	Description
`cache`	Temporary cache files
`code_cache`	-
`databases`	SQLite Databases
`files`	Normal files storage
`no_backup`	-
`shared_prefs`	Shared preferences

Table 3: Filesystem structure of GBoard internal data folder

Table 3 shows the hierarchy of relevant directories of GBoard private storage. Some folders are unknown and empty. Next, we analyze the SQLite databases.

3.5.1 *Databases*

GBoard internally has some SQLite3 databases to manage user data including `trainingcache*.db`, `superpacks.db`, `PersonalDictionary.db`, `gboard_clipboard.db`, `expression-history.db` and other databases depending on GBoard's version and on how GBoard is being used. For example, `expression-history.db` only appears in the most recent versions of GBoard. Tracking changes between databases was a major challenge in this project, as the release frequency of new GBoard versions is quite high. This was a hard task since internal databases do not have a proper changelog and the only way to do it is by checking one by one.

Training Cache

The training cache, more specifically `trainingcache2.db`, is a database with user data temporarily managed in First In, First Out (FIFO) order. It holds a maximum of 1500 entries, which are used by GBoard for federating learning using TensorFlow (Hard et al., 2019b). Federated learning is a deep learning scheme, where a bunch of devices contributes anonymously for the training of deep learning algorithms. According to Hard et al. (2019b), the participation of a mobile device only happens at nighttime, when the device is charging and connected to a wifi network. According to the tested versions, this database appears to be already present since version 9.4.11.312687073.

Citation from Yang et al. (2018): "We collect training data for this model by observing user interactions with the app: when surfacing a query suggestion to a user, a tuple (features; label) is stored in an on-device training cache, a SQLite based database with a time-to-live based data retention policy".

Forensically speaking, this database is the most important for user data collection, even temporary because it contains inserted and deleted words associated with a sequence number and a timestamp, making the data very trackable. It is important to note that some specific application fields such as password fields are not captured by this database.

Moreover, interestingly, this database stores written input on incognito mode, making this functionality pretty useless in a security perspective. This is considerably bad for the user, as secret information is often written in this mode, although very crucial in a forensic perspective.

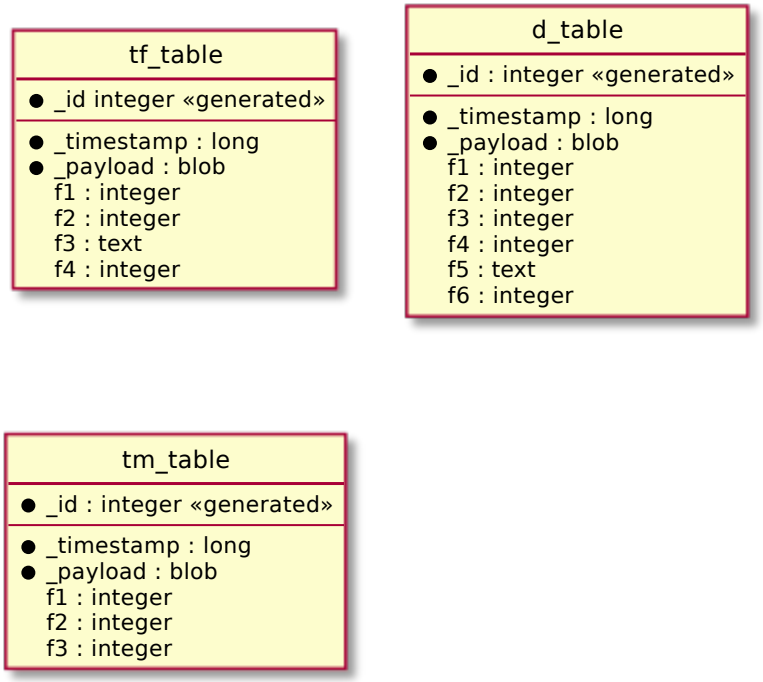

Figure 4: Relevant tables from the database schema of the training cache

Figure 4 shows three tables from `trainingcache2.db`: `tf_table` holds the typed character sequences by the user, `d_table` stores the deleted character sequences and `tm_table` keeps the position in the form of a global sequence of the whitespaces and emoji characters written by the user. With the information contained in these three tables, it is possible to construct a typed phrase.

Field	Description
`_id`	Auto-generated primary key
`timestamp`	When the entry was created (UNIX epoch milliseconds)
`_payload`	Blob encoded in Protocol buffers

Table 4: Common fields of training cache database tables

More in-depth, each table has an auto-generated primary key `_id`, an associated timestamp field named `_timestamp` (UNIX Epoch milliseconds) and a payload named `_payload` encoded in a protocol buffer[6] (Table 4). The other fields meaning depends on the table. The table `sqlite_sequence` is responsible to store the last incremented values of the global sequences used at least once by the tables. This table is an internal table from SQLite.

In dumps performed by Khatri (2021), `_payload` field has additional data in the protocol buffer, such as smart suggestions, although, in our tested versions, none of our dumps produced this behaviour, unfortunately.

Field	Description
`f1`	Unknown
`f2`	Increment of the global typed sequence
`f3`	Negative accumulation of the field `f4`
`f4`	Character sequence size
`f5`	Representation of the deleted string
`f6`	Unknown

Table 5: Fields description of d_table table

For the `d_table` table (Table 5), the `f2` field is an increment of the global typed sequence, the `f3` field is a negative accumulation of the field `f4`, which is an incrementation of the previous values by combining them with a negative operation, the `f4` field represents the character sequence size and the `f5` field is the representation of the deleted string. The fields `f1` and `f6` are unknown.

[6] https://developers.google.com/protocol-buffers/

Field	Description
`f1`	Unknown
`f2`	Increment of the global typed sequence
`f3`	Representation of the typed string
`f4`	Unknown

Table 6: Fields description of tf_table table

For `tf_table` table (Table 6), the field `f2` is an increment of the global typed sequence and field `f3` is the typed string. Fields `f1` and `f4` are unknown.

Field	Description
`f1`	Unknown
`f2`	Increment of the global typed sequence
`f3`	Unknown

Table 7: Fields description of tm_table table

Finally, for table `tm_table` (Table 7), `f2` represents the global typed sequence. The fields `f1` and `f3` are unknown.

Although this database contains much more tables, it was found empty with the presented test conditions, unfortunately. According to Yogesh Khatri's forensic blog (Khatri, 2021), some version contains much more information when tested with a Pixel phone such as the application package name, text field path and even the type of the text field. Because we didn't have a Pixel phone for this project, we decided to not rely on this behaviour unless tested properly.

Additional investigation was performed to check why those databases were not generated. By decompiling the GBoard Android Package (APK) using Mobile Security Framework (MobSF), we detected that, even in the most recent versions of GBoard, training cache database files names are hard written in the code and other versions of this database were also present in the strings literal table such as `trainingcache\%d` and `trainingcachev2.db`. Unfortunately, no interesting logic were found to successfully replicate the behaviour because all code was obfuscated and hard to read.

Clipboard

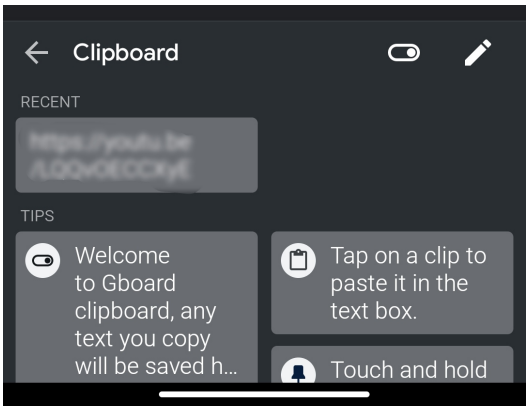

Figure 5: GBoard clipboard management feature enabled

The clipboard database – `gboard_clipboard.db` – stores all the copied items captured by the Android clipboard. According to the Android documentation (Project, 2020), Android provides a system service to access the `ClipboardManager` global instance. This service allows an app to monitor and capture all content that passes through Android clipbard. GBoard listens to this service to grab everything from the clipboard and stores an entry in the database and/or in the file storage, depending on the content type. However, note that this content is only available when the clipboard management feature is enabled in the GBoard settings, which is disabled by default.

This feature has special interest from forensic analysts that are looking for credentials inside the clipboard, because, even though the keyboard detects password fields, GBoard still stores them, if copied to the clipboard. This is particularly interesting, because, as seen in Figure 5, although the GBoard app only shows recent clipboard entries, within its databases, it keeps all data that get copied into the clipboard from the beginning.

According to the tested versions, this database is already present since version `9.4.11.312687073`.

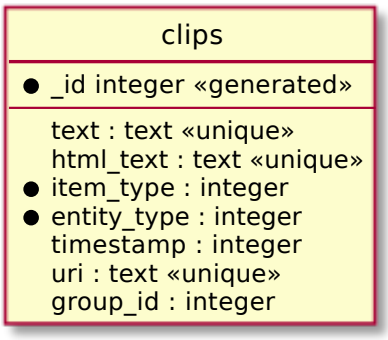

Figure 6: Table from the database schema of the clipboard

Figure 6 shows the `clips` table which holds all saved clipboard entries from the GBoard background `ClipboardManager` service listener. Table 8 lists the fields of the clips table.

Field	**Description**
`_id`	Auto-generated primary key
`text`	Representation of the copied text, if available
`html_text`	Representation in HTML of the copied text, if available
`item_type`	Partially unknown
`entity_type`	Partially unknown
`timestamp`	When the entry was created (UNIX epoch milliseconds)
`uri`	URI to the copied document
`group_id`	Unknown

Table 8: Fields description of clips table

More in depth, `clips` table has an auto-generated primary key `_id`, the `text` field which represents the copied text, if any, the `html_text` field which is the HTML representation of `text`, if it is a text entry and the HTML representation is available, the `timestamp` field representing a timestamp in UNIX epoch formatted in milliseconds and the `uri` field with the Uniform Resource Identifier (URI) to the copied document, if the clipboard entry is a non-text entry, such as an image. The fields `item_type` and `entity_type` are partially unknown and `group_id` is completely unknown. Even with a self descriptive field name, the possible values have uncertain meaning.

Personal Dictionary

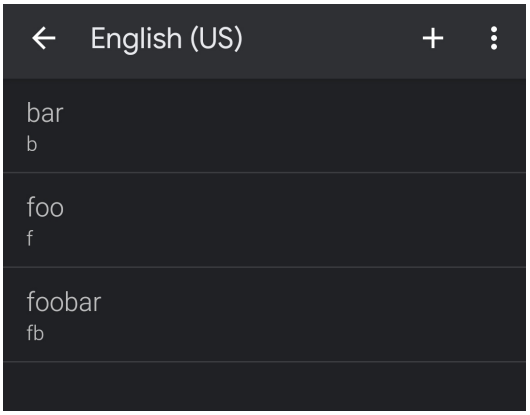

Figure 7: Personal dictionary feature of GBoard

Personal dictionary (`PersonalDictionary.db`) is a database that stores a user defined key-value entry of frequent words and shortcuts, respectively. This is very useful when the user writes a specific word very frequently, or in a forensics perspective, when the user writes a word but with a "masked" meaning with the objective to hide information, for example, "money" to "paper copies". According to the tested versions, this database is already present since version 9.4.11.312687073.

