



Digital forensic analysis of the private mode of browsers on Android [☆]

Xosé Fernández-Fuentes ^{*}, Tomás F. Pena, José C. Cabaleiro

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS), Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, 15782 Santiago de Compostela, Spain

ARTICLE INFO

Keywords:

Android forensics
Mobile device forensics
Digital forensics
Browsing artifacts
Private browsing
Internet privacy

ABSTRACT

The smartphone has become an essential electronic device in our daily lives. We carry our most precious and important data on it, from family videos of the last few years to credit card information so that we can pay with our phones. In addition, in recent years, mobile devices have become the preferred device for surfing the web, already representing more than 50% of Internet traffic. As one of the devices we spend the most time with throughout the day, it is not surprising that we are increasingly demanding a higher level of privacy. One of the measures introduced to help us protect our data by isolating certain activities on the Internet is the private mode integrated in most modern browsers. Of course, this feature is not new, and has been available on desktop platforms for more than a decade. Reviewing the literature, one can find several studies that test the correct functioning of the private mode on the desktop. However, the number of studies conducted on mobile devices is incredibly small. And not only is it small, but also most of them perform the tests using various emulators or virtual machines running obsolete versions of Android. Therefore, in this paper we apply the methodology we presented in a previous work to Google Chrome, Brave, Mozilla Firefox, and Tor Browser running on a tablet with Android 13 and on two virtual devices created with Android Emulator. The results confirm that these browsers do not store information about the browsing performed in private mode in the file system. However, the analysis of the volatile memory made it possible to recover the username and password used to log in to a website or the keywords typed in a search engine, even after the devices had been rebooted.

1. Introduction

The smartphone has become a device that does not leave us at any part of the day: it is ready to wake us up in the morning as well as to notify us that it is time to go to bed. It helps us on a personal level to keep in touch with family and friends or to permanently save special moments in the form of photos or videos. It helps us in a professional level to organize our calendar and not miss any important meeting or to be aware of any problem or emergency in our company. And, of course, it also helps us when we want to “disconnect”, watching a funny video, reading a book, or listening to music on our favorite streaming service.

Today, more than ever, we perform most activities that require an Internet connection with a smartphone. In 2021, Internet traffic generated by mobile devices, whether smartphone or tablet, was over 57%

according to Statcounter (2023). This figure represents a substantial increase considering that in 2016 it was less than 49%, in 2013 it was less than 21% and, in 2011, it only represented 7%. In terms of the popularity of mobile operating systems, 70% of mobile devices were running Android in 2021.

Being a device that is with us practically all day long with permanent Internet connection, users are increasingly demanding better guarantees in terms of security and privacy. This call for attention to privacy and security is thanks to relatively recent scandals such as the Edward Snowden's revelations (Greenwald and MacAskill, 2013; Greenwald, 2013), the Cambridge Analytica scandal (Thorbecke, 2019), or the big Equifax data breach (Berghel, 2017). Events that make us realize that our data may not be as protected as it should be or that applications do not keep their word not to collect information about

[☆] This work has received financial support from the Consellería de Cultura, Educación e Ordenación Universitaria of the Xunta de Galicia (accreditation 2019-2022 ED431G-2019/04, reference competitive group 2022-2024, ED431C 2022/16) and the European Regional Development Fund (ERDF), which acknowledges the CITIUS-Research Center in Intelligent Technologies of the University of Santiago de Compostela as a Research Center of the Galician University System. This work was also supported by the Ministry of Economy and Competitiveness, Government of Spain (Grant No. PID2019-104834 GB-I00). X. Fernández-Fuentes is supported by the Ministerio de Universidades, Spain under the FPU national plan (FPU18/04605).

^{*} Corresponding author.

