Systematic Evaluation of Forensic Data Acquisition using Smartphone Local Backup

Julian Geus[a,*], Jenny Ottmann[a,*] and Felix Freiling[a,*]

[a]Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany

ARTICLE INFO

Keywords:
Mobile Forensics
Storage Acquisition
iOS Backup
Android Backup

ABSTRACT

Due to the increasing security standards of modern smartphones, forensic data acquisition from such devices is a growing challenge. One rather generic way to access data on smartphones in practice is to use the local backup mechanism offered by the mobile operating systems. We study the suitability of such mechanisms for forensic data acquisition by performing a thorough evaluation of iOS's and Android's local backup mechanisms on two mobile devices. Based on a systematic and generic evaluation procedure comparing the contents of local backup to the original storage, we show that in our exemplary practical evaluations, in most cases (but not all) local backup actually yields a correct copy of the original data from storage. Our study also highlights corner cases, such as database files with pending changes, that need to be considered when assessing the integrity and authenticity of evidence acquired through local backup.

1. Introduction

Despite many advances over the years (Freiling, Groß, Latzo, Müller and Palutke, 2018), forensic data acquisition is a tough challenge, especially for smartphones, where security features are making attempts of unauthorized data collection from the devices themselves increasingly futile. Many mobile devices, however, are heavily integrated into networked ecosystems so that the devices themselves are usually not the only storage location of interesting data. Especially smartphones today are in many cases merely the cache of data that is in fact stored on remote servers. Furthermore, cloud storage is commonly used as the location of backup data at the request of users. This has sparked an interesting stream of research in *cloud forensics* (Roussev, Ahmed, Barreto, McCulley and Shanmughan, 2016; Manral, Somani, Choo, Conti and Gaur, 2020).

Tool manufacturers have, however, cleverly observed that backup data can be accessed even without access to cloud storage. The trick is to instruct smartphones to perform a *local* backup to an attached laptop and acquire data from there. Judging from multiple blog posts, this method appears to be commonly used by established "black box" tools for forensic data acquisition (Oxygen Forensics, 2022a; Lorentz and Mahalik, 2022; Magnet Forensics, 2019; Mahalik, 2021). But despite the fact that local backup appears to be one of the few device and manufacturer independent methods for data acquisition that does not need elevated privileges, we are not aware of any related work that assesses the reliability of such acquisition methods and the quality of data which they acquire.

The fact that nobody currently appears to question the suitability of local backup as a sound evidence acquisition method is worrisome since a negative answer would not only affect future cases but also old ones. Consider, for example, a case from the past in which decisive evidence had been acquired from a mobile phone using the software from a well-known forensic service provider. If it turned out that that software was using local backup and that local backup acquisition is commonly incomplete, unreliable or, even worse, often leads to data alteration, the validity of the evidence may be challenged, which could entail a reconsideration of the case.

One reason why local backup methods have not yet been evaluated could be that it is not exactly clear how to do it for two main reasons. Firstly, with Android and iOS there are two large families of backup methods with (at least for Android) multiple variants that are hard to compare. This matter is further complicated by the fact that the backup process is usually app oriented, which leads to even more variance. Secondly, backups regularly run concurrently to the OS and applications which makes it hard to define a ground truth to which any backup data can be compared.

1.1. Contributions

The main contribution of this paper is a generic yet practical evaluation procedure to assess the reliability and data quality of data acquired through local backup. We demonstrate its usefulness by executing it on the two dominating smartphone platforms, Google's Android and Apple's iOS. More precisely, we evaluate what data can and cannot be retrieved from using the local backup service of the platform and to what degree the data retrieved from the backup corresponds to the data that was originally stored on the device. Our evaluation procedure can handle different types of backup, namely content-based as well as file-based backup, and allows for easy repeatability of the evaluation for future versions of operating systems and apps.

More concretely, we executed our procedure on an instance of Android's full local backup and specific package data backup with app-downgrading for various applications,

*Corresponding authors.
Email addresses: julian.geus@fau.de (J. Geus); jenny.ottmann@fau.de (J. Ottmann); felix.freiling@fau.de (F. Freiling)
ORCID(s): 0009-0001-8270-1964 (J. Geus); 0000-0003-1090-0566 (J. Ottmann); 0000-0002-8279-8401 (F. Freiling)

as well as iOS's encrypted and unencrypted local backups. We limit the evaluation to one Android and one iOS device, because the differences between the OS versions are unlikely to be significant, but the practical execution entails considerable efforts. For every case, we executed a total of 20 evaluation iterations to provide reliable results and to locate outliers. The results for Android showed that in the large majority of cases, the acquired data was the same as the source data on the device. The cause of the cases where files showed alterations could most likely be accounted to concurrent background processes since those occurred at random. The results for iOS were similar, but most database files showed alterations compared to their source counterpart. Further investigation showed that this was due to the merging of uncommitted database changes during the backup process. These altered files accounted for over 10% of the data in both encrypted and unencrypted backup runs, which both had around 700 files included in the backup.

For our imaginary legal case, we therefore can attest the forensic suitability of the evidence with the aforementioned limitations.

In summary the contributions of this paper are the following:

- An in-depth categorization of different backup types and an analysis of the amount of included data.
- The development of a generic evaluation procedure which takes the type of data into account.
- Practical backup acquisition evaluations using an Apple iPhone and an Android device.
- The creation of comprehensive backup datasets, which we make accessible to the community (Grajeda, Breitinger and Baggili, 2017) upon request.[1]

1.2. Related Work

Analyses of backup data acquisition are often described in various blog posts from forensic service providers e.g. SalvationDATA (2020); Farley Forensics (2019). For the app-downgrading process on Android devices, Oxygen Forensics (2021, 2022b) posted information about the procedure and its caveats. As mentioned above, to the best of our knowledge, no comparable systematic and scientific evaluation attempts for data acquisition using local backups of smartphones have been carried out so far.

Related work in a broader sense was done by Chang, Teng, Tso and Wang (2015), who created an overview of extractable data on jailbroken and non-jailbroken iPhones and evaluated the impact of jailbreaking on the acquired backup data. They concluded, that the jailbreak does not seem to have an impact on the data. Similarly, Hassan and Pantaleon (2017) analyzed the impact of rooting on the acquired data of an Android device, with the conclusion that no changes could be observed. These studies were carried out on outdated OS versions using outdated jailbreaking and rooting procedures, so it is hard to generalize the results. They also did not publish their datasets. In contrast, we not only consider current OS versions but, more importantly, also provide a generic evaluation procedure that can be repeated under different environmental conditions.

Generally related to the present paper is a study by Son, Lee, Kim, James, Lee and Lee (2013) who describe a tool for forensically sound Android data acquisition using a custom recovery. While being interesting, this method is mostly irrelevant on modern smartphones today due to the increased security standards.

1.3. Paper Outline

In Section 2 we provide an overview of the different backup mechanisms for Android and iOS. This is followed by detailed information about our developed evaluation procedure, the data classification strategy, and the device preparation in Section 3. The results of the evaluation, as well as the device specific procedures, are presented in Section 4 for Android and Section 5 for iOS, each accompanied by a comprehensive analysis of the data. In Section 6, we lay out the limitations of our work and draw a short conclusion in Section 7.