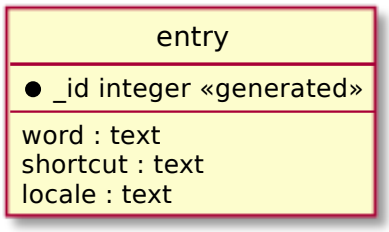

Figure 8: Table from the database schema of the personal dictionary

Figure 8 shows the structure of the table `entry` that allocates every key-value entry of the dictionary.

Field	Description
`_id`	Auto-generated primary key
`word`	Word to shortcut
`shortcut`	Shortcut to be used by the user
`locale`	Locale of the desired keyboard

Table 9: Fields description of entry table

This table has a `word` field to store the word to shortcut, a `shortcut` field that represents the shortcut and a `locale` field to store the locale of the desired keyboard, because GBoard supports multi-language keyboards.

Expression History

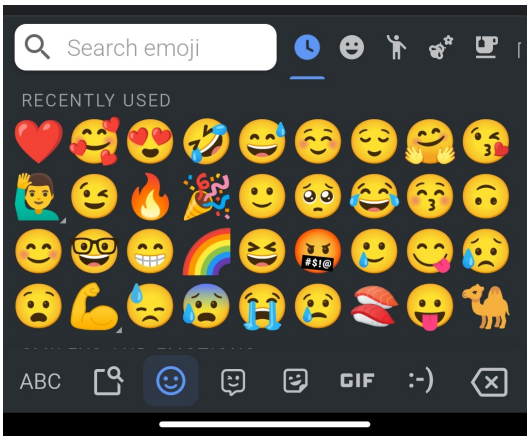

Figure 9: Expression History feature of GBoard

The expression history – `expression-history.db` – database represents the number of shares/appearances of a given emoji or emoticon. A share is considered valid simply by writing an emoji on the keyboard. From a forensic point of view, this data can be very useful to preliminarily check the type of conversation. For example, a more friendly conversation tend to have more emojis and a serious conversation much less.

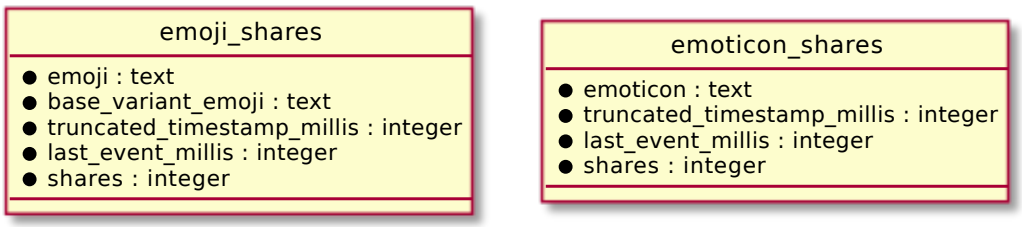

Figure 10: Tables from the database schema of the expression history

Figure 10 shows both `emoji_shares` and `emoticon_shares` that represents emojis and emoticons shares, respectively. Table 10 details the common fields of tables `emoji_shares` and `emoticon_shares`.

Field	Description
`last_event_millis`	Last time the entry was written (UNIX epoch in milliseconds)
`shares`	Number of shares of the entry
`truncated_timestamp_millis`	Identify the month and year of the entry (UNIX epoch in milliseconds)

Table 10: Common fields of expression history database tables

They share `last_event_millis` field which represents a timestamp in UNIX epoch formatted in milliseconds and `shares` fields that represents the number of shares of a specific emoji/emoticon. The field `truncated_timestamp_millis` seems to be a truncated timestamp formatted in UNIX epoch (milliseconds) that identifies the month and year of the entry.

Field	Description
`emoji`	Text representation of the emoji
`base_variant_emoji`	Base variant of `emoji`
`emoticon`	Text representation of the emoticon

Table 11: Fields description of emoji_shares and emoticon_shares table

Particularly, `emoji` field is a text representation of an emoji, `base_variant_emoji` field is a base emoji variant of `emoji` field, and `emoticon` field is a text representation of the emoticon. We hypothesize that the expression history database is linked to GBoard predictions for emojis and emoticon Ramaswamy et al., 2019.

3.5.2 *Other files*

Besides the aforementioned SQLite databases, GBoard private data encompasses other files. From the forensic point, one interesting file is `UserHistory.dict`. The file seems to contain multiple formatted data structures in which one of them is an empty Matching Algorithm with Recursively Implemented StorAge (MARISA) structure (Yata, 2020). Resorting to the `strings` utility from GNU binutils[7], we could figure out that the `UserHistory.dict` files holds the history of the user typed words. However, it was impractical to parse it in a consistent manner due to the complexity of the data structure.

[7] https://www.gnu.org/software/binutils/

Other meaningful forensic data sources includes the `translate_cache` folder, as shown next.

Translation Cache

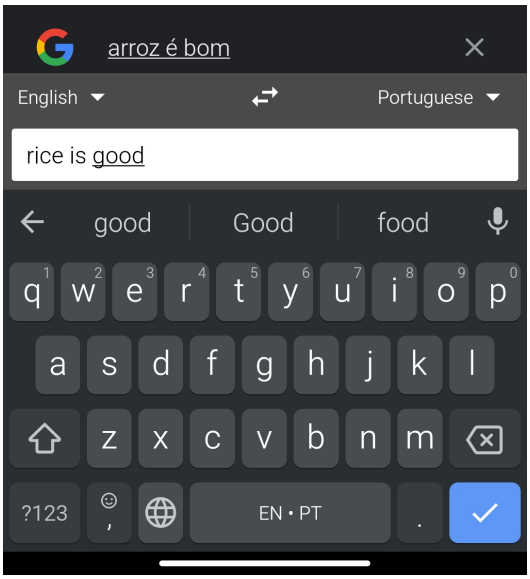

Figure 11: Translation feature of GBoard

Translation cache is a cache produced when the embedded translator in GBoard is used. By typing words in the translator textbox, GBoard queries, through the Internet, Google translator API and writes the translated word sequence. To speedup the process, it caches old translations to avoid repeating the query to Google Translator API. The forensic analyzer can take this as an advantage to fetch the typed words plus the translation and metadata associated with it, namely the date/time, as it comes from Google's servers.

```
1  https://translate.google.com/translate_a/single?client=ak&dt=t&dt=ld&dt=qca&dt=r⌋
   ↪  m&dt=bd&dj=1&sl=auto&tl=pt&hl=en&ie=UTF-8&oe=UTF-8&q=rice+is+good
2  GET
3  0
4  HTTP/1.1 200
5  19
6  content-type: application/json; charset=utf-8
7  date: Fri, 19 Mar 2021 00:30:08 GMT
8  content-disposition: attachment; filename="json.txt"; filename*=UTF-8''json.txt
9  content-encoding: gzip
10 Cache-Control: max-age=259200, max-stale=259200
11 OkHttp-Sent-Millis: 1616113807248
12 OkHttp-Received-Millis: 1616113807374
```

Listing 3: Reduced example of an HTTP request to the translation API

Listing 3 presents a shortened example of a cached Hypertext Transfer Protocol (HTTP) request. Although the requests are made over Hyper Text Transfer Protocol Secure (HTTPS), they are stored unencrypted, except for the initial Transport Layer Security (TLS) handshake. Metadata like `date` header and all the GET parameters make context reconstruction a lot easier.

```
{
   "sentences":[
      {
         "trans":"arroz é bom",
         "orig":"rice is good",
      }
   ],
   "src":"en",
   "confidence":1.0
}
```

Listing 4: Reduced example of an HTTP response from the translation API

Listing 4 presents a reduced example of the HTTP response from the translation API. With such metadata its possible to reconstruct the context with the translated text.

3.6 REARRANGED DATA

Because some data present in the databases are not directly fetchable, sometimes we needed to process it in order to get meaningful information. This way, it is possible to provide different perspectives of the information instead of just raw table data or merge information in one place.

In this section, we present the Structured Query Language (SQL) queries that we implemented to achieve three different perspectives of the data located in the training cache database. The goal is to obtain a high level view of the data, to yield what was effectively written on GBoard.

3.6.1 *Training Cache*

The entries in the *training cache* database is keystroke/word-based and not a continuous stream of data. Indeed, each entry is added upon a simple user task. For example, when a key is pressed a single character is registered in `tf_table`.

The same goes when a character is deleted as it gets recorded in the `d_table`. Additionally when an emoji or a whitespace is inserted, an entry in `tm_table` is added. For suggestions or when speech-to-text is used, the entire string is registered in `tf_table` rather than an entry for each corresponding character.

To give a meaningful high level view of the data, we reorganized its processing into three major subtables, each with its own perspective: History Timeline, Assembled Timeline and Processed History.

History Timeline is a simple table ordered by time that provides the most unitary level of the information stored in the training cache with a timestamped sequence attached with a boolean to represent if a certain sequence was deleted or added.

```
SELECT time, sequence, deleted, _timestamp FROM (
        SELECT datetime(_timestamp/1000, 'unixepoch') as time, f2, f3 as
        ↪  sequence, false as deleted, _timestamp
        FROM tf_table
        UNION
        SELECT datetime(_timestamp/1000, 'unixepoch') as time, f2, f5 as
        ↪  sequence, true as deleted, _timestamp
        FROM d_table
)
ORDER BY time, f2
```

Listing 5: SQL query to fetch History Timeline data

Listing 5 shows the SQL query performed in order to get the desired information. Information from the `f2` and `f3` fields of `tf_table` is used as inserted data and information from the `f2` and `f5` fields of `d_table` is used as deleted data. Then, we merge it together and order it by time and the provided accumulation sequence.

Assembled Timeline is an enhanced version of the History Timeline table with the data grouped by time and deletion. This way its possible to have multiple entries together based on time and the action performed, so all deleted and inserted actions that have a common timestamp get grouped in the same single entry.

Listing 6 shows the SQL query performed to get the Assembled Timeline data. It is based on the query in the Listing 5, but the information is grouped by time and the deleted boolean and then the sequence entry is concatenated.