E-mail addresses: xosefernandez.fuentes@usc.es (X. Fernández-Fuentes), tf.pena@usc.es (T. F. Pena), jc.cabaleiro@usc.es (J.C. Cabaleiro).

users. It is thanks to these wake-up calls that most smartphones enable device encryption by default (Krishnan et al., 2019), the most popular messaging apps incorporate end-to-end encryption (Afzal et al., 2021; Endeley, 2018; Keshvadi et al., 2020), the latest versions of Android and iOS include more granular control when granting permissions to an app (Almomani and Al Khayer, 2020; Kollnig et al., 2022; Wijesekera et al., 2017), the European Parliament introduced the General Data Protection Regulation (GDPR) (Voigt and Von dem Bussche, 2017) in 2016, the percentage of web pages using HTTPS increased from 25% in 2014 to more than 80% by 2023 (Let's Encrypt, 2023), and so on and so forth. All this thanks to the fact that users are gradually becoming more aware of the problem and are demanding a higher level of security and privacy. Because we must not forget that, as Glenn Greenwald said (Miles, 2014), “we all need places where we can go to explore without the judgmental eyes of other people being cast upon us [...] It's really in the private realm where dissent, creativity and personal exploration lie.”

A feature added to browsers aimed at providing a higher level of privacy is the private mode. This browsing mode promises that no data related to the navigation performed will be stored on the device. It is a really useful feature when using a shared computer or to avoid the collection of browsing habits. In addition, some browsers activate extra functions when this mode is used, such as blocking known trackers by default or setting the Do Not Track header. Of course, the private mode is not a feature that has been added recently to browsers. For example, Google Chrome incorporated incognito mode more than 10 years ago, at the end of 2008. This feature seems to have been well received by users, since according to a study (Habib et al., 2018) published in 2018, an increasing number of users make use of the private mode, and not only for privacy reasons, but also for security and practical reasons. The problem is that, as stated in DuckDuckGo (2017), the protection provided by this browsing mode is usually overestimated.

Different papers can be found in the literature that test the correct functioning of the private mode of the different browsers. The vast majority of them are focused on testing the private mode on the desktop version of the browsers, while the number of papers focused on the mobile versions is much smaller. Not only is it smaller, but there is no work that has performed the tests on a real Android device in recent years. The only work, as far as we know, that did the tests on a real device was (Al Barghouthy et al., 2013). It was published 10 years ago, only tested a single browser, did not analyze the contents of the volatile memory, and the browsing session performed was incredibly small.

The two most recent works (Younis et al., 2021; Thompson, 2022) did not perform the tests with a real device. One of them used an emulator while the other used a virtual machine. The emulator used by (Younis et al., 2021) was Nox (2023). This emulator is focused on running Android games on a system running Windows or macOS. The modifications made to the Android system by the Nox company are not public, so it is very difficult to know the type of optimizations or changes made to the operating system. This makes it impossible to know whether, from a security and privacy point of view, it represents a realistic environment comparable to using a physical device. The other problem with this emulator is that the latest version available is based on Android 9. This version of Android was first released in August 2018 and is currently discontinued (endoflife.date, 2023). The test environment used by the second paper mentioned (Thompson, 2022) was a VirtualBox virtual machine running Android-x86 (2022). Android-x86 is a community-driven open source project aimed at enabling Android to run on an x86 system. The latest version available is 9.0-r2, based on Android 9. However, this project is currently discontinued. The main problem with using virtualized environments or emulators is the evidence acquisition methods used. In this last mentioned work (Thompson, 2022), the authors create an image of the virtual machine disk using a tool (Exterro, 2023) designed to create forensic images of computer data. There is no equivalent tool for mobile devices. So this scenario is not realistic, since as the authors themselves acknowledge, “achieving results similar to

the research findings using physical Android devices may prove challenging [...] even with root access, it is very difficult to gain a full disk image that includes unallocated space”.

Another important point to emphasize about these recent works, apart from the fact that they use an obsolete version of the operating system, is that both of the solutions used (NoxPlayer and Android-x86) are neither official nor supported by Google. The main architecture of Android is ARM64, and there is no official support for x86 processors (Reidt, 2022). There is not even official support for ARM32, with Google forcing all apps available in the Play Store to be 64-bit versions as of August 2021 (Sims, 2021). Not only that, but the latest device released by Google, the Google Pixel 7, is not even capable of running 32-bit apps (Wang, 2022). Therefore, one of the points we pursued when performing the tests is that they were performed on a physical and current device.