2. A Visit to the Zoo of Backup Methods

The two major mobile platforms, Android and iOS, provide extensive backup functionalities to secure critical data. This data consists primarily of user- and application data, as well as device settings. The backup options can be classified into different categories according to their storage location, their scope and the type of backed up data, as described in the following sections. We will focus exclusively on the backup mechanisms provided by the OS while ignoring third-party and vendor-specific solutions.

2.1. Local vs. Remote Backup

Both platforms offer a possibility to store the backup data locally, on a connected PC, or on the accompanying cloud services, i.e. Google Drive for Android and iCloud for iOS.

The local backup mechanisms have a high value in forensics, due to their ease of use and scope of data. For Android, it can be executed with the *backup* command from the *Android Debug Bridge (ADB)* command-line tool. For iOS the local backup can be created natively on an Apple Mac, using the iTunes software for Windows, or OS independent with the open-source toolset *libimobiledevice* (libimobiledevice, 2022), which is a community project that reimplements Apple's proprietary iDevice communication protocol.

The remote backup is often created automatically, depending on the user's settings, to either of the cloud services. Free storage is already included for iCloud and Google Drive with a user's Google or Apple account. Access to this data for forensic purposes is, however, complicated, or in some instances impossible.

[1] Due to the inclusion of sensitive data like geolocation information, we are not able to make our datasets available for unauthenticated public download. Interested researchers should contact the authors to get access to the datasets.

The trend of backups is moving strongly towards remote backups, with the ADB backup command already marked deprecated since 2019. Therefore, it might be omitted completely in future updates. Furthermore, apps targeting Android 12 and up are automatically disabled for ADB backup. On iOS, however, the local backup function is still actively supported. iOS's local backup can additionally be decrypted using a user-defined password, which leads to the inclusion of more sensitive user data in the backup.

2.2. Full vs. Selective Backup

Every installed app can decide which of its files will be included or excluded from the backup operation. This can be done, for both Android and iOS by saving a file to a specific filesystem location that is excluded or included by default or by manually excluding or including files from the backup. The platform's backup mechanism works mostly in an app or package oriented way, which defines the granularity of the backup. We refer to the backup of only one or more specified packages as selective backup. Whereby, the backup, which iterates all packages, is called a full backup.

However, there are exceptions, since not only app data can be included in a local backup. On Android, the media folder, containing files like images or documents, can optionally be included. iOS follows the concept of domains in its backup mechanism, where one domain in the backup data corresponds to a filesystem location and a certain group of files. The package data is stored in an *AppDomain* folder, one for each application in the backup data. Further domain examples are the *CameraRollDomain*, which includes the images from the phone's camera, or the *HomeDomain*, with various files from the user's home directory.

For Android, a selective backup can be created by specifying the packages with ADB's backup command. By using the `-full` switch, all packages are iterated and therefore included. On iOS, no such possibility exists, only a full backup including all domains and packages can be created. The only influence on the amount of data included in an iOS backup is by enabling or disabling the backup encryption, since encrypted backups contain more sensitive data, like health data or stored passwords.

2.3. File-based vs. Content-based Backup

The problem with app-oriented backup design is that only the app itself can really know the meaning of the data it contains, and therefore, which data needs to be included in a backup. The freedom of applications to determine the content of their backups results in two different types of data that needs to be treated differently when performing acquisition: *file-based* and *content-based* data (see Figure 1).

We refer to one-by-one copies of files from the device's filesystem as file-based data. In contrast to file-based data, content-based data is a copy of only partial data contents out of a file. The resulting file may be of the same format with reduced or rearranged content or of a completely different file format. Therefore, the resulting data does not represent an entire file but rather a subset of data stored in the file (e.g. only one table from a bigger database). There are content-based parts in both platforms' backup mechanisms, with Android having a broader spectrum. In Android, especially system packages use the so-called *key-value backups* to store valuable information, like device settings and the call history. Key-value backup is a special backup mechanism, that stores the data as key-value pairs. By default an app's backup is file-based, but a developer can choose to use key-value backups instead, by defining its own *BackupAgent* class or extending from it. By doing so, the entire backup and restore procedure for the data has to be implemented. However, not many third-party apps are using this legacy mechanism anymore. A special case of content-based data is Android's SMS backup, which is not exactly a key-value backup but still only includes parts from the original database file.

iOS has no equivalent to Android's key-value backups. Almost the entire backup mechanism is file-based, with one exception which is the backup of the device's *keychain*. The keychain is a database containing separately encrypted, sensitive bits of data, like the user's login passwords. Parts of that keychain are included in an encrypted backup as content-based data.

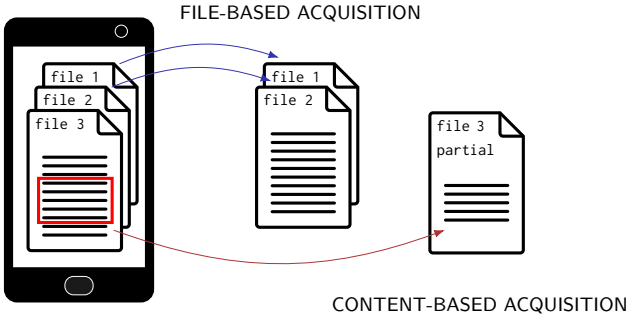

Figure 1: Data types that can be acquired by the different acquisition methods.

2.4. App-downgrading

The fact that the apps themselves decide on how to participate in a backup implies that different versions of apps might also behave differently regarding backup requests by the OS. If an app behaves more favorable regarding backup in an older version than in the current version, one strategy is to re-install the older version before performing the backup operation. This procedure is known as *app-downgrading*. Selecting a particular version of an app therefore is a way to influence the behavior of the local backup.

App-downgrading is of special importance for Android, since many applications in their latest version no longer support local backups of their data. However, in older app versions Android's local backup mechanism was often available. Therefore, by downgrading the app, it is possible to access local files of many forensically relevant applications that would otherwise be inaccessible. In Android the process involves multiple steps. First, a backup of the application's current APK file is created, followed by uninstalling the current version without deleting its local data, installing an

older version with backup support and executing the backup process for this package. After a successful backup, the current app version can be reinstalled to restore the initial state.

A patch was applied to the Android Open Source Project (AOSP) in 2016, that should prevent data access by downgrading (Klyubin, 2016). However, this technique is still working and is evidently also being used by forensic service providers (Oxygen Forensics, 2021; Cellebrite, 2019; Belkasoft, 2021).

3. Evaluation Methodology and Experimental Setup

There are different ways to conduct an evaluation, which can generally be distinguished between more practical approaches, using real devices, and laboratory approaches, with an emulator or development boards. In this paper, the evaluation is performed with actual hardware to simulate conditions as close to the forensic practice as possible. But this approach also entails some disadvantages, especially concerning the accessibility of data.

The focus of this work lies on (1) local backup and (2) the backup service offered by the two platforms Android and iOS. We therefore neither consider data acquisition from cloud storage nor from backups performed by specialized third-party software. Within this scope, we evaluate data acquisition both from file-based as well as content-based backup.