Processed History is the most refined version of the training cache information. By combining the inserted and deleted information, it is possible to "diff" the deleted

```sql
SELECT time, group_concat(sequence, '') as sequence, deleted, _timestamp FROM (
        SELECT datetime(_timestamp/1000, 'unixepoch') as time, f2, f3 as
        ↪  sequence, false as deleted, _timestamp
        FROM tf_table
        UNION
        SELECT datetime(_timestamp/1000, 'unixepoch') as time, f2, f5 as
        ↪  sequence, true as deleted, _timestamp
        FROM d_table
)
GROUP BY time, deleted
ORDER BY time, f2
```

Listing 6: SQL query to fetch Assembled Timeline data

characters from the inserted information, yielding an attempt to reconstruct the actual phrase written by the user. This high level representation is possibly the most useful for forensic analysts. Although some information present in this table can be inaccurate. Indeed, inaccuracies can occur when the user moves the keyboard cursor while writing. As there is no recorded data tracking cursor movements, it becomes impossible to reproduce phrases with 100% accuracy. This problem can be circumvented by manually looking into the information present in the other generated tables and use common sense to match the words.

```sql
SELECT time, group_concat(sequence, '') as sequence, _timestamp
FROM (
        SELECT datetime(_timestamp/1000, 'unixepoch') as time, _timestamp,
        ↪  group_concat(f3, '') as sequence
        FROM tf_table
        WHERE (_timestamp, trim(f3, ' ')) NOT IN (
                SELECT _timestamp, f5
                FROM d_table
        )
        GROUP BY time, f4
        ORDER BY _timestamp, f2
)
GROUP BY _timestamp
```

Listing 7: SQL query to fetch Processed History data

Listing 7 shows the SQL query performed to fetch the Processed History data. This query diffs the information present in `tf_table` and filter any associated entry in the `d_table`. The information is then concatenated and sorted by time, as usual.

3.6.2 *Clipboard*

In GBoard, clipboard can have two types of information: text and images. Internally, the table `clips` stores every entry, but the actual content can be spread out in

different places. For text, the content is stored directly in the database, but for images, it is better for them to store it directly in the file system, instead of embedding it in the database file for performance reasons.

To extract the images, we need to look into the `uri` field in the database. If this field is not null, we can automatically know that the entry is a file. In Android, an URI pointing to a content starts with `content://` followed by the package name and the service that provides that content. For example, for GBoard, an URI pointing to a content present in the files data folder, starts with `content://com.google.android.inputmethod.latin.fileprovider/`. The rest is simple, just extract the remaining path, concatenate it with the root plus the `files` folder and read its content.

3.6.3 *Translation Cache*

When the user uses the translation feature of the keyboard two files are stored in the cache: an HTTP request file and an HTTP response file. Each file has a corresponding hash followed by a dot and either `0` (zero) for the request or `1` (one) for the response. This way, it is possible to associate both files in a single translation query. To rearrange the date in a timeline, the `Date` header is extracted from the request.

The response can be encoded in one of several formats. The encoding depends on the environment. The identification of the format is done through the `content-encoding` header field containing `gzip` or other values, acceptable in the HTTP standard (Mozilla, 2021).

In this chapter, we presented the databases of GBoard, focusing on the most meaningful ones, from a perspective of reconstructing phrases written by the user. In the next chapter, we focus on the Autopsy digital forensic software.

4

AUTOPSY

In this chapter, we briefly review the Autopsy digital forensic software, focusing on its main characteristics. The second half of the chapter details on one of our contribution to Autopsy: creation of an up-to-date package of Autopsy, so that it can be easily installed on Arch Linux and Debian, both Linux distributions.

4.1 THE SOFTWARE

Autopsy is an easy-to-use, Graphical User Interface (GUI)-based suite of software that allows efficient analysis on hard drives and smartphones. It has a plugin architecture that can be extended with add-on Java or Jython-based modules. Autopsy resorts to The Sleuth Kit Library[1] to access different types of file formats and file systems to extract data from storage. The Sleuth Kit is a collection of command-line tools and a C library that allows analysis of disk images and recovery of deleted files (Carrier, 2021d).

4.2 DESIGN

This digital forensics software follows in its design some key aspects:

- **Easy-to-use:** this software was designed to be intuitive. Wizards and navigation techniques help the user with everything he/she needs by making it easier to replicate steps without unnecessary reconfiguration (Contributors to Wikipedia projects, 2020). It tries to automate as much as possible, not only to facilitate its usage but to reduce the number of human errors that might occur otherwise (Carrier, 2021c).
- **Extensible:** one of the most important features is its plugin architecture. With this, any user can extend its capabilities by adding new functionality through non-native modules (Contributors to Wikipedia projects, 2020). Third-party

[1] http://sleuthkit.org/sleuthkit/

add-ons can be found in the official Module Github repository of Autopsy[2]. Users can also create their add-on modules[3]. Writing a custom module from scratch can lead to multiple advantages, even if a module that does the same already exists. This is because the user has the freedom to mold their creation the way it fits them better.

- **Fast:** a single device to analyse can hold many gigabytes of data. To be productive, the forensic practitioner must analyze the most amount of data in the least amount of time. Autopsy accomplishes this by running its tasks in parallel using multiple cores (except `sqlite3` modules because they do not support concurrency), hence shortening the time needed to analyze a drive's data. Although it might still take, in extreme cases, hours to finalize the analysis, the researcher does not have to wait that much time to start getting results. This is because Autopsy provides results in real-time as soon as they are found (Carrier, 2021a).

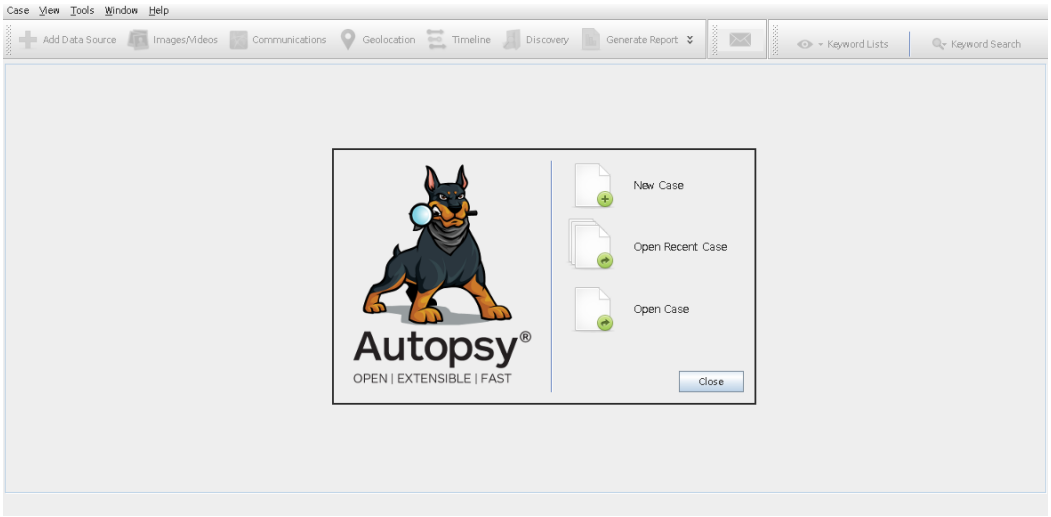

Figure 12: Autopsy startup screenshot

4.3 FEATURES

A great tool must have significant features that make it worth spending time learning and using it. Some of the features of Autopsy include the ability to collaborate with multiple forensic scientists in a large case, the capacity to analyze various format types, and an extensible report infrastructure. Collaborating with other researchers

[2] https://github.com/sleuthkit/autopsy_addon_modules
[3] http://www.sleuthkit.org/autopsy/docs/api-docs/

helps to spread the workload of a task making it more manageable to perform. Having a team working on a big case also ensures reliability on the outcome, as multiple examiners are looking for information and double-checking coworkers. Not only does the task get done a lot quicker, but it also becomes more trustworthy and less error-prone. Autopsy is capable of reading disk images, local drives, or a folder with local files. Different alternatives like these improve the usefulness of the software, as users stop feeling restrained to a single choice.

Although Autopsy also has graphical interfaces that display timeline analysis, it also provides a way to bind a custom report. By default HTML, Excel Spreadsheet (XLS), and Body file reports are available and are configurable to display the information the user deems more valuable. The investigator is allowed to generate multiple reports at a time. The report functionality can be extended by adding special-purpose report modules. Likewise ingest modules, report modules can be coded in Java and Jython programming languages. This way, Autopsy can be customized according to the needs of the forensic practitioners (Carrier, 2021b).

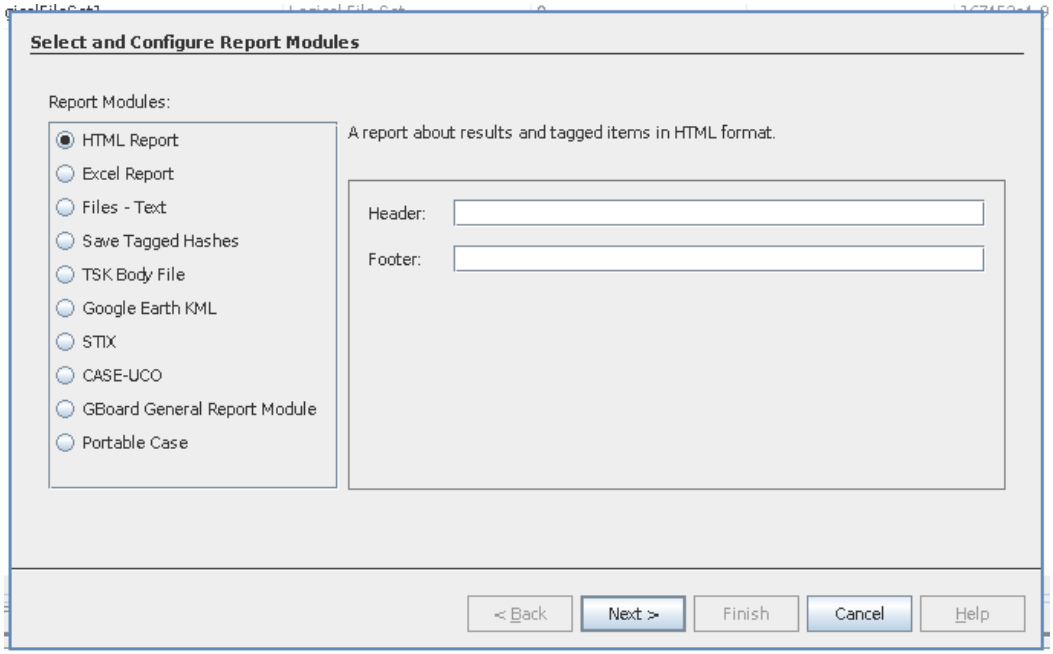

Figure 13: Autopsy interface to generate custom reports

4.4 LINUX PACKAGE

One of our goals was to have our GBoard4A software available for multiple OSes, namely Windows 10 and Linux, and to run under the latest versions of Autopsy.