The main objective of this work is to perform a forensic analysis of the private mode included in the mobile version of the most popular web browsers. The test environment consisted of a real mobile device and two virtual devices created with the official Android emulator. In this way, it was possible to verify to what extent the results can be extrapolated when running the browsers on two completely different architectures, a physical one (ARM64) and an emulated one (x86_64).

The starting point of the analysis was to have the device to be studied unlocked, or if it had some kind of lock configured, it was assumed that the unlock code was known. This starting point may not seem feasible. However, referenced below are two scenarios, one of them happening this 2023, where it is not a problem that the device is locked, as there are ways to get it in the desired state. The first one is how the creator of Silk Road was captured. The law enforcement agents had to obtain the computer the moment the creator logged in as an administrator on the Silk Road website. It did not matter that the device had full disk encryption or that the creator used really strong passwords. Law enforcement officers just had to wait for the right situation to snatch the unlocked computer out of his hands in a library (Mullin, 2015). In the other scenario, which can be read about this very year (Imran, 2023), the attackers position themselves strategically in a public place, such as a bar, and watch when the victim unlocks the phone in order to clearly see the unlock code. Then, the attackers simply steal the device at the right time. Google has already announced that Android 13 will include a feature called “Enhanced Pin Privacy” aimed at preventing this kind of over-the-shoulder attacks (Brinkmann, 2023).

Our goal in this paper is not to show how to capture the device to be studied, or how to discover the passcode, or to reveal a vulnerability that allows bypassing the lock screen. Our goal is to show what information an analyst, or an attacker, can retrieve from a private browsing session once the device falls into their hands. In this paper we aim to evaluate the behavior of private mode and answer questions such as the following: Is it possible to retrieve information about the navigation performed in private mode if the browser has been completely closed? Are there any artifacts left on the device about the browsing done if the device has been rebooted?

The tests performed are designed in such a way that they can be repeated on another device without the need for any specific hardware for evidence acquisition. It does not matter if the device is fully encrypted, as our starting point assumes that the passcode is known. Of course, if the passcode is unknown, and it is assumed that no vulnerability exists to bypass the lock screen, then the probabilities of recovering useful information from the device are close to zero. However, although this type of vulnerability is not very common, just a few months ago Google fixed a vulnerability in its Pixel phones that allowed the lock screen to be bypassed completely (Schütz, 2022).

As mentioned above, our test environment consisted of one physical device and two virtual devices running Android. When choosing the browsers, we wanted our tests to cover the following points: testing the most popular browser, testing browsers with different engines, and testing browsers that are focused on user privacy. The first browser selected

was Google Chrome due to having a market share of over 60% according to StatCounter. The second browser selected for its growth in recent years, and for being focused on protecting users' privacy, is Brave. These two browsers are based on the free and open-source software project Chromium, whose engine is Blink and uses V8 as its JavaScript engine. To include browsers with different engines, and thus add more variety to the tests, we decided to add Mozilla Firefox and Tor Browser. Mozilla Firefox uses the Gecko engine and SpiderMonkey as its JavaScript engine. Tor Browser is based on Mozilla Firefox and its main feature is anonymity, making use of the Tor network to achieve it. The idea behind testing Tor Browser was to determine whether the Tor Project team had made changes that would make it difficult to obtain evidence in a forensic analysis.

In order to test the private mode of the selected browsers in a consistent and exhaustive way, we applied the methodology we presented in a previous work (Fernández-Fuentes et al., 2022). In this paper, a methodology to perform a forensic analysis was presented and applied to Mozilla Firefox and Google Chrome browsers running on a Linux machine and on a virtual machine also running a Linux distribution.

The rest of the paper is structured as follows: Section 2 summarizes previous work related to forensic analysis of the Android operating system; Section 3 describes the methodology used to analyze the four browsers mentioned above in the three Android environments; the results obtained and their discussion are covered in Sections 4 and 5, respectively; finally, Section 6 contains the closing thoughts.