3.1. Evaluation Model

Based on the previous considerations, we present a generic practical evaluation approach, which serves as a template for the specific evaluation processes. The method of *differential forensic analysis*, similar to Garfinkel, Nelson and Young (2012), is used as a baseline for the experiments.

To generalize from file- and content-based data, we consider all backed up data as consisting of name-value pairs (n, v). For file-based backup, the name n corresponds to the unique filepath including the filename and the value v to the file's content, i.e., the complete bitstring stored under the filename in the filesystem. For content-based backup, n has to identify the unique file but also the data object inside that file (e.g., a tag or column name from a database) while v is the value of that data object.

The evaluation is performed by systematically comparing the values acquired from local backup to those values that were previously existing under the same name on the device, a process that we call an *iteration* and which is depicted in Figure 2. To reduce the effect of noise we perform multiple iterations.

In practice, the local backups contain both, file-based and content-based data. For the file-based parts, a simple hash-sum comparison is used to test for equality. For the content-based data, which mostly stems from structured files, like databases, the data itself is divided by the smallest indivisible unit, according to the original filetype. The comparison is made by simply comparing these values, according to their original interpretation, which is mostly string-based.

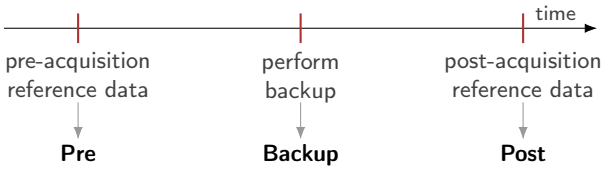

Figure 2: Generic evaluation model

In each iteration, three steps are performed:

1. We collect a set of *pre-acquisition reference data* denoted as *Pre*. *Pre* corresponds to the *maximal set* of name-value pairs that can be obtained by the local backup method to be evaluated.

2. We then perform data acquisition using that backup method. The resulting set of name-value pairs is denoted by *Backup*.

3. After the backup is complete, we collect a set of *post-acquisition reference data* denoted as *Post*. The scope of this data set should be identical to that of *Pre*.

The necessity to create a maximal set results from the fact that the selection of backup data is performed by each specific app, i.e., we do not know beforehand which data objects will be part of *Backup*. In this sense, *Pre* and *Post* are an overapproximation of *Backup* regarding the set of name-value pairs. This implies that not all files contained in *Pre* and *Post* will also be present in *Backup*. Still, *Backup* may contain data that is not present in either *Pre* or *Post*. We explain the different conditions that may occur below.

The creation of *Pre* and *Post* critically depends on the platform and the concrete device. Conceptually, it is similar to the creation of *Backup*, but since it serves as a reference, any reliable data acquisition method that creates a *snapshot* of the filesystem can be used here.

Creating snapshots on mobile devices requires privileged access (as root user). The only possibility for full data access on iOS devices is to enable privileged access by exploiting vulnerabilities. This process is called *jailbreaking*; it introduces many changes to iOS and is accompanied by the installation of an unofficial app store most of the time. But due to the lack of options, we chose this method for the iPhone, to enable reference data access.

Certain Android phones offer the option to unlock their bootloader, which disables Android's *Verified Boot*, and thus, enables booting custom systems. With an unlocked bootloader, there are two options to gain access to the filesystem data for reference data acquisition. A custom recovery system, like *TWRP*, can be booted instead of Android and can be used to access the phone's data or the Android OS can be modified to enable access to the root account. To be in line with iOS and to increase the methods comparability, the reference data acquisition for Android will take place

from within the OS as well, by enabling root access on an unlocked phone. This method was also chosen, because tests showed a huge amount of data alterations during the reboot necessary for the recovery mode option, which can be mitigated with the acquisition from within Android.

3.2. Data Evaluation

After performing the above steps, we evaluate the data. Generally, the basic item of interest in our evaluation is a single name-value pair. Depending on whether or not a pair with a specific name exists in *Pre*, *Post*, or *Backup* and what value is associated with it, we distinguish the following data sets. For ease of notation, by $names(X)$ we refer to the set of all names occurring in the set X of name-value pairs.

- E: The set of all observed names in *Pre* together with those in *Backup*. Formally:

  $$names(Pre) \cup names(Backup)$$

  This set characterizes the size of the data acquisition experiment.

- N_{over}: The set of all names that are in *Pre* but not in *Backup*. Formally:

  $$names(Pre) \setminus names(Backup)$$

  This set characterizes the amount of existing data that was not backed up. Since *Pre* is intentionally an overapproximation of *Backup*, the size of this set merely quantifies the degree of overapproximation.

- N_{new}: The set of names that are in *Backup* but not in *Pre*. Formally:

  $$names(Backup) \setminus names(Pre)$$

  This set characterizes the spurious or "new" data in the backup that did not appear to exist before.

- N_{both}: The set of all names that are both in *Backup* and in *Pre*. Formally:

  $$names(Pre) \cap names(Backup)$$

  This set characterizes the amount of existing data that was backed up.

  Note that by construction $|E| = |N_{over}| + |N_{new}| + |N_{both}|$.

- V_{eq}: The set of all names of all name-value pairs where the names are both in *Backup* and in *Pre* and where the corresponding values are equal. Formally:

  $$names(Backup \cap Pre)$$

  Note that the intersection is on name-value pairs and not only on names. Therefore this set characterizes the elements that were backed up and whose content in the backup did not change.

- V_{ch}: The set of all names of name-value pairs whose names are both in *Pre* and *Backup* but where the values are different. Formally:

  $$\{n : (n, v) \in Pre \wedge (n, v') \in Backup \wedge v \neq v'\}$$

  This set is comprised of backed up elements whose content appears to have changed.

  Note that by construction $|N_{both}| = |V_{eq}| + |V_{ch}|$

The following sets further subdivide the set V_{ch}, i.e., those data objects that have changed before or during backup. Based on a comparison with *Post* this subdivision attempts to characterize possible causes of the change of content:

- P_{mis}: The set of all names in V_{ch} which do not occur in *Post*. Formally:

  $$V_{ch} \setminus names(Post)$$

- P_{mback}: The set of all names in V_{ch} that appear in *Post* and where the value in *Backup* is equal to that in *Post*. Formally:

  $$\{n : n \in V_{ch} \wedge (n, v) \in Backup \wedge (n, v) \in Post\}$$

  This set characterizes those elements that changed before the backup but not thereafter. If the creation of *Pre* happens shortly before the backup, elements in this set may have been changed *by* the backup mechanism.

- P_{mpre}: The set of all names in V_{ch} where the value in *Backup* neither matches the value in *Pre* and *Post* but where *Pre* and *Post* agree on the value. Formally:

  $$\begin{aligned}\{n : n \in V_{ch} &\wedge (n, v) \in Backup \wedge (n, v) \notin Post \\ &\wedge (n, v') \in Pre \wedge (n, v'') \in Post \\ &\wedge v' = v''\}\end{aligned}$$

  This set characterizes backed up elements, where the original data on the storage device does not appear to have changed but where the backed up value is different.