Also, we found out that Autopsy packages for the two Linux distributions we used for development (Arch Linux and Debian) were not up-to-date, severely lagging behind Autopsy current version (4.18). One solution was manually building Autopsy on every new release. Although this was a valid answer, it is only a short-term solution that would not help future developers and users. We wanted to maximize the limited time we had to develop this project by making every decision count.

We decided to create a package with the latest version of Autopsy and a packager that generates packages for Arch Linux and Debian automatically. A package facilitates the groundwork required to have a workable Autopsy. The user simply needs its package manager to install the prebuilt package in its system.

4.4.1 *Arch Linux*

We had experience with packaging for Arch Linux distributions, so creating a package for Arch Linux was straightforward. The package is in the official Arch User Repository (AUR) and is named `autopsy-bin`[4]. Alternatively, the `autopsy` package[5], also stored in the AUR, can be installed.

4.4.2 *Debian*

For Debian, we had to opt for a different approach. Debian packages are a bit more complex than Arch Linux, in our eyes, and we wanted to ensure everything was well executed and portable. Initially, we explored the documentation of Debian packages and how they are configured and organized (Debian Wiki Team, 2021). The thought process was to create packages for both Arch Linux and Debian distributions and leave a well-document article in the LabCIF organization explaining how they work. This ensured that anyone could utilize the package and update it if needed. However, the structure of Debian packages quickly pushed us away from that path, and we decided to go with a tool to do it for us.

FPM

By using an extern tool we assured the package was always going to be created correctly and future updates by different investigators would not mess up the package

[4] https://aur.archlinux.org/packages/autopsy-bin/
[5] https://aur.archlinux.org/packages/autopsy/

structure. FPM is a command-line program design to help building packages. With this program creating platform-native packages is simple, more trustworthy, and more reliable. It can generate a package for a distribution based on another package from a different one and tweak existing packages (Sissel, 2021a). A usage example of FPM can be seen in figure 8.

```
1 fpm -s <source_type> -t <target_type> [list of sources]...
```

Listing 8: Building a package using FPM

The `source_type` field is where the package is coming from. A directory *(dir)*, a Rubygem *(gem)*, an RPM Package Manager (RPM) *(rpm)*, a Python package *(python)*, and a PHP: Hypertext Preprocessor (PHP) pear module *(php)* are examples of source types. The `target_type` field defines the output package form. Some common examples are RPM *(rpm)* and Debian *(deb)* (Sissel, 2021c).

A real-world example is in the examples folder of the FPM GitHub repository (Sissel, 2021b). Below, we can analyze the showcased Makefile 9 that creates the jruby package:

```
1  NAME=jruby
2  VERSION=1.6.1
3  DOWNLOAD=http://jruby.org.s3.amazonaws.com/downloads/$(VERSION)/$(TARBALL)
4  TARBALL=$(NAME)-bin-$(VERSION).tar.gz
5  TARDIR=$(NAME)-$(VERSION)
6
7  PREFIX=/opt/jruby
8
9  .PHONY: default
10 default: deb
11 package: deb
12
13 .PHONY: clean
14 clean:
15         rm -f $(NAME)-* $(NAME)_* || true
16         rm -fr $(TARDIR) || true
17         rm -f *.deb
18         rm -f *.rpm
19
20 $(TARBALL):
21         wget "$(DOWNLOAD)"
22
23 $(TARDIR): $(TARBALL)
24         tar -zxf $(TARBALL)
25
26 .PHONY: deb
27 deb: $(TARDIR)
28         fpm -s dir -t deb -v $(VERSION) -n $(NAME) -d "sun-java6-bin (>> 0)" \
29             -a all -d "sun-java6-jre (>> 0)" --prefix $(PREFIX) -C $(TARDIR) .
30
31 .PHONY: rpm
32 rpm: $(TARDIR)
33         fpm -s dir -t rpm -v $(VERSION) -n $(NAME) -d "jdk >= 1.6.0" \
34             -a all --prefix $(PREFIX) -C $(TARDIR) .
```

Listing 9: Package creation using FPM

To understand what the previous Makefile does, we need to look at the `name`, `version`, `download`, and `deb` fields. Starting from top to bottom, we first declare the `name` and `version` fields which are self-explanatory. Next, we define the Uniform Resource Locator (URL) that indicates the package version of jruby to download. This URL will download a `tar.gz tarball` containing the binary file needed for the package creation. Jumping to the `deb` section in line 26 is where the command-line expression for the Debian package creation is defined. The most important fields in this command are the same specified in the example above 8. The `source_type` is a directory located at `jruby-1.6.1`, and the `target_type` is a Debian package. The following arguments define the dependencies needed and where the installation is going to take place. Running the command bellow should now generate the Debian package.

```
1  make package
```

Now it can be installed using the dpkg tool.

```
sudo dpkg -i jruby-1.6.0.RC2-1.all.deb
```

Just like that, FPM created a package for JRuby with a pretty intuitive configuration.

```
% /opt/jruby/bin/jirb
>> require "java"
=> true
>> java.lang.System.out.println("Hello")
Hello
=> nil
```

Listing 10: JRuby installed using FPM running example

4.4.3 *Packager*

Now that we had a clear vision of the solution we wanted to go for, we needed to think about the next step. We could create the packages manually and updating them as needed, but this would not make it easier for future users. We wanted a solution that allowed updating the packages and making them available at the same time.

GitHub Actions

The choice was to make use of GitHub Workflows and GitHub Actions. GitHub Actions helps to automate tasks within the software life cycle. GitHub Actions are event-driven, which means that a series of commands are executed after a trigger. GitHub Workflows are automated procedures that are added to a repository. A Workflow has one or more jobs that can be scheduled or triggered by an event (GitHub Docs Team, 2021). We used GitHub Workflows as a Continuous Integration (CI) to build the Arch and Debian packages. The CI configuration is a YAML Ain't Markup Language (YAML) file, which makes it very intuitive to work with. When the CI runs, it builds both packages and when done, releases them as artifacts. These packages are downloadable, but each is only available for 90 days. Users should not rely on the artifacts built by the CI as they have an expiration time, and instead, they should go to the releases and pick the version they want to use.

Configuration

YAML is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted (Contributors to Wikipedia projects, 2021). Listing 11 shows a small portion of the CI configuration[6].

```yaml
jobs:
  arch-packages:
    runs-on: ubuntu-latest
    container:
      image: archlinux
      options: --privileged # Needed for building in clean chroot
      volumes:
        - /sys/fs/cgroup:/sys/fs/cgroup # systemd nspawn needs cgroups
    steps:
      - name: Install required dependencies
        run: |
          pacman -Syu --noconfirm
          pacman -S git --noconfirm --needed --noprogressbar
      - name: Clone sleuthkit-java build repo from AUR
        run: |
          git clone https://aur.archlinux.org/sleuthkit-java.git
          ↪ $HOME/sleuthkit-java
          (cd $HOME/sleuthkit-java; git checkout $PACKAGE_SLEUTHKIT_JAVA_SHA)
      - name: Build sleuthkit-java
        uses: FFY00/build-arch-package@v1
        with:
          PKGBUILD: $HOME/sleuthkit-java/PKGBUILD
          OUTDIR: /tmp/artifacts
      - uses: actions/upload-artifact@v2
        with:
          name: arch-packages
          path: /tmp/artifacts/*.pkg.tar.zst
      - name: Clone autopsy-bin build repo from AUR
        run: |
          git clone https://aur.archlinux.org/autopsy-bin.git $HOME/autopsy-bin
          (cd $HOME/autopsy-bin; git checkout $PACKAGE_AUTOPSY_BIN_SHA)
      - name: Build autopsy-bin
        run: |
          chown user -R $HOME/autopsy-bin
          cd $HOME/autopsy-bin
          extra-x86_64-build -- -U user -I "$(find "/tmp/artifacts" -name
          ↪ 'sleuthkit-java-*.pkg.tar.zst')"
          mv *.pkg.* /tmp/artifacts
      - uses: actions/upload-artifact@v2
        with:
          name: arch-packages
          path: /tmp/artifacts/*.pkg.tar.zst
```

Listing 11: Packager - arch-packages job

The syntax of a YAML file is pretty intuitive. In this sample, we can see the job of the `arch-packages`. The software image is defined, and then the steps are executed accordingly. The configuration file is structured to be scalable. If in the future an extra distribution is needed, then it can be added with another job.

[6] https://github.com/labcif/autopsy-packager/blob/main/.github/workflows/packaging.yml

Arch package

The Arch Linux package generated by the CI bases itself on the `autopsy-bin` AUR package. As shown in Listing 12, the step execution clones the autopsy-bin package and builds it manually. This is the same process as installing the package locally using an AUR helper.

```
- name: Clone autopsy-bin build repo from AUR
run: |
    git clone https://aur.archlinux.org/autopsy-bin.git $HOME/autopsy-bin
    (cd $HOME/autopsy-bin; git checkout $PACKAGE_AUTOPSY_BIN_SHA)
- name: Build autopsy-bin
run: |
    chown user -R $HOME/autopsy-bin
    cd $HOME/autopsy-bin
    extra-x86_64-build -- -U user -I "$(find "/tmp/artifacts" -name
    ↪  'sleuthkit-java-*.pkg.tar.zst')"
    mv *.pkg.* /tmp/artifacts
```

Listing 12: Packager - Arch package build

Debian package

As previously stated, the Debian package is generated using FPM. To keep the complexity much shorter, FPM supports receiving a package as its `source_type`. Our implementation uses this feature. GitHub Workflows allows having a job depending on another. This means we can have the Debian job depending on the Arch job, allowing the Arch package to be built first. Then in the Debian job, we can utilize that package to generate the other as seen in Listing 13.

```
- name: Install Dependencies
run: |
    apt-get update -y
    apt-get install -y --no-install-recommends build-essential rubygems ruby
    ↪  ruby-dev curl git zstd locales
    sed -i -e 's/# en_US.UTF-8 UTF-8/en_US.UTF-8 UTF-8/' /etc/locale.gen
    dpkg-reconfigure --frontend=noninteractive locales
    update-locale LANG=en_US.UTF-8
    gem install --no-document fpm
    fpm -t deb \
        -n "autopsy" \
        -v "${PACKAGE_AUTOPSY_BIN_VERSION}" \
        --iteration "$PACKAGE_SLEUTHKIT_JAVA_REL" \
        --no-auto-depends \
        -d "openjdk-8-jre" \
        -d "sleuthkit-java (>= ${PACKAGE_SLEUTHKIT_JAVA_VERSION})" \
        -d "testdisk" \
        -d "openjfx" \
        -a amd64 \
        -s pacman autopsy-bin-${PACKAGE_AUTOPSY_BIN_VERSION}-${PACKAGE_A
        ↪  UTOPSY_BIN_REL}-x86_64.pkg.tar.zst
    mkdir -p /tmp/artifacts
    mv *.deb /tmp/artifacts
```

Listing 13: Packager - Debian package build

Artifacts

The built packages are then made available as artifacts in the Github Actions tab. Every time the CI runs, it outputs new artifacts with the current version in the configuration file. This means that if different runs can provide identical package versions. Artifacts are available for download if not expired, as stated above. Figure 14 shows all packaging workflow runs. The most recent run is at the top of the list.