2. Related work

In the literature, a wide variety of studies can be found that performed digital forensics on different applications running on the Android operating system. For example, there are numerous studies that analyzed different instant messaging applications such as IMO (Ababneh et al., 2017; Tri et al., 2018; Sudozai et al., 2018), WeChat (Zhou et al., 2015; Wu et al., 2017, 2018; Rath et al., 2018), Telegram (Satrya et al., 2016; Rath et al., 2018), WhatsApp (Shortall and Azhar, 2015; Rath et al., 2018), Twitter (Wu et al., 2018), or Viber (Rath et al., 2018). There are also works that analyzed email, like (Umar et al., 2019) and (Chen and Mao, 2016), or other studies, such as (Apostolopoulos et al., 2013), that studied whether it was possible to recover the authentication credentials of a total of 30 applications by dumping the memory allocated to those processes.

When it comes to web browser forensics, the number of papers focusing on mobile platforms, concretely on Android, is quite small, especially when compared to the number of papers analyzing desktop platforms. The following paragraphs include, in chronological order, some of the most relevant and recent studies that perform forensic analysis of different browsers running on Android.

In 2013, Nedaa Al Barghouthy et al. (2013) tested the Orweb browser running on a Samsung Galaxy S2 smartphone with Android 2.3.3. Their goal was to determine what information could be retrieved from the file system after using the browser. The analysis revealed that it was not possible to recover any artifacts related to browsing on the stock device due to the lack of permissions to access sensitive folders. However, with the device rooted, they were able to access the entire file system, which allowed them to recover some activities performed with Orweb from the browser profile folder.

In 2014, Nedaa Al Barghouthy and Andrew Marrington (2014) repeated their previous experiment using the idea presented by Timothy Vidas et al. (2011). The problem with many of the commercial tools that allow physical acquisition of a device is that they require root privileges. By rooting a device, the system and user partitions are being modified, which could be destroying important evidence. Therefore, the approach of Timothy Vidas et al. (2011) was to replace the recovery partition with the necessary tools to obtain the evidence from the device, thus avoiding modifying the rest of the partitions. With this approach, Nedaa Al Barghouthy et al. first tested the Orweb browser on

a stock device, but with the recovery partition modified, and then repeated the tests with the device rooted. Their conclusion is that, given almost identical results, it is preferable not to root the device. However, they also emphasize that their analysis did not include the acquisition of the device's memory, which would have required root privileges.

In 2021, L. B. Younis et al. (2021) tested Google Chrome, Mozilla Firefox, Dolphin, and Opera running on a rooted Android emulator called NoxPlayer. Each of the browsers was tested in both normal and private mode. The browsing session performed consisted of visiting various websites and accessing Gmail to send and receive various emails. Evidence collection was limited to creating a memory dump immediately after the browsing session was completed. The analysis of the evidence with the Autopsy tool revealed that Google Chrome provided the lowest level of privacy, whereas Mozilla Firefox provided the highest level. It should be noted that having performed all tests on an emulator, extrapolation of the results to a real device may not be straightforward.

In 2022, Warren Thompson (2022) focused on determining what artifacts could be recovered from the file system after using the private mode of different browsers on Android. Specifically, the browsers analyzed were Google Chrome, Mozilla Firefox, DuckDuckGo, and Tor Browser running on a VirtualBox virtual machine with Android x86 version 9.0-r2. The tests consisted of accessing several web pages (bookmarking some of them), logging into several websites (saving the credentials in the browsers that allowed it), sending and receiving emails between two email accounts, and performing several searches in different online search engines. Once the browsing was complete, FTK Imager was used to create an image of the VM's hard drive. The analysis of the different images revealed that DuckDuckGo created the most artifacts, even allowing the recovery of images loaded during navigation. The best of the four browsers was Mozilla Firefox, as it generated the fewest artifacts on disk. The author concludes that several of the artifacts were recovered from unallocated space. Therefore, repeating this type of analysis on a physical device could be complicated, since obtaining a complete disk image is not straightforward, even with root privileges. In addition, it should be noted that Android has been using full disk encryption for several versions now, which makes it even more difficult to obtain information from unallocated space.