- P_{nom}: The set of all names in V_{ch} where the value in *Backup* neither matches the value in *Pre* and *Post* and where also *Pre* and *Post* disagree. Formally:

  $$\begin{aligned}\{n : n \in V_{ch} &\wedge (n, v) \in Backup \wedge (n, v) \notin Post \\ &\wedge (n, v') \in Pre \wedge (n, v'') \in Post \\ &\wedge v' \neq v''\}\end{aligned}$$

  This set characterizes elements where the cause of the mismatch is hard to determine because the data in *Post* does not provide any helpful information.

  Note that by construction $|V_{ch}| = |P_{mis}| + |P_{mback}| + |P_{mpre}| + |P_{nom}|$.

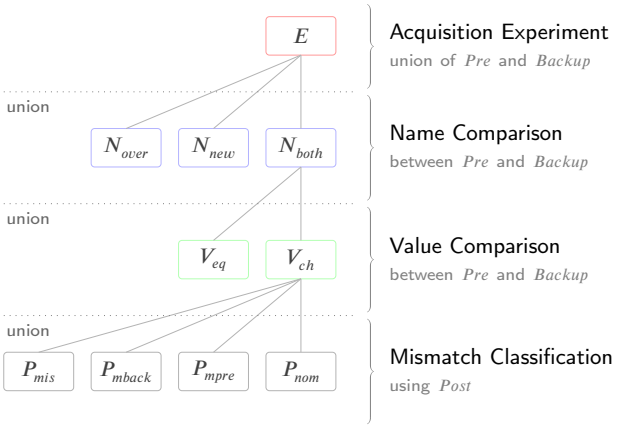

Figure 3: Relations between the data sets

PIXEL 2 DEVICE INFORMATION	
Device	Google Pixel 2
Codename	walleye
Storage	64 GB
Android	11
Build Number	RP1A.201005.004.A1
Encryption	File-Based (FBE)
SOFTWARE INFORMATION	
adb	28.0.2-debian
Android Backup Extractor	v20210530
Magisk	25.2

Table 1
Hardware and software details of the Pixel 2 device and software information used for the acquisition.

The relations between the defined sets of data are depicted in Figure 3.

For the items in V_{ch}, i.e., those that changed between *Pre* and *Backup*, we provide an indication of the degree of changes that occurred (or the degree of similarity). Therefore, the similarity ratio r is calculated with the *quick_ratio* function from the SequenceMatcher in Python's difflib library (Python Software Foundation, 2022), which can determine the similarity between two byte sequences. This is done by comparing the values of the two matching names for their likeness and as result, a value between 1.0 (identical) and 0.0 (completely different) is returned. The number of items considered for this value is the cardinality of the set V_{ch} and thus the sum of all value-mismatches, with a name that exists in both *Pre* and *Backup*. To avoid the distortion caused by different sized sets, the average is weighed by the sizes s. Thus, the $\bar{r}_w$ value is the weighted arithmetic mean of all r, calculated by the formula

$$\bar{r}_w = \frac{1}{\sum_{i=1}^{|V_{ch}|} s_i} \left(\sum_{i=1}^{|V_{ch}|} s_i \cdot r_i \right)$$

and similarly the weighted standard deviation σ_w is calculated as

$$\sigma_w = \sqrt{\frac{1}{\sum_{i=1}^{|V_{ch}|} s_i} \left(\sum_{i=1}^{|V_{ch}|} s_i (r_i - \bar{r}_w)^2 \right)}.$$

These values provide a general indication of the number of changes within the files that are affected by a value-mismatch.

3.3. Android Device Setup

The device used for the Android evaluation is a Google Pixel 2. It has the benefits of enabling a bootloader unlock and all released OS images can be officially downloaded from Google's website. Detailed information about the device and the software tools used for the acquisition are listed in Table 1.

To perform the evaluation, a possibility to obtain reference data is crucial. As described above, the reference data is acquired within Android using root privileges. Furthermore, a credible usage behavior has to be simulated. Accordingly, the device needs to be prepared for our needs.

The first step was to unlock the bootloader of the phone to enable the installation of custom partition images, which allows for the neccessary system adjustments required for rooting. The actual rooting process was accomplished with *Magisk*. Magisk is delivered as Android APK file which enables patching the default *boot.img* which is part of the factory image. The patch enables apps and shell sessions over ADB to acquire root privileges. Therefore, the patched *boot.img* file has to be flashed onto the device's *boot* partition using *fastboot*.

After the device is set up to enable full data access, basic traces to simulate a normal usage behavior have to be created. This includes some incoming and outgoing phone calls, SMS messages, and emails. Furthermore, contacts have been added and pictures were taken with the device's camera. Moreover, the widely used messenger applications *WhatsApp*, *Telegram*, and *Facebook Messenger* were installed, alongside the *Firefox* browser, *Instagram*, and *Twitter*.

The state of the device before the acquisition is of importance as well. To simulate an acquisition under realistic circumstances, the device is assumed to be in a currently seized state with default settings and the device is protected with a passcode. Furthermore, in compliance with the forensic guidelines for mobile devices (Ayers, Brothers, Jansen et al., 2014, p.28), the SIM card was removed before the acquisition and it was ensured that the device cannot connect to any network.

3.4. iOS Device Setup

The device chosen for the iOS backup evaluation is an iPhone 8, which was released in 2017, but still supports the latest OS version (iOS 16). This device has the benefit of being susceptible to the *checkm8* BootROM vulnerability, and hence, privileged access can be gained. Detailed device

IPHONE 8 DEVICE INFORMATION	
Device	iPhone 8
Model Number	MQ6G2ZD/A
Storage	64 GB
iOS Version	14.6
SOFTWARE INFORMATION	
libimobiledevice	1.3.0-160
iproxy	2.0.2-24
checkra1n	0.12.4 beta
iOSbackup (python library)	0.9.923
irestore	1a78c2f

Table 2
Hardware and software details of the iPhone 8 and software information used for the acquisition.

information and the software used for the acquisition is provided in Table 2.

The iPhone must also be prepared for the evaluation with default usage behavior and privileged access for reference data gathering. Therefore, the preparation steps include applying a jailbreak to the device and creating usage traces.

The jailbreaking of the iPhone is accomplished with the *checkra1n* tool (checkra1n) that uses the checkm8 BootROM vulnerability. After a successful jailbreak, a new app symbol with the checkra1n logo appears on the home screen. This app can be used to remove the jailbreak from the device or to install *Cydia*, which enables the installation of various packages and system modifications. To enable access over SSH, Cydia is utilized to install the *OpenSSH* package. After the installation, an SSH connection over the default port 22 is possible.

Similar to the Android device, some default usage behavior was simulated, to ensure a realistic dataset. For that purpose, the apps *WhatsApp*, *Telegram*, *Signal*, and *Youtube* were installed, activated, and basic usage was simulated. For the Apple account, the default settings for iCloud and privacy options were used. Moreover, an email account was set up, SMS messages were received and sent, contacts were created, calls simulated, and photos taken.

Again, the state of the device, in which the evaluation takes place needs to be considered. Since the reference data is gathered from within the OS using the elevated privileges from the jailbreak, the device's pre-acquisition state is powered on, locked, but without a lock screen passcode, due to limitations of the jailbreak. And in compliance with the aforementioned guidelines, all communication capabilities were disabled.