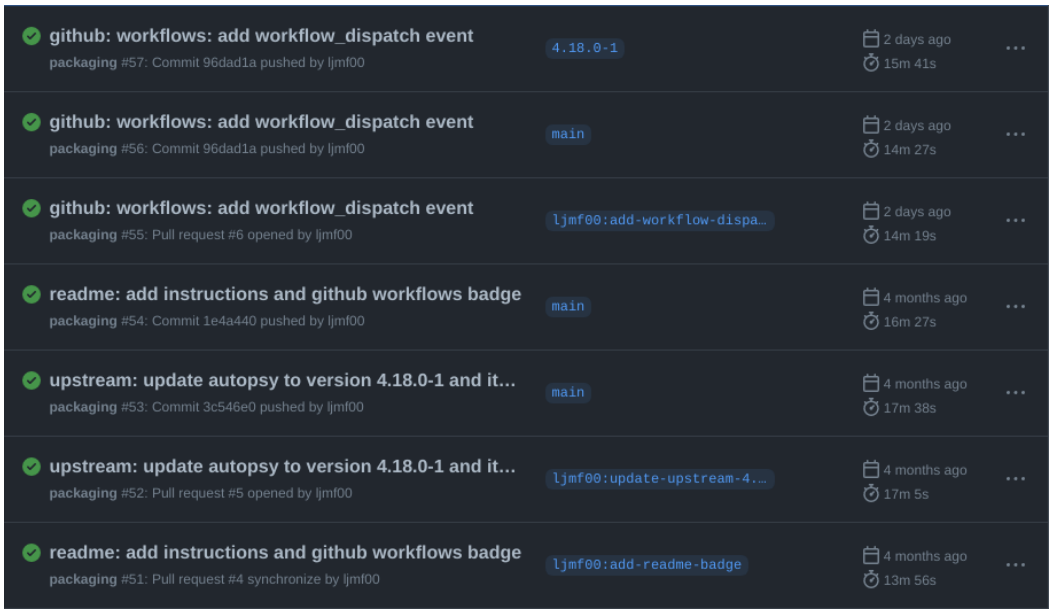

Figure 14: Packager CI runs

The latest run contains the information shown in Figure 15 below. On this page, users can analyze the logging of the selected CI run. The displayed information is very straightforward. On the left, we can see the jobs ran, and in the middle, the order they executed. The `arch-packages` job ran first, and the `build-run` ran last. This information follows the CI configuration analyzed above. Clicking on each, we can explore step by step how the execution performed. Each step contains a log detailing every command and action made.

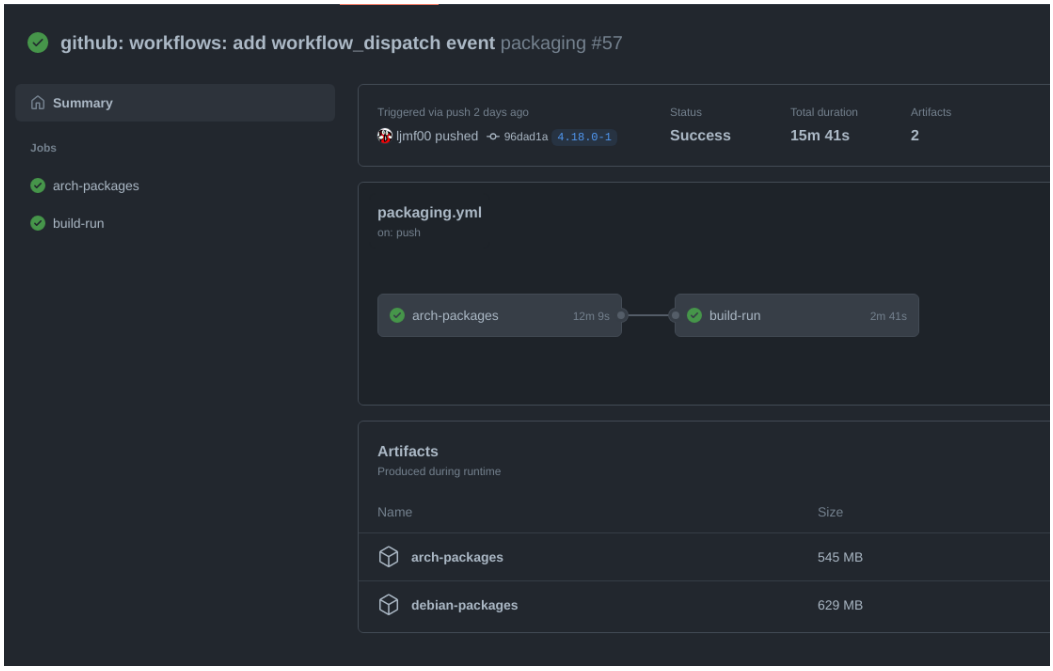

Figure 15: Packager CI runs

Lastly, the artifacts are placed at the bottom. If they are not expired, they are available for download. Simply clicking on the package downloads it.

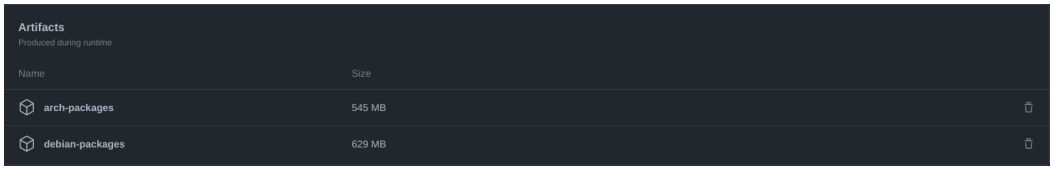

Figure 16: Packager CI runs

Releases

Although artifacts are a good way of getting the most updated package versions (according to the configuration file), they are not reliable. Exploring the releases

separator should be the priority. In the releases menu, not only is potentially available the latest version but also previously released versions. For instance, if the version needed was 2 or 3 minor versions older and was available as a release, it could be downloaded and installed like any package. The release for version `4.18.0-1` is shown in Figure 17.

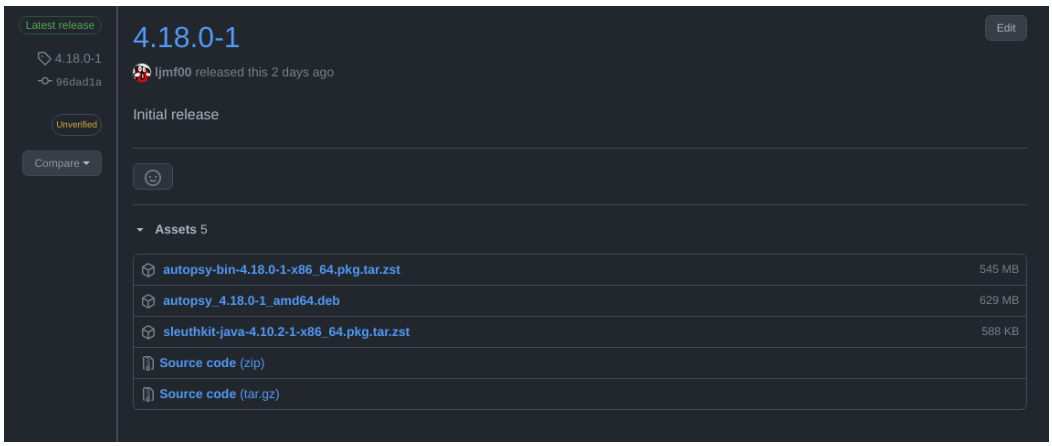

Figure 17: Packager release 4.18.0-1

In this chapter, we dealt with Autopsy, a well-known digital forensic open source software. Specifically, we first reviewed the main characteristics of Autopsy, and then described our contribution in providing packages for an easy installation of Autopsy under Arch Linux and Debian distributions. For this purpose, we detailed the followed methodology and the needed GitHub configurations. In the next chapter, we present our GBoard4A software tool and its coupling Autopsy software modules.

5

GBOARD APPLICATION AND MODULE

In this chapter, we describe our developed GBoard standalone application called GBoard4A, the integration with Autopsy digital forensics software and the development of two modules for Autopsy.

Although digital analysis tools might be complex to implement, the end-user should not experience complexity when working with the software. The usage of any software tool should be simple, user-friendly, and effective. A software tool conceptualized with this in mind provides to any user a better working experience. It also improves productivity by performing daily work tasks without much trouble, no matter what level of knowledge the user has outside their working environment. Providing this was something we had in mind when planning how to deliver the final product.

5.1 STANDALONE APPLICATION

For this project, we decided to develop a standalone Command-line Interface (CLI) application separated from the Autopsy module. This decision came along with the advantage of future extensibility and reusability of the application with other forensics tools and the philosophy of software isolation to achieve better software quality by performing more isolated testing.

Because of the constant evolution of the software, extensibility and reusability is a key feature to reach a faster software growth and it is part of a good system design principle to effortlessly extend a piece of software to different systems and environments (Johansson and Löfgren, 2013). To achieve that we created a simple command-line API. With that approach, it is possible to any software integrate our application and build a module, easily. This is particularly useful for our usecase because this work includes the development of two different modules for the Autopsy software.

Software isolation is part of the UNIX philosophy for creating minimalist and modular software and it is very important for the development of quality software.

Quoting Salus (1994), "Write programs that do one thing and do it well". By creating a sub-program only to fetch GBoard information, it is possible to detect errors more easily due to the reduced number of tasks it needs to perform and therefore test the software behaviour in a more effective way. This is important because software tends to become complex and hard to reproduce in a monolithic scenario.

5.1.1 *Operating modes*

The standalone application has two distinguishing modes for different purposes: root analysis, which performs a full analysis from a given GBoard root path, and multiple path analysis which can perform multiple analyses on different files or folders by giving multiple entries.

Root path analysis is useful when a full dump of GBoard internal folder is done and it is intended to preform a full forensic analysis. This is mostly used with the Autopsy modules to report the desired results.

Multiple path analysis, on the other hand, is useful when a partial dump is available or the GBoard files are not organized in the same way GBoard does internally. This is possible with an auto-detection mechanism that seeks which file is. Additionally, this is particularly useful when the intention is to only analyse a single file.

```
Some information about the program

-r --root-dir GBoard root directory analysis (must be used alone)
-d      --dir GBoard directory analysis
-f     --file GBoard file analysis
-t     --type Output format type (default: json)
-o   --output Output file path of analysis report
-v  --verbose Print extra information on the analysis
-h     --help This help information.
```

Listing 14: Ouput of the help information of the standalone application

Listing 14 shows the output of the standalone application when `--help` argument is given. To run a root path analysis, the argument `--root-dir`/`-r` should be present followed by the path of the GBoard internal folder dump. For multiple path analysis, either `--file`/`-f` or `--dir`/`-d` should be present followed by the file or directory path to analyse. All the mentioned arguments can be provided multiple times, and root path analysis is only allowed when `--root-dir`/`-r` is used alone.