In the literature there are also different toolkits for Android that allow acquiring forensic artifacts from different applications in an automated way. For example, AndroKit (Asim et al., 2019) can be highlighted for the case of web browsers. AndroKit extracts from the device's file system different artifacts from Google Chrome, Opera, Mozilla Firefox, and Dolphin browsers. Some of the artifacts it is able to obtain are bookmarks, downloaded files, web history, user credentials, or stored sessions. In addition, AndroKit can root the device in order to successfully access the folders where the browsers store their profiles. A demonstration of how it works can be seen in (Asim et al., 2019), where the authors run AndroKit on an Android emulator and on two Samsung smartphones.

It is important to point out that this type of toolkits do not focus on testing the private mode or analyzing the contents of memory. Therefore, the amount of information they can retrieve from a private browsing session is incredibly small. They would only be able to retrieve information about the browsing performed in private mode if the browser left any artifacts in the file system.

3. Methodology

This section describes the methodology used to perform the tests, which was presented in detail in (Fernández-Fuentes et al., 2022). The methodology consists of five phases: environment setup, monitoring changes, browsing, data acquisition, and analysis. The following subsections describe each of them as well as the adjustments made to adapt them to the Android operating system.

3.1. Environment setup

This phase is intended to design and deploy the environment where the browser, or browsers, will be executed. The initial configuration that needs to be applied to the browser, or browsers, is also included in this phase.

To get a clearer picture of how the private mode behaves in different scenarios, the methodology recommends, when possible, the use of multiple environments in which to test the browsers. Among the possible environments, some options indicated are 1) selecting multiple operating systems, 2) using the same operating system but changing different low-level options (such as kernel parameters), and 3) using a virtualized and a bare metal system.

In this case, we decided to test the selected browsers in three different environments, one bare metal and two virtualized. For the bare metal environment we used a Samsung Galaxy Tab S6 Lite LTE (version SM-P615) running the factory operating system updated to the latest version available at the time of testing (Android 13 with kernel version 4.14.113). For the virtualized environments we decided to use the Android Emulator version 32.1.11.0 (Android Developers, 2023). Specifically, we created an Android Virtual Device (AVD) with Android 13 (API 33 and ABI x86_64) and another with Android 9 (API 28 and ABI x86_64), both using the hardware profile of a Pixel 4. The reason for creating these two particular AVDs is because of the similarities they have with the Samsung tablet. The Android 13 emulator runs the same version of Android as the tablet and the Android 9 emulator uses a kernel version of the same branch as the tablet. Both AVDs were executed on a Windows 11 version 22H2 (OS Build 22621.1413) running on a computer with the following characteristics: an Intel Core i7-9700K processor, 32 GB RAM, and a 1 TB Intel NVMe SSD.

Although the use of this type of emulators is not intended to be used by the end users of the applications, we believe that it provides a new perspective when analyzing how browsers behave in different circumstances. It should not be forgotten that the most recent work in this area (such as Younis et al., 2021 and Thompson, 2022) uses some sort of emulator. Therefore, adding a bare metal environment in our experiments allows us to check to what extent the results obtained in an emulator can be extrapolated to those obtained in a real device.

When preparing a device for testing, the following considerations should be taken into account:

1. Use a “clean” device. For this it is recommended that the device has been reset to factory settings. This will ensure that the artifacts found in the evidence must have been created by the browser being tested and not by another application or by previous use. Once reset, install the necessary updates as well as the desired versions of the browsers to be tested.
2. Disable automatic updates of both the operating system and applications. The reason for this is to prevent operating system and browser versions from changing between different executions, which could lead to inconsistent results.