The method of reference data gathering must be implicitly trusted. Apart from the jailbreak tool, of which the source code is only partially published, all tools are open-source and widely used. However, the jailbreak tool is not explicitly used for data acquisition but only provides the necessary privileges and its developers are well-known. This leads to reasonable confidence about the soundness of the reference data acquisition process. However, since the jailbreak tool has full access to the OS, a compromised system cannot entirely be ruled out, which should at least be noted.

4. Android Backup Results

We now present the evaluation process and its results for Android using ADB's local backup function. This is Android's most generic acquisition method that can be used on any device, no matter the OEM or software version. Since the amount of data that can be acquired with a local backup is limited, the method was complemented by app-downgrading, with numerous apps that have special relevancy for forensics.

4.1. Evaluation Process

The data included in the backup process is a subset of the files from the `/data/data/`, `/data/user_de/0/`, and `/data/media/0/Android/data/` package directories, which depends on the installed apps and their respective backup settings. In addition, the user's media files, which are located in the subfolders of the `/data/media/` directory can be backed up, which is also taken into account in the evaluation.

The reference data sets *Pre* and *Post* for the evaluation process are acquired using the root account and the *tar* and *nc* command-line tools. The reference data sets for the full backup include the entire contents of the folders described above. For the app-downgrading process, only the package-specific subdirectory inside those folders was considered.

The actual backup acquisition process was executed by using ADB's backup command. For a full backup, the corresponding ADB command is used with the respective parameters to include all app data, media files, and key-value data. It results in an *Android Backup (.ab)* file, which is converted to a tar archive using the *Android Backup Extractor* utility.

For the evaluation of the app-downgrading method the apps displayed in Table 3 are chosen. These apps were activated and used prior to the evaluation to ensure conditions and data sets are reasonably realistic. The versions to downgrade to were determined using a blog post by Oxygen forensics as a reference (Oxygen Forensics, 2021). The respective APK files can easily be found with a quick online search. This process, again, results in an Android Backup file, which can be converted to a tar archive that contains the data of the respective package.

The evaluation of file-based backup contents is based on comparing filenames and hash-values of the file content. However, since the file structure of the backup does not match the structure of the filesystem, the backup's file paths need to be mapped to their original locations first. This is possible with the names of the package's subfolders, which identify the original file location.

For the evaluation of content-based backup data, we focus on key-value backups which can be utilized by applications instead of the more modern file-based alternative. Furthermore, the backup of SMS data is content-based because it is stored as a compressed archive containing a JSON

App	Package Name	Installed Version	Downgrade Version
WhatsApp	com.whatsapp	*2.23.1.76*	*2.11.399*
Telegram	org.telegram.messenger	*8.9.3*	*1.3.1*
Firefox	org.mozilla.firefox	*109.1.1*	*39*
Facebook Messenger	com.facebook.orca	*393.0.0.18.92*	*14.0.0.16.14*
Twitter	com.twitter.android	*9.73.0-release.0*	*4.0.3*
Instagram	com.instagram.android	*267.0.0.18.93*	*7.9.0*

Table 3
Information about the apps chosen for the downgrading acquisition.

file, with partial data from the database. The evaluation of key-value backups pose a challenge, because an app has full control over the backed-up data and the format of the key-value items can vary. Therefore, analyzing the data from third-party applications might not be useful, but is also not of particular relevance, since none of the considered third-party applications support this legacy backup mechanism. However, Android system applications still make use of this feature to backup valuable data. Khatri (2019) analyzed key-value backups in a series of blog posts, with a special focus on the call logs and system settings, which are used as a reference for the evaluation.

The entire process was repeated for a total of 20 times for the full backup and app-downgrading procedures to eliminate outliers.

4.2. File-Based Backup

The results of the file-based parts of the backup process are presented in Table 4. These include all files that can be acquired with a local backup and the app-downgrading process. The vertical categories are therefore divided by the **Full Backup** and **App-Downgrading** data, which in turn is segmented by the individual applications (WhatsApp, Telegram, Firefox, Facebook Messenger, Twitter, and Instagram).

Due to the large number of evaluation runs, the results shown in the table were grouped based on equal values in the **Value Mismatch Classification** (P_x classes). The number of runs in a group can be taken from the first column (#). Thus, all remaining values in the table are averaged, as can be seen in the heading. The rows are arranged in descending order according to the group's magnitude, i.e. according to its significance.

The **File Classification** (N_x) sets are an indicator of the validity of each individual run. Especially the N_{new} class, which indicates omissions of *Backup* data in the reference set, is crucial for the value comparison. Thus, an empty set, which we can observe in most cases, means that all files from *Backup* are included in the value comparison. In the case of an empty N_{new} set, the magnitude of set N_{both} must be equal to the size of *Backup*. On some iterations, we encountered a non-empty N_{new} set, which can occur due to issues during the copying process of the reference data, or by newly created temporary data in the timeslot between acquiring *Pre* and *Backup*.

The **Value Classification** (V_x), provides an insight into the quality of the acquired data from the backup process. Therefore, data alterations between *Pre* and *Backup* are reflected in the number of elements in set V_{ch}. As can be seen in the table, the app-downgrading processes where affected in numerous cases, especially in the case of Instagram. However, since it is not a regular phenomenon, but only affects a limited number of runs, it can be concluded that these changes are not an indication for alterations due to the backup process or the app-downgrading procedure. They merely capture the degree of concurrency of a given program.

According to the observed alterations, the interesting segments of the table concerning the **Value Mismatch Classification** are the app-downgrading sections. Here we can observe several classes of mismatches according to the *Post* data comparison. However, their occurrence is only sporadic and scattered among the classes. No repeating patterns can be observed. Thereby, the suspicion of concurrent data changes, during or between the individual acquisition processes, is reinforced.

4.3. Content-Based Backup

Now, the results and the scope of content-based data in the backup process are presented. Therefore, the following content-based files from the backup were considered for the evaluation due to their importance:

- File: **`000000_sms_backup`**

The `com.android.providers.telephony/d_f/` directory from the backup includes this file, it contains the user's SMS data and is available even without including key-value backup data. There can be more than one SMS backup file, numbered sequentially in ascending order, but due to the overseeable amount of messages, only one was created in the process. The *Backup* data consists of the logical values of this file, mapped to their original location. Those are the values *address*, *body*, *date*, *date_sent*, *read*, *status*, and *type* from the *sms* table and the *recipients* from the *threads* and *canonical_addresses* table of the *mmssms.db* file. Therefore, the *Pre* and *Post* data sets were comprised of the contents from the *mmssms.db* database file, which again is an overapproximation of the backup data.

- File: **`com.android.calllogbackup.data`**

This key-value backup from the `com.android.calllogbackup/k/` directory contains information about the call history.