To be able to switch between different analysis reports is possible to specify a report type with `-t`/`--type` and a custom output path instead of standard output, which is the default, with `-o`/`--output` followed by an output file path. The available report types are `json` for JavaScript Object Notation (JSON) format and `html` for HTML format. Additionally it is possible to run the application with `-v`/`--verbose` to print extra information for debugging purpose.

5.1.2 *Output Report Types*

This application implements two types of reporting, useful for different situations: JSON and HTML.

JSON Type

JSON and JSON-like data structures are common and widely used by software industry for bridging two applications or services and still maintain a readable file format. These data structures are often implemented part of a programming language standard library because of its popularity (*JSON* 2021). The JSON reporter is particularly useful for modularization. It provides a way to easily integrate with other tools by using an easy language to parse.

```
{
        "rootPath": "/location/to/gboard-folder",
        "dictionaries": [
                {
                        "path":
                        ↪  "./gboard-folder/databases/PersonalDictionary.db",
                        "entries": []
                }
        ],
        "trainingcache": [
        ]
}
```

Listing 15: Reduced example of the JSON output report

Listing 15 shows a reduced example of the JSON output report. Only two data sources are described in the example, but `expressionHistory`, `translateCache` and `clipboard` are also available. Additionally, `rootPath` describes the root path, in case a full analysis is performed and data sources can have multiple objects in case a multiple path analysis is performed, although the same format used for both

types of analysis to naturally simplify the result parsing and the overall application maintenance.

HTML Type

HTML is a widely used markup language standardized for web applications to render text and images. Sometimes forensics analyzers wants a detailed custom report for more in-depth analysis. This format is useful to display textual and graphical information in a pleasant way and it is used by this project to generate a printable interactive report.

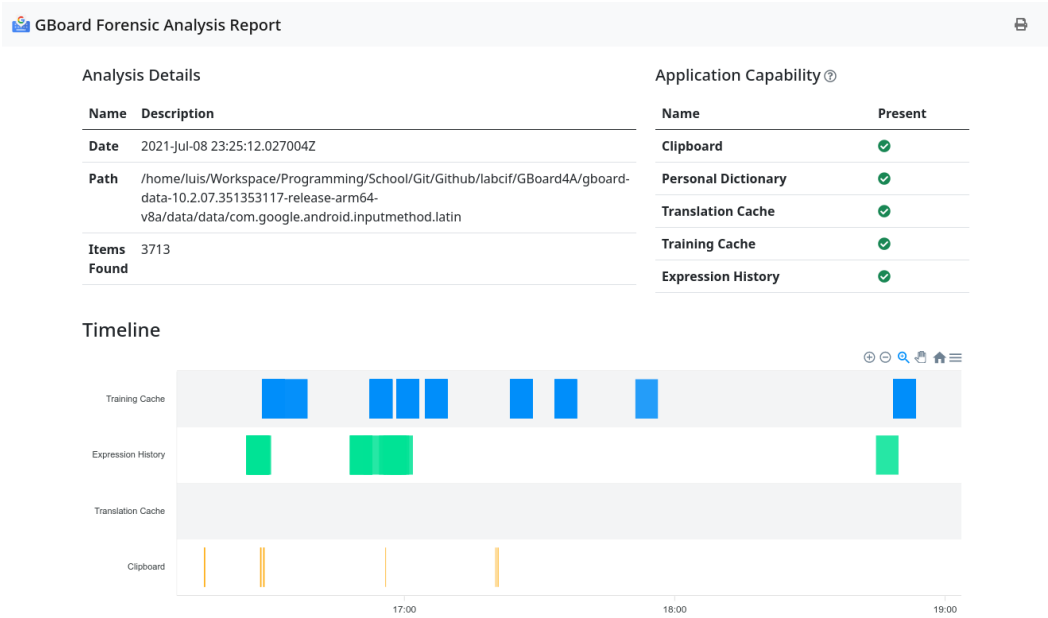

Figure 18: Generated report of a GBoard analysis

Figure 18 shows an example output of a generated report from a GBoard analysis.

This report is easily searchable, responsive, so it fits on every screen, it is adjustable for printing purposes, by expanding all the scrollable boxes and it is portable, so it has everything bundled in a single file. Each component is designed to be opened and saved individually by expand it with a button present in the right corner and additionally, every chart can be saved in Scalable Vector Graphics (SVG), Portable Network Graphics (PNG) or Comma-separated values (CSV). This is designed specifically to help the forensic analyst to have a good user experience and be able to export information quickly.

Firstly, on the left, it has a table with some brief information about the analyzed content that includes the date of the performed analysis, the full path of the GBoard root folder, if it exists and a counter with the number of items found. On the right, it has another table that describes if a certain capability was found.

Figure 19: Example of a generated timeline

Figure 19 shows the scrollable timeline with a navigation feature to shrink and expand it by zooming in and out, moving it around with the "hand" icon, restore the default view with the "home" icon and an hamburger menu to export it. All this features can be used with common keyboard and mouse shortcuts and also the buttons present in the right corner. This is useful to seek when the victim was more activity and compare side-by-side each found capability to see what was used at some point in time.

For each capability, one or more tables is shown. On the left of each table, a little question mark is present so the forensic analyst can hover and read more about what is the propose of that table. On the right, a button is present to expand the table in a new window.

Entries

Time	From	To	Original	Translated	Raw
30 May 2021 20:36:12	pt	en	O	O	Request Response
30 May 2021 20:36:14	pt	en	Ojulio	Ojulio	Request Response
30 May 2021 20:36:34	pt	en	O julio gosta de carpetes amarel	Julio likes yellow carpets	Request Response
30 May 2021 20:36:26	pt	en	O julio gosta d	Julio likes it	Request Response
30 May 2021	pt	en	atu	act	Request

Figure 20: Example of the Translation cache table

Figure 20 shows an example of the generated table for translation cache with the described features.

For easier visualization of the expression history, a histogram chart is provided, so the forensic analyst can visually compare what is the most shared emoji/emoticon and therefore evaluate at first a glance if the conversation is a more friendly one or not.

In the chart, it is possible to hover and see the corresponding value of the selected entry and on the top corner, a hamburger menu is present to export the chart.

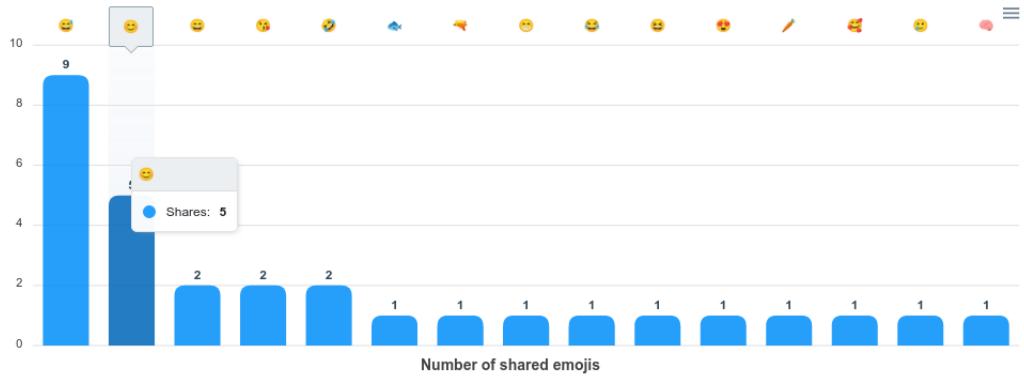

Figure 21: Example of a generated histogram chart

Figure 21 shows an example of a generated histogram chart based on the Expression History information.

5.1.3 *Internal architecture*

To achieve easier modularization and code scalability, we decided to divide the program in four modules: the gatherers, the models, the reporters and the analysis module. This way, it is possible to write features without disrupting the entire program design. Separating this logic from the rest of the code is crucial to add support for future versions of the application.

The gatherers module contains all the gatherers for each implemented data source. A gatherer contains logic to fetch data from a specific data source. For example, the training cache data source has a corresponding `TrainingCacheGatherer` that fetches everything related to training cache related databases.

The models module contains the corresponding serializable Data Transfer Object (DTO) structures for each data source.

The reporters module has the logic to generate each implemented output report of the analyzed data.

The analysis module is responsible for detecting data sources, load the appropriate data source gatherer and merging the gathered information into a serializable structure. This module is seen as a bridge for the models and the gatherers. File detection includes detecting files by their payload signature and/or check their internal structure. This module is useful to detect different file versions in newer or older versions of GBoard. This way, file detection is separated from the gatherers.

```
00000000: 5351 4c69 7465 2066 6f72 6d61 7420 3300  SQLite format 3.
00000010: 1000 0101 0040 2020 0000 0012 0000 0005  .....@  ........
00000020: 0000 0000 0000 0000 0000 0002 0000 0004  ................
00000030: 0000 0000 0000 0005 0000 0001 0000 0004  ................
00000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Listing 16: Example of the first bytes present in a SQLite database

For example, Listing 16 show the first bytes of a SQLite database. In this example the file gatherer compares the the first 16 bytes with the SQLite3 file signature to check if it is an SQLite database or not and then proceed to detect the internal structure of the database. Because every SQLite database from Android has a table with metadata, it is possible to check if it is a GBoard database by checking the existance of that table and then some additional structure, if needed.