Once the tablet was updated and configured and the AVDs were created, the following browsers were installed:

1. Google Chrome 104.0.5112.97
2. Brave 1.49.132
3. Mozilla Firefox 111.1.1
4. Tor Browser 102.2.1-Release (12.0.4)

One thing that is important to point out is that the first three browsers have a normal browsing mode as well as a private browsing mode. However, the Tor Browser only allows browsing in private mode. This is because this browser is designed to anonymize the generated network traffic (thanks to the use of the Tor network) as well as to ensure that no browsing related information is stored on the device used.

Table 1

Login information stored in the keychain of each of the browsers tested.

Web site	Username	Password
https://www.reddit.com	Test9861	oneRandomPassword41+
https://www.reddit.com	Test9862	oneRandomPassword42+
https://forums.linuxmint.com	Test99172	oneRandomPassword46+

3.1.1. Browser setup

Once the test environment is ready, it is necessary to prepare the selected browsers. The only requirement for applying the methodology is to add, at least, one username and password to the browser keychain. The reason for this requirement will become clearer when the predefined browser session is described in Section 3.3.

Of course, the methodology does not restrict the configuration to be used. For example, to check that the “delete browsing data on exit” function really works correctly, independent runs can be performed with this option enabled and disabled. Once the runs have been carried out, taking the appropriate precautions between runs to avoid contaminating the evidence, the results can be compared to check if this option decreases the amount of information that can be retrieved.

In this case, the configuration used in each of the browsers was the default. The only thing that was changed was to add several usernames and passwords to the keychain of each of the browsers. Specifically, the entries that can be seen in Table 1 were stored.

The procedure for inserting the entries differs from browser to browser. In the case of Google Chrome and Brave it was necessary to access each of these websites without using the incognito mode and log in. Once logged in, the browser asked if the credentials should be saved. From the incognito or private mode, these browsers did not allow information to be saved to the keychain, which makes sense, since saving this information would leave a record on the device that the website was accessed. Mozilla Firefox also asked whether the credentials used to log in to a website should be stored in the keychain. However, unlike Google Chrome or Brave, Mozilla Firefox allowed new credentials to be saved in both normal and private mode. Alternatively, Mozilla Firefox offers the possibility to manually add entries to the keychain from the preferences. In the case of the Tor Browser, the only way to save new credentials was to manually enter them from the preferences, as it did not ask if the credentials should be saved when logging in to a website.

Once the browsers were configured, the cache, history, and cookies were deleted from each browser. This step is especially important in the case of Google Chrome and Brave, since it was necessary to access the different websites to store the login information.

3.2. Monitoring changes

To capture the artifacts created by the browser, it is necessary to be able to detect changes in the file system as well as to be able to dump the device’s RAM. Of course, the tools to be used will vary depending on the operating system selected.

3.2.1. Dumping the memory

It is very important to select a tool that completely dumps the volatile memory of the device. It is not valid to use a tool that only dumps the memory space associated with a certain application. There are two reasons for this:

1. If only the memory space associated with an application is dumped, memory areas that were associated with the application but were freed are not going to be dumped. These regions may not yet have been overwritten and, therefore, may still contain interesting information.
2. With this kind of tools it is not possible to obtain a memory dump when the application has ended. Therefore, they are not useful for evaluating which artifacts can be recovered after the browser has been closed, or after the device has been restarted.

To dump the memory of the tablet intended for testing we used the Samsung Upload Client (Kerler, 2023) tool developed by B. Kerler. This tool is based on the work of Arstenstein and Goldman (2017), who discovered that they could execute arbitrary code in the Samsung Bootloader and developed a tool (sboot_dump, 2017) to dump the memory of a Samsung device. The only requirement to use the tool is to enable the built-in debugging function called S-Boot Upload Mode on the device. To do so, it is necessary to dial *#9900# and, in the pop-up screen, set the Debug Level option to High. Once activated, the procedure to dump the memory is as follows:

1. Force reset the device by pressing “Power” + “Volume down” buttons.¹
2. As soon as the screen turns off, press “Power” + “Volume up” until the device enters the S-Boot Upload Mode.
3. Connect the device to a computer.
4. Run the Samsung Upload Client tool (Kerler, 2023) with the desired options to start the dump. The dump can be of a specific range, of specific areas, or of the entire memory.