	Filecount			File/Value Classification						Value Mismatch Classification					
					$\lvert E\rvert$			$\lvert N_{both}\rvert$		$\lvert V_{ch}\rvert$				Similarity Ratio	
#	$\lvert Pre\rvert$	$\lvert Backup\rvert$	$\lvert Post\rvert$	$\lvert E\rvert$	$\lvert N_{over}\rvert$	$\lvert N_{new}\rvert$	$\lvert N_{both}\rvert$	$\lvert V_{eq}\rvert$	$\lvert V_{ch}\rvert$	$\lvert P_{mis}\rvert$	$\lvert P_{mback}\rvert$	$\lvert P_{mpre}\rvert$	$\lvert P_{nom}\rvert$	$\bar{r}_w$	σ_w
Full Backup															
20	10853	1365	10856	**10856**	9511	0	1365	1365	0	0	0	0	0	0.0000	0.0000
App Downgrading															
WhatsApp															
15	226	217	226	**226**	9	0	217	217	0	0	0	0	0	0.0000	0.0000
5	204	214	223	**223**	9	19	195	194	1	0	1	0	0	0.7951	0.0000
Telegram															
20	374	157	374	**374**	217	0	157	157	0	0	0	0	0	0.0000	0.0000
Firefox															
11	2179	185	2178	**2179**	1994	0	185	185	0	0	0	0	0	0.0000	0.0000
4	2178	185	2179	**2179**	1994	0	184	183	1	0	1	0	0	0.6936	0.0000
3	2179	185	2178	**2179**	1994	0	185	184	1	1	0	0	0	0.4442	0.0000
2	2178	185	2178	**2179**	1994	0	184	183	1	0	0	0	1	0.5565	0.0000
Facebook Messenger															
11	719	643	713	**719**	75	0	643	643	0	0	0	0	0	0.0000	0.0000
5	713	643	719	**720**	77	6	636	634	2	0	2	0	0	0.8648	0.0139
3	730	646	727	**740**	94	10	636	633	3	0	3	0	0	0.9915	0.0352
1	720	645	724	**720**	75	0	645	644	1	0	1	0	0	0.8550	0.0000
Twitter															
11	146	89	147	**146**	57	0	89	89	0	0	0	0	0	0.0000	0.0000
7	145	89	145	**146**	56	0	88	87	1	0	1	0	0	0.9615	0.0000
2	144	88	145	**144**	56	0	88	86	2	0	2	0	0	0.8160	0.0888
Instagram															
16	748	683	746	**762**	78	14	669	665	4	0	3	1	0	0.9942	0.0283
1	741	683	748	**741**	58	0	683	682	1	0	1	0	0	0.9905	0.0000
1	763	712	776	**774**	62	11	701	698	3	0	2	1	0	0.9928	0.0325
1	745	685	750	**763**	78	18	667	661	6	0	5	1	0	0.9961	0.0223
1	744	684	749	**762**	78	18	666	659	7	0	6	1	0	0.9964	0.0242

Table 4
Results of Android's file-based data evaluation of the full backup and the app-downgrading evaluation runs.

The *Backup* data consists of a subset of the *calllog.db* file. This data includes the values *_id*, *number*, *presentation*, *date*, *duration*, *type*, *subscription_component_name*, *subscription_id*, *phone_account_address*, and *block_reason* from the *calls* table. The database file, therefore, determines the content of the *Pre* and *Post* data sets.

- File: `com.android.providers.settings.data`

This key-value backup file from the `com.android.providers.settings/k/` directory contains a subset of settings from various files. Therefore, *Pre* and *Post* were comprised of the values in the files *settings_config.xml*, *settings_global.xml*, and *settings_secure.xml*, which contain various device settings, and of the values of *WifiConfigStore.xml*, and *WifiConfigStoreSoftAp.xml*, that store various WiFi and network settings.

The detailed results of the content-based evaluation are depicted in Table 5. The content-based data was taken from the full backup runs used for the file-based evaluation, whereby only one instance is shown in the table since there were no variations. Furthermore, columns that do not provide additional information have been dropped. The values with matching names were compared using a string-based comparison. More key-value files existed in the backup data, but most of them are dummy files or don't include relevant data, and since the content-based evaluation is more time-consuming, only parts deemed forensically relevant were considered.

As can be seen in the table, the content-based parts of the evaluation process did not show any signs of alterations or deviations over all backup iterations. This result was expected, due to the very limited amount of data.

Atomic Elements			Name/Value Classification				
				$\lvert E\rvert$			$\lvert N_{both}\rvert$
$\lvert Pre\rvert$	$\lvert Backup\rvert$	$\lvert Post\rvert$	$\lvert E\rvert$	$\lvert N_{over}\rvert$	$\lvert N_{new}\rvert$	$\lvert N_{both}\rvert$	$\lvert V_{eq}\rvert$
SMS Backup							
365	56	365	**365**	309	0	56	56
Calllog Backup							
101	20	101	**101**	81	0	20	20
Settings Backup							
494	51	494	**494**	442	0	51	51

Table 5
Results of the content-based evaluation runs.

4.4. Summary

Overall, Android's backup process shows a high level of data integrity, which is only affected by the program's and the OS's concurrency. However, these effects were expected and should always be taken into account. Particularly in the case of the app downgrading procedures, which require significant modifications, it is noteworthy that the private application data of the corresponding packages does not experience any alterations in the process. Of course, these results are not generally applicable and have to be redone for different applications or under different circumstances.

5. iOS Backup Results

In the following, we present the results and the evaluation process of iOS's backup function. Contrary to Android, iOS's backup mechanism is still actively supported and enables access to a large set of forensically important data.

5.1. Evaluation Process

The amount of data acquired by iOS's backup mechanism cannot be specified in advance, since it depends on the installed apps, their data quantities and backup policies, the iCloud settings, and others. Therefore, the set of data acquired as reference data for *Pre* and *Post*, is again an overapproximation, that ensures a backing of all files contained in the backup. Since iOS's backup process also includes files from various folders in its filesystem, the overapproximation will contain the entire contents of the data partition, mounted at `/private/var/`.

The acquisition of *Pre* and *Post* is accomplished by using the previously applied jailbreak, which enables root access over SSH. The SSH access to the iPhone is established over USB using *iproxy* from the libimobiledevice project. The data copy process is carried out by compressing the directory with the *tar* command whose output is piped over SSH to the connected PC.

For the backup creation, libimobiledevice's *idevicebackup2* tool is used since it enables easy utilization under Linux without the need for iTunes or macOS. Furthermore, two datasets, with and without backup encryption are created. The unencrypted backup is comprised of a subset of the data of an encrypted backup, but since no further processing steps are necessary, it can be ensured that any modifications to the data originate from the backup process. For the decryption of the encrypted backups, the python library *iOSbackup* is used. Because of some shortcomings of this library, which led to broken database files, the *irestore* tool, that is publicly available on GitHub (Dunham, 2021), was used in addition.

iOS's backup contains file-based data for the most part. However, some content-based data is present as well, which needs to be considered separately.

Similar to the previous evaluation processes, iOS's local backup mostly creates direct file copies. These files can then simply be compared to their reference counterpart. However, the assignment of the files to their original filesystem location is not trivial since all filenames and the file structure differ. The mapping to the respective filesystem data can be achieved with the contents of the *Manifest.db*, which contains a domain and the relative path including the original filename for each file.

We now turn to the content-based data. On encrypted backups, some of the keychain entries are included, which contain valuable user passwords and login information. The decryption of this data is achieved by using the *irestore* tool, which offers the option to extract the keychain entries as a JSON file. Since the keychain database in the reference data set is encrypted using the device's hardware key, encryption of the database entries is not possible. However, several login passwords are stored on the phone, which can in turn be compared to the acquired backup data.