```
SELECT name FROM sqlite_master
WHERE LOWER(type) IN ('table','view')
AND LOWER(name) = 'android_metadata'
```

Listing 17: Query to check existence of Android metadata table

Listing 17 show the query executed to detect the `android_metadata` table.

5.2 AUTOPSY INTEGRATION

One of the main goals of this project is to integrate the created standalone application with the Autopsy software by creating third-party modules. For this project we developed two different modules for Autopsy: an Ingest Data Source Module and a General Report Module, both communicating with the standalone application.

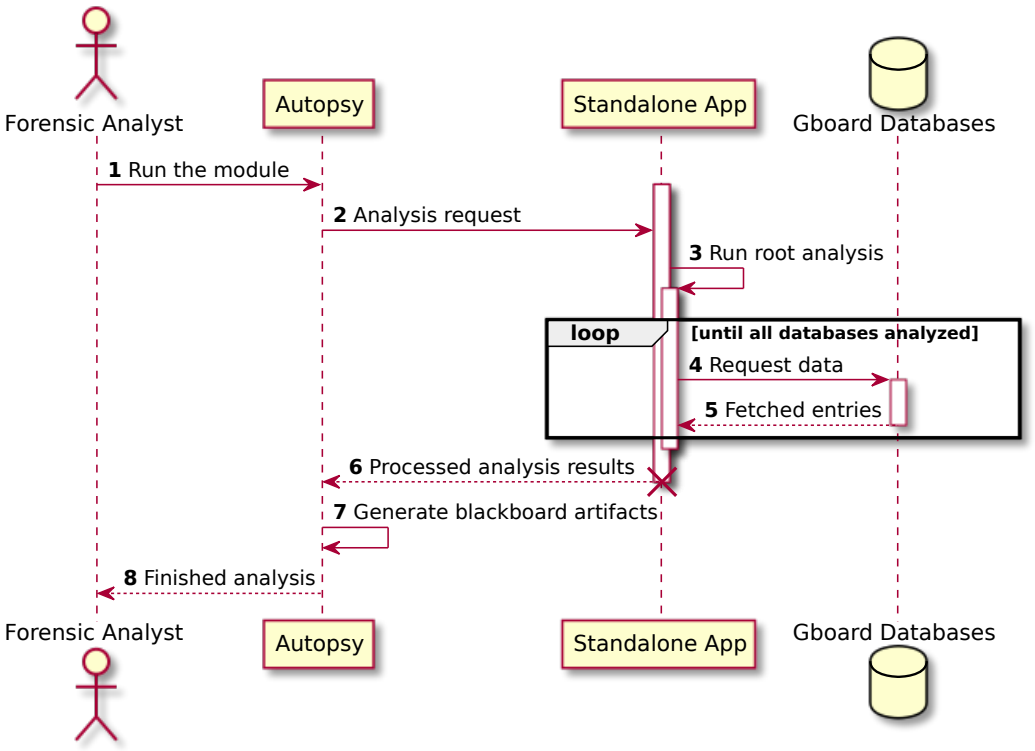

Figure 22: Sequence diagram of the Autopsy modules

Figure 22 shows a sequence diagram of the Autopsy modules interaction with the standalone application.

5.2.1 *Ingest Data Source Module*

The Ingest Data Source Module is intended to publish the data gathered by the standalone application to the blackboard functionality of the Autopsy. To do this, we used the generated JSON output of the standalone application as a bridge format and parsed it on the Jython side.

In order to publish the information on the blackboard, we needed to first create some custom artifacts to fit our extracted content, except for the clipboard content, which the Autopsy already has a built-in blackboard type. Artifacts such as Assembled timeline, emojis and emoticons expression history, history timeline, processed history, translation cache and personal dictionary were created. Those blackboard artifacts directly match the previously created models in the standalone application.

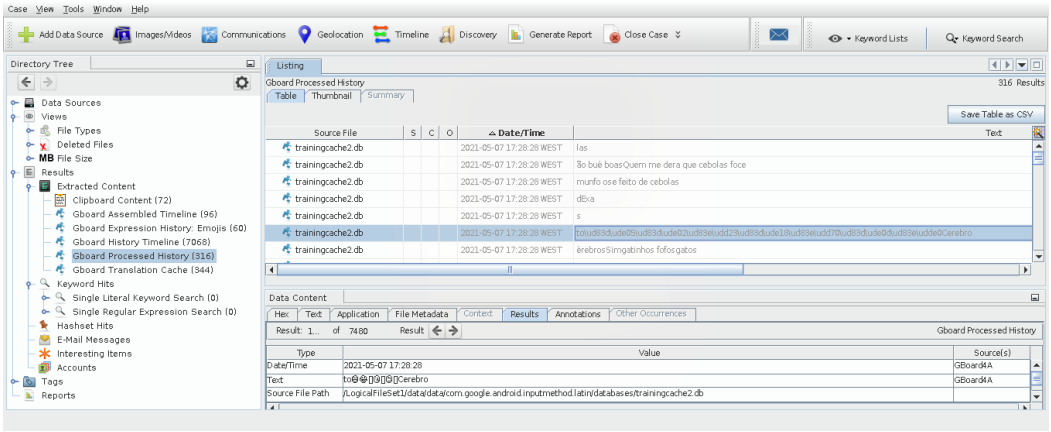

Figure 23: Example of extracted content generated by the GBoard Autopsy Ingest Module

Figure 23 shows an example ouput of extracted content generated by the GBoard Autopsy Ingest Data Source Module on a full analysis.

5.2.2 *General Report Module*

The General Report Module is intended to generate a report of any kind. In our case, this module generates an HTML formatted report based on the information gathered by the standalone application. To generate this report, we simply call the standalone application using the HTML reporter. Unlike Ingest Data Source Module, this module doesn't parse the output of the standalone application and rather directly maps it to a file specified by Autopsy.

5.3 TESTS AND USE CASES

During the development of this project various tests were performed to guarantee reliability and stability across versions both for GBoard data gathering and for our standalone application behaviour. To achieve this we made manual and automated tests continuously during the development.

5.3.1 *Manual testing environment*

All the development of this work was made primarily in Linux, using Arch Linux, although, some tests were made in Windows to garantee that our tool runs as expected with the Autopsy integration. We couldn't test manually macOS as the release for Autopsy in `brew` package manager was very outdated.

Initially, our main testing environment were based on the docker we created[1] with Android Emulator bundled with an Android 30 API-based image. This image, prebuilt by Google, ships a preload version of GBoard. During the development of the project, we discovered that this image unable the installation of new versions of GBoard because the application signature of preload versions missmatch with the release branch.

To proceed with this we needed to either make the `/product` android partition writable or have a privileged image with root access bundled with Google APIs to have the release version of GBoard. None of the above seem to be practical and we decided to move in a different direction by leaving Android Emulator and rooting our own devices.

With the devices properly rooted with the Magisk software[2], it was now possible to install different versions of GBoard. This was a good choice, as a physical device increased the number of collected data and simplified the manipulation of GBoard because the device was been extensively used. For completion, some versions of GBoard was also tested on Genymotion.

[1] https://gist.github.com/ljmf00/5db17e4cca66e762d51a86bae0712652
[2] https://github.com/topjohnwu/Magisk

5.3.2 *Automated Continuous Integration*

To extensively test the internal and external behaviour of our standalone application we decided to implement an automated test suite in our development pipeline. To do this, we used the recent tool from Github, called Github Actions[3]. This tool allow developers to create custom CI/Continuous Delivery (CD) workflows to automate the software development workflow and provide better software quality. (GitHub Docs Team, 2021)

This allowed us to automate our CI/CD pipeline by compiling the code, running our test suite and deploy artifacts automatically in Windows, Linux and macOS environments, at the same time. Combining this with a Pull Request (PR) development strategy we continuously tested our application when adding new features or fixing bad application behaviour.

Unfortunately, Autopsy doesn't provide an easy way to properly do automatic integration testing with third-party modules, so we didn't consider implementing that.

5.3.3 *Results*

In Chapter 3 we briefly described the results of our tested GBoard versions, especially in section 3.2 the detailed testing environment. For this project, we fetched, if available, information from clipboard management, training cache, translation cache, expression history and personal dictionary.

	Translation Cache	**TC**[1]	**Clipboard**	**EH**[2]	**PD**[3]
9.4.11.312687073	Yes	Yes	Yes	No	Yes
10.0.02.338070508	Yes	Yes	Yes	Yes	Yes
10.2.07.351353117	No	Yes	Yes	Yes	Yes
10.4.04.361808908	No	Yes	Yes	Yes	Yes
10.5.03.367007960	No	No	Yes	Yes	Yes

[1] Training Cache
[2] Expression History
[3] Personal Dictionary

Table 12: Gboard – tested capabilities by version

[3] https://docs.github.com/en/actions

Table 12 shows the tested capabilities by each GBoard version previously enumerated.

Some versions were more extensively tested and produced more results, but this information may vary from user to user. More active users may produce better results, although, if the user happens to have third-party application to periodically clean the applications cache or manually force cleaning it, some information is gone. If this is the case, for example, translation cache may not be captured.

Contrary to what we observed, training cache is only wiped on data cleaning, rather than cache cleaning. Although, this is still considered a cache, due to the ring buffer behaviour explained previously and the fact that this database is limited to 1500 entries. In practice, the forensic analyst will only get the most recent 1500 results from that capability.

The other application capabilities seem to consider application data. If the user deletes application data, because it also includes deleting cache, everything valuable, forensically speaking, is lost, unfortunately.

5.3.4 *Real-world Results*

To present more accurate and real-world results, we decided to extensively test GBoard for three months with daily usage and more normal conversations providing a more natural and effective usage of the keyboard. In total, 6379 entries has been collected.

Despite the content having the same structure, we detected some anomalies on the expression history behaviour. For a long run, expression history started do duplicate entries in the database, which leads to two emojis with different numbers in shares.

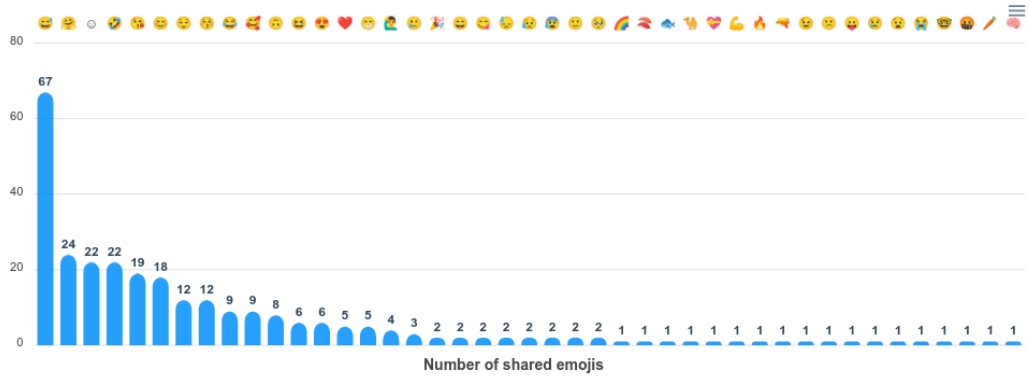

Figure 24: Real-world example of an expression history chart

Figure 24 shows a real-world example of an expression history chart. As shown, it is clear that the user uses a certain emoji very frequently. Moreover, it is possible to quickly understand that conversations are highly expressive and emotional.

It is interesting to see that every information from the beginning of the installation is retained, except the training cache, as expected. The application or Android itself seems to not have a mechanism to internally wipe cache or old information periodically, especially the translation cache, which is specifically in the `cache/` folder. Although, it needs more time to effectively prove that.

Another interesting fact discovered only with real-world usage, is the fact that the user seems to delete and move the cursor a lot. This is a natural behaviour in a keyboard like GBoard, as people tend to write a lot of words incorrectly, by mistake. Sometimes, incorrect order is also present, as timestamps in the training cache are not so accurate. The combination with processed history and assembled timeline is very useful for these situations.

6

CONCLUSION

Technology is constantly evolving and people choose unconsciously to use their smartphone for daily tasks and in today's society this tool is inevitably a tool for work usage. Therefore, forensic analysis on these devices has an exceptional interest. This work explored the potential of GBoard, in a forensic perspective by identifying the usability of the most known GBoard features for this purpose.

As described throughout this report, we developed an application to tackle the most interesting forensic assets of GBoard and integrated the reporing with the Autopsy software.

The application allows to extract valuable clipboard content, take out expression and emotional status of the written conversations, pull out partially or an entire conversation throughout the caching mechanisms for translation or training models and bring out shortcuts of key words potentially valuable on criminal cases.