A total of twenty evaluation runs for the encrypted- and unencrypted backup processes are executed consecutively. The chosen quantity provides a good compromise between a big enough number of test runs for a conclusive outcome while keeping the required storage space and the runtime within reasonable limits.

5.2. File-Based Backup

The evaluation results of the file-based data of iOS's local backup are displayed in Table 6. The table is separated into *Encrypted Backup* and *Unencrypted Backup*, which each include an additional $|V_{ch}|$ **Overlapping Files** row, as a measurement of files that are affected in each P_x category in every run. Similar to Android's result table, the rows are grouped by the P_x categories and sorted in descending order by the group's magnitude.

The general **Filecount** sections show that the amount of data in the reference sets *Pre* and *Post* is slightly changing. This was expected due to the size of the data sets and concurrently running programs. However, the *Backup* data stayed constant over all runs, which indicates a high degree of determinism in this process. The discrepancy between the amount of files included in encrypted and unencrypted backups is based on the fact that encrypted backups generally contain more data.

The table's **File Classification** section especially highlights the validity of all runs, since set N_{new} is empty, and therefore, N_{both} always contains the entire *Backup* set. This leads to the inclusion of all files in the value comparison in every iteration.

The P_{mis} set from the **Value Classification** category, that is empty in each iteration, again reinforces the validity, as it indicates that all the files from N_{both} are also present in the *Post* set, and therefore, can be classified in greater detail. P_{mback} always contains at least one file, which is affected in every iteration, as can be observed in the overlapping row. Since its values are always equal between *Backup* and *Post*, the change occurs between *Pre* acquisition and backup creation. On further examination, the file contains unidentifiable metadata in XML format, including a timestamp, that seems to be updated every time the backup process is executed. The remaining occasionally occurring cases are probably attributable to concurrency, which can also be stated about the single entry in P_{nom}, which has a different value in all three data sets. The most interesting observation is the high value of P_{mpre} elements, that are constant over all iterations, and as the overlapping row suggests, also concerns the same files in each iteration. This set represents files that are identical before and after the backup's execution, but with value changes in the *Backup* data. This suggests an alteration that is happening during the backup process. Furthermore, the **Similarity Ratio** section attests high value similarities, but the $\bar{r}_w$ and σ_w values are almost identical over all runs. This leads to the assumption that the files are subject to the same changes in every iteration.

On closer inspection, we found that the affected files were exclusively SQLite databases. The majority of databases that were subject to changes were still contextually identical to the reference set concerning their logical data, however, some showed alterations to various degrees. This anomaly could be traced to the *Write-Ahead Logging* mode of an

	Filecount			File/Value Classification						Value Mismatch Classification					
					$\lvert E\rvert$			$\lvert N_{both}\rvert$		$\lvert V_{ch}\rvert$				Similarity Ratio	
#	$\lvert Pre\rvert$	$\lvert Backup\rvert$	$\lvert Post\rvert$	$\lvert E\rvert$	$\lvert N_{over}\rvert$	$\lvert N_{new}\rvert$	$\lvert N_{both}\rvert$	$\lvert V_{eq}\rvert$	$\lvert V_{ch}\rvert$	$\lvert P_{mis}\rvert$	$\lvert P_{mback}\rvert$	$\lvert P_{mpre}\rvert$	$\lvert P_{nom}\rvert$	$\bar{r}_w$	σ_w
Encrypted Backup															
14	39400	715	39401	**39400**	38685	0	715	630	85	0	1	84	0	0.9275	0.1438
4	39401	715	39399	**39401**	38686	0	715	629	86	0	2	84	0	0.9275	0.1438
1	39398	715	39398	**39398**	38683	0	715	628	87	0	2	84	1	0.9275	0.1438
1	39402	715	39402	**39402**	38687	0	715	628	87	0	3	84	0	0.9275	0.1438
				$\lvert V_{ch}\rvert$ Overlapping Files						**0**	**1**	**84**	**0**		
Unencrypted Backup															
18	39404	682	39404	**39404**	38722	0	682	611	71	0	1	70	0	0.9310	0.1351
1	39405	682	39405	**39405**	38723	0	682	610	72	0	2	70	0	0.9310	0.1351
1	39406	682	39407	**39406**	38724	0	682	609	73	0	3	70	0	0.9311	0.1351
				$\lvert V_{ch}\rvert$ Overlapping Files						**0**	**1**	**70**	**0**		

Table 6
Results of the file-based data evaluation for the encrypted and unencrypted backups.

SQLite database, that, when in effect, creates a WAL file, which contains not yet committed database changes. These files, however, are not included in the backup process, but when their content is committed manually to the databases in the reference data, the mismatches disappear. This leads to the conclusion, that the backup process commits the data of the WAL files to the backed up databases. According to (Caithness, 2012), a deletion in a database with Write-Ahead Logging that is not yet committed would still be recoverable. This information is therefore missing in the acquired data of an iOS backup.

5.3. Content-Based Backup

The only content-based data of iOS's local backup is the keychain. The keychain on the device consists of the single database file `/private/var/Keychains/keychain-2.db`. Its contents, however, are separately decrypted using the device's hardware key, and therefore, their decryption is not possible. Hence, no *Pre* and *Post* data sets are obtainable. On a backup, the *keychain-backup.plist* file is created, which contains a subset of the database file as encrypted entries that are only protected with the backup password. The *Backup* set is therefore comprised of the decrypted values from this file. But since the contents of the *keychain-2.db* is not decryptable, our evaluation procedure cannot be applied to this data. To at least provide an indication of the data's integrity, various browser login information, as well as WLAN authentication data were stored on the device. This data is secured in the keychain and included in the backup. The comparison of those values to their *Backup* counterpart showed no alterations.

5.4. Summary

In summary, iOS's backup process is a good source for forensically relevant data. It should, however, be noted that the acquired data is partially subject to changes due to the backup process. But since the alterations stem from the inclusion of already executed database changes, the data still reflects the actual database state. Therefore, some forensically relevant data might be missing, but this does not invalidate the acquired data. Similar to Android, some concurrency related changes occured, which were expected and cannot completely be ruled out since the acquisition process uses the OS of the mobile device.

6. Limitations

It is important to note that the differences between OS and application versions, as well as device manufacturers, can be considerable. Consequently, the result cannot be generalized, instead, the evaluation is intended to provide a reference that must be redone under different environmental conditions.

Since backup data compatibility is necessary between different OS versions, we do not expect the results of the general backup mechanism to differ significantly. Changes can, however, not be ruled out without a thorough evaluation. Especially app-downgrading should be taken with a grain of salt since the outcome of that procedure is strongly dependent on the application and its handling of its private data. According to a blog article by Oxygen Forensics (Oxygen Forensics, 2022b), huge differences between OS versions and device manufacturers exist for this method, which might lead to the inability to restore the app's initial state or even the loss of data. Accordingly, if this method is deemed necessary, an evaluation under similar environmental conditions should be carried out first.