During the development of this project, some difficulties were found essentially on reverse engineering techniques. Due to the lack of knowledge on this topic, some complex internal data structures were left behind such as the `UserHistory.dict` file. Other difficulties were found in replicating other GBoard dumps such as seen in Khatri (2021) were also impractical on our side due to insufficient awareness about the testing environment.

6.1 FUTURE WORK

After finishing our development and testing GBoard we discovered some tasks to tackle and some improvements to take after analysing the results, including doing some extensive tests with Google Pixel devices, applying Artificial intelligence on some processed results for better correctness and research more in the reverse engineering of some GBoard parts.

6.1.1 *Google Pixel testing*

As described by Khatri (2021) some prior work has been done about GBoard in a forensics perspective. As an improvment, we tried to replicate the work and develop from there, but unfortunately, the behaviour we got was brutally different. We searched about Pixel phones, and their major selling point is their unique features that Google make on purpose. One of them interacts directly with GBoard and is advertised as exclusive to Google Pixel phones (TECH, 2020). Therefore, we believe that our different results has something to do with Pixel devices and it is worth to investigate GBoard with them.

Furthermore, because this results conflicts with all our tested dumps such as database content missmatch, we decided to not implement this behaviour unless properly tested by knowing specifically what triggers this behaviour and find a mechanism to differentiate those dumps more easily and consistently parse it.

6.1.2 *Artificial Intelligence usage*

From our real-world results, we found out that proccessed history can be difficult to interpret by the forensic analyst and manually recurring to other tables, such as assembled history, can be a tedious process and sometimes the foresic analyst cannot find a pattern easily.

In this case, using Artificial Intelligence could potentially help find those patterns and, in the end, provide a sorted and more clear version to the forensic analyst to look for, improving the overall analysis of digital forensic cases.

6.1.3 *Reverse engineering*

Due to lack of our knowledge in reverse engineering of complex data structures, some work were left behind. For those who have interests in this area, research in this topic could be beneficial to understand and find more ways to dump written conversations more easily using `UserHistory.dict` file. Other files may also contain useful information but seem to be structured in the same format of `UserHistory.dict` such as files like `Email.dict` and `Contacts.dict`, potentially provide useful information such as frequently used emails and contacts.

BIBLIOGRAPHY

Carrier, Brian (July 2021a). *Autopsy: Fast Results*. [Online; accessed 9. Jul. 2021]. URL: http://www.sleuthkit.org/autopsy/fast.php.

— (July 2021b). *Autopsy: Features*. [Online; accessed 9. Jul. 2021]. URL: http://www.sleuthkit.org/autopsy/features.php.

— (July 2021c). *Autopsy: Intuitive*. [Online; accessed 8. Jul. 2021]. URL: http://www.sleuthkit.org/autopsy/intuitive.php.

— (July 2021d). *The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools*. [Online; accessed 8. Jul. 2021]. URL: http://www.sleuthkit.org.

Çehreli, Ali (2015). *Programming in D*. [Online, accessed 2021-06-25] http://ddili.org/ders/d.en/. Draft2Digital. ISBN: 978-1-519-95441-1.

Chung, Ching Chang (Dec. 2019). *Android Studio 3.5.2 Emulator. How to View list of applications*. [Online; accessed 12. Jul. 2021]. URL: https://stackoverflow.com/questions/59188013/android-studio-3-5-2-emulator-how-to-view-list-of-applications.

Contributors to Wikipedia projects (Dec. 2020). *Autopsy (software)*. [Online; accessed 8. Jul. 2021]. URL: https://en.wikipedia.org/w/index.php?title=Autopsy_(software)&oldid=997205652.

— (June 2021). *YAML*. [Online; accessed 11. Jul. 2021]. URL: https://en.wikipedia.org/w/index.php?title=YAML&oldid=1031049392.

Cranz, Alex (May 2021). *There are over 3 billion active Android devices*. [Online; accessed 13. Jul. 2021] https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021.

Curry, David (May 2021). *App Store Data (2021)*. [Online; accessed 13. Jul. 2021] https://www.businessofapps.com/data/app-stores/.

Davenport, Corbin (n.d.). *Gboard passes one billion installs on the Play Store*. [Online, accessed 2021-06-27] https://www.androidpolice.com/2018/08/22/gboard-passes-one-billion-installs-play-store/.

Dean, Brian (Apr. 2021). *How many people use social media in 2021?* [Online; accessed 13. Jul. 2021] https://backlinko.com/social-media-users.

Debian Wiki Team (July 2021). *Packaging - Debian Wiki*. [Online; accessed 10. Jul. 2021]. URL: https://wiki.debian.org/Packaging#Introduction_to_Debian_Packaging.

Department, Statista Research (Mar. 2021). *Average number of new Android app releases via Google Play per month from March 2019 to February 2021*. [Online; accessed 13. Jul. 2021] https://www.statista.com/statistics/1020956/android-app-releases-worldwide/.

Esch, Daan van et al. (2019). *Writing Across the World's Languages: Deep Internationalization for Gboard, the Google Keyboard*. arXiv: 1912.01218 [cs.HC].

GitHub Docs Team (July 2021). *Introduction to GitHub Actions*. [Online; accessed 11. Jul. 2021]. URL: https://docs.github.com/en/actions/learn-github-actions/introduction-to-github-actions.

Google (July 2021). *Gboard*. [Online; accessed 17. Jul. 2021]. URL: https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin.

Google Developers (June 2021). *Android Debug Bridge*. [Online; accessed 26. Jun. 2021]. URL: https://developer.android.com/studio/command-line/adb.

Hard, Andrew et al. (2019a). *Federated Learning for Mobile Keyboard Prediction*. arXiv: 1811.03604 [cs.CL].

— (2019b). "Federated Learning for Mobile Keyboard Prediction". In: arXiv: 1811.03604 [cs.CL].

Johansson, Niklas and Anton Löfgren (Apr. 2013). "Designing for Extensibility: An action research study of maximizing extensibility by means of design principles". In: [Online; accessed 7. Jul. 2021]. URL: https://gupea.ub.gu.se/bitstream/2077/20561/1/gupea_2077_20561_1.pdf.

JSON (June 2021). [Online; accessed 8. Jul. 2021]. URL: https://www.json.org/json-en.html.

Kastrenakes, Jacob (Jan. 2021). *Apple says there are now over 1 billion active iPhones*. [Online; accessed 13. Jul. 20201].

Khatri, Yogesh (June 2021). *Gboard has some interesting data.*. [Online; accessed 1. Jul. 2021]. URL: https://www.swiftforensics.com/2021/01/gboard-has-some-interesting-data.html.

Megamanfan3 (Nov. 2020). *Ficheiro:Pixel 4a Android 11 Launcher.png – Wikipédia, a enciclopédia livre*. [Online; accessed 12. Jul. 2021]. URL: https://pt.wikipedia.org/wiki/Ficheiro:Pixel_4a_Android_11_Launcher.png.

Mozilla (July 2021). *Content-Encoding - HTTP | MDN*. [Online; accessed 18. Jul. 2021]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding.

O'Dea, S. (May 2021). *Android - Statistics & Facts*. [Online; accessed 26. Jun. 2021]. URL: https://www.statista.com/topics/876/android.

Patel, Rajan (May 2016). *Meet Gboard: Search, GIFs, emojis & more. Right from your keyboard.* [Online, accessed 2021-06-27] https://blog.google/products/search/gboard-search-gifs-emojis-keyboard/.

Project, Android Open Source (Dec. 2020). *Copy and Paste.* [Online; accessed 1. Jul. 2021]. URL: https://developer.android.com/guide/topics/text/copy-paste.

Ramaswamy, Swaroop et al. (2019). *Federated Learning for Emoji Prediction in a Mobile Keyboard.* arXiv: 1906.04329 [cs.CL].

Reith, Mark, Clint Carr, and G. Gunsch (2002). "An Examination of Digital Forensic Models". In: *Int. J. Digit. EVid.* 1. URL: https://www.semanticscholar.org/paper/An-Examination-of-Digital-Forensic-Models-Reith-Carr/c73f47d8385f452dfd25bbaab754874b65594ccd.

rejectedsoftware (July 2021). *diet-ng.* [Online; accessed 8. Jul. 2021]. URL: https://github.com/rejectedsoftware/diet-ng.

Salus, Peter (June 1994). *Quarter Century of UNIX, A.* xn–3ug : xn–2ug Addison-Wesley. URL: https://www.amazon.co.uk/Quarter-Century-UNIX-Addison-Wesley-Systems/dp/0201547775.

Shafranovich, Yakov (Apr. 2013). *The application/sql Media Type.* RFC 6922. DOI: 10.17487/RFC6922. URL: https://rfc-editor.org/rfc/rfc6922.txt.

Sissel, Jordan (July 2021a). *fpm.* [Online; accessed 10. Jul. 2021]. URL: https://github.com/jordansissel/fpm.

— (July 2021b). *fpm.* [Online; accessed 11. Jul. 2021]. URL: https://github.com/jordansissel/fpm/tree/master/examples/jruby.

— (July 2021c). *fpm - packaging made simple.* [Online; accessed 10. Jul. 2021]. URL: https://fpm.readthedocs.io/en/latest/index.html.

SQLite (n.d.). *About SQLite.* [Online, accessed 2021-06-25] https://sqlite.org/about.html.

TECH, HT (Sept. 2020). *Android 11 features that are exclusive to Google Pixel phones.* [Online; accessed 12. Jul. 2021]. URL: https://tech.hindustantimes.com/tech/news/android-11-features-that-are-exclusive-to-google-pixel-phones-71599991862593.html.

Turner, Ash (June 2021). *How Many People Have Smartphones Worldwide (Jun 2021).* [Online; accessed 2. Jul. 2021]. URL: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

Yang, Timothy et al. (2018). "Applied Federated Learning: Improving Google Keyboard Query Suggestions". In: arXiv: 1812.02903 [cs.LG].

Yata, Susumu (June 2020). *MARISA: Matching Algorithm with Recursively Implemented StorAge*. [Online; accessed 7. Jul. 2021]. URL: https://www.s-yata.jp/marisa-trie/docs/readme.en.html.