Furthermore, one can question the viability of local backups in the future, since, as already mentioned, Android's local backup method has already been marked as deprecated. However, the method is still widely used as an acquisition method and the evaluation of its forensic suitability applies not only to future investigations but also serves as validation of past cases. If the ADB backend to create a local backup to a PC is ultimately removed from Android, it is still possible to copy the phone's data onto another device due to Android's data migration feature. Accordingly, the backup data might still be obtainable using a prepared device, that serves as an intermediate backup destination. However, the app-downgrading procedure might not be possible anymore. With iOS, on the other hand, it is unlikely for the local backup method to disappear in the foreseeable future. In general, some form of local backups will probably be around for quite some time, as there are still users with limited

internet access as well as those who prefer to handle their private data locally due to privacy concerns.

7. Conclusion and Future Work

Based on our evaluation model, including the data classification and categorization, we were able to make comprehensive observations about the backup processes of Android and iOS.

In summary, our developed evaluation procedure was successfully executed on two instances and provides, despite the described limitations, a better understanding of the validity of the data acquired during a backup acquisition. Especially for forensic purposes, where the data can determine the outcome of a legal case, accuracy is of particular significance.

Since we cannot make our dataset publicly available due to privacy reasons, one of our goals for future research is to repeat the experiments on a publishable yet realistic dataset. Furthermore, the scope of the practical evaluation can always be extended to more OS versions, device manufacturers, or apps. Accordingly, a fully automated test environment with the use of virtual devices would be a logical continuation.

Acknowledgements

Work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393541319/GRK2475/1-2019).

We specifically wanted to thank Janine Schneider, Jan Gruber, Frank Breitinger and Immanuel Lautner for their helpful comments and the useful discussion about the contents of the paper.

References

Ayers, R., Brothers, S., Jansen, W., et al., 2014. Guidelines on mobile device forensics .

Belkasoft, 2021. How to acquire data from an Android device using APK downgrade method. URL: https://belkasoft.com/Android_APK_downgrade_method. accessed: 2022-09-22.

Caithness, A., 2012. The forensic implications of sqlite’s write ahead log. Digital Investigation-Digital evidence: if it’s there, we’ll find it .

Cellebrite, 2019. Exclusive access to WhatsApp data and another 40 apps on Android devices. URL: https://www.cellebrite.com/en/productupdates/exclusive-access-to-whatsapp-data-and-another-40-apps-on-android-devices/. accessed: 2022-09-22.

Chang, Y.T., Teng, K.C., Tso, Y.C., Wang, S.J., 2015. Jailbroken iphone forensics for the investigations and controversy to digital evidence. Journal of Computers 26, 19–33.

checkra1n, . checkra1n Jailbreak for iPhone 5s through iPhone X, iOS 12.0 and up. URL: https://checkra.in/. accessed: 2022-09-22.

Dunham, S., 2021. iOS Backup Extraction. URL: https://github.com/dunhamsteve/ios. accessed: 2022-09-16, Commit: 1a78c2f935f112f30840e5d871805dfbb092348d.

Farley Forensics, 2019. Forensic Analysis of iTunes Backups. URL: https://farleyforensics.com/2019/04/14/forensic-analysis-of-itunes-backups/. accessed: 2023-03-29.

Freiling, F.C., Groß, T., Latzo, T., Müller, T., Palutke, R., 2018. Advances in forensic data acquisition. IEEE Des. Test 35, 63–74. URL: https://doi.org/10.1109/MDAT.2018.2862366, doi:10.1109/MDAT.2018.2862366.

Garfinkel, S., Nelson, A.J., Young, J., 2012. A general strategy for differential forensic analysis. Digital Investigation 9, S50–S59.

Grajeda, C., Breitinger, F., Baggili, I., 2017. Availability of datasets for digital forensics–and what is missing. Digital Investigation 22, S94–S105.

Hassan, M., Pantaleon, L., 2017. An investigation into the impact of rooting android device on user data integrity, in: 2017 Seventh International Conference on Emerging Security Technologies (EST), IEEE. pp. 32–37.

Khatri, Y., 2019. ADB keyvalue backups and the .data format. URL: https://www.swiftforensics.com/2019/10/adb-keyvalue-backups-and-data-format.html. accessed: 2022-08-17.

Klyubin, A., 2016. Disallow downgrading of non-debuggable packages. URL: https://android.googlesource.com/platform/frameworks/base/+/921dd75. accessed: 2022-09-22.

libimobiledevice, 2022. libimobiledevice A cross-platform FOSS library written in C to communicate with iOS devices natively. URL: https://libimobiledevice.org/. accessed: 2022-09-22.

Lorentz, P., Mahalik, H., 2022. Android Data Collection Simplified. URL: https://cellebrite.com/en/android-data-collection-simplified/. accessed: 2022-09-11.

Magnet Forensics, 2019. How to Image a Smartphone with Magnet ACQUIRE. URL: https://www.magnetforensics.com/resources/image-smartphone-magnet-acquire/. accessed: 2022-09-12.

Mahalik, H., 2021. How To Find iTunes Backup Data. URL: https://cellebrite.com/en/how-to-find-itunes-backup-data/. accessed: 2022-09-12.

Manral, B., Somani, G., Choo, K.R., Conti, M., Gaur, M.S., 2020. A systematic survey on cloud forensics challenges, solutions, and future directions. ACM Comput. Surv. 52, 124:1–124:38. URL: https://doi.org/10.1145/3361216, doi:10.1145/3361216.

Oxygen Forensics, 2021. Android App Downgrade. URL: https://blog.oxygen-forensic.com/android-app-downgrade/. accessed: 2022-08-29.

Oxygen Forensics, 2022a. Android Extraction Updates in Oxygen Forensic® Detective 14.2. URL: https://blog.oxygen-forensic.com/android-extraction-updates-in-oxygen-forensic-detective-14-2/. accessed: 2022-09-11.

Oxygen Forensics, 2022b. Downgrade Method: what should be known before the procedure. URL: https://blog.oxygen-forensic.com/downgrade-method-what-should-be-known-before-the-procedure/. accessed: 2023-01-12.

Python Software Foundation, 2022. difflib - Helpers for computing deltas. URL: https://docs.python.org/3/library/difflib.html. accessed: 2022-09-11.

Roussev, V., Ahmed, I., Barreto, A., McCulley, S., Shanmughan, V., 2016. Cloud forensics-tool development studies & future outlook. Digit. Investig. 18, 79–95. URL: https://doi.org/10.1016/j.diin.2016.05.001, doi:10.1016/j.diin.2016.05.001.

SalvationDATA, 2020. [Case Study] Mobile Forensics: Forensic Data Extraction from Android Devices Using ADB (Android Debug Bridge) Part III. URL: https://blog.salvationdata.com/2020/08/07/case-study-mobile-forensics-forensic-data-extraction-from-android-devices-using-adb-android-debug-bridge-part-iii/. accessed: 2023-03-29.

Son, N., Lee, Y., Kim, D., James, J.I., Lee, S., Lee, K., 2013. A study of user data integrity during acquisition of android devices. Digital Investigation 10, S3–S11.

CRediT authorship contribution statement

Julian Geus: Conceptualization, Methodology, Investigation, Writing - Original Draft, Writing - Review and Editing. **Jenny Ottmann:** Conceptualization, Methodology, Investigation, Writing - Original Draft, Writing - Review and Editing. **Felix Freiling:** Conceptualization, Methodology,

Investigation, Writing - Original Draft, Writing - Review and Editing, Supervision.