The main advantage of this method is that it does not require rooting the device or replacing the kernel. Moreover, once the debugging option is enabled, the memory dump process can be started with the device locked. Of course, there are other options for dumping the memory of an Android device, such as, LiME (Sylve, 2012; Sylve et al., 2012), AMExtractor (Yang et al., 2016), or AMD (Yang et al., 2017).

LiME is a Loadable Kernel Module that dumps the RAM to a file when it is loaded. The main difficulty in using it on Android is that it requires unlocking the device’s bootloader as well as recompiling the kernel to allow loading modules, as this option is disabled by default.

AMExtractor makes use of the virtual device /dev/kmem, avoiding the need to load a module in the kernel. Still, it requires root privileges to be able to access that virtual device. However, the main limitation of this tool is that the /dev/kmem device is not available on 64-bit ARM architecture (AArch64). Therefore, this option will be valid depending on the Android device being used.

The AMD tool takes advantage of the firmware update protocols to dump the memory of an Android device. The main advantages are that there is no need to root the device or use a custom kernel, no debugging options need to be enabled, it works even if the device is locked, and the dump process can be started without rebooting the device. The only drawback is the compatibility with different devices, as the tool has only implemented support for certain Samsung and LG devices.

To dump the memory of the two AVD instances, instead of using a third-party tool, we made use of the functionality provided by Windows 11 for this task. Specifically, the steps to be performed to obtain a memory dump of a particular process are as follows: open the Windows Task Manager in the Details tab, find the corresponding process and select the “Create dump file” option from the context menu. In this case, the process from which the memory was to be dumped was qemu-system-x86_64.exe.

3.2.2. Monitoring changes in the file system

When monitoring disk changes, it is recommended to choose a tool that supports specifying the directories to monitor as well as the events to capture. The reason for limiting the directories to be monitored is basically to try to eliminate noise generated by other applications and thus reduce subsequent analysis tasks.

In this case, the inotifywait for Android (2016) tool was used to monitor disk changes. This version is based on the inotifywait tool, which is included in the inotify-tools (2022) package, but adapted for Android. Its operation and configuration is the same as the original tool.

¹ The buttons to be pressed may vary from one device to another.

Table 2
The path where the user profile is located in the file system for each of the browsers tested.

Browser	Profile path
Google Chrome	/data/data/com.android.chrome
Brave	/data/data/com.brave.browser
Mozilla Firefox	/data/data/org.mozilla.firefox
Tor Browser	/data/data/org.torproject.torbrowser

The only required parameter when using inotifywait is to specify the directory, or directories, to be monitored. In this particular case, the directories to be watched were the folders where the different browsers store the user profiles. Specifically, the path monitored for each of the selected browsers can be seen in Table 2.

Taking advantage of the fact that the tool offers the possibility of restricting the types of events to be monitored, we decided that only the following ones would be recorded: creation of files or directories, modification of files, and deletion of files or directories. Thanks to this option, the subsequent processing of the log file is simplified by avoiding the recording of “innocuous” events, such as opening a file to read its contents or closing a file that was open in read-only mode.

With a stock device it is not possible to access the folders mentioned in Table 2. The problem is that without this access it is not possible to monitor the changes made by the browsers in the file system. Therefore, in order to successfully complete this phase, the tablet was rooted in order to be able to run the aforementioned monitoring tool on the directories containing the browser profiles. In the case of the AVDs, no additional modifications were necessary because the AVDs created correspond to the versions without the Play Store, which allow root access by default.

3.3. Browsing

This phase of the methodology is intended to design a browsing session that will be subsequently performed in the different browsers in all deployed environments. Thanks to having a predefined session, it can be easily repeated while facilitating the subsequent analysis of the evidence.

The browsing session to be carried out in private mode has to follow the scheme below:

1. Access a web page that hosts music or videos. Use the search engine and play one of the elements returned by the search. The reason behind this point is to answer the following question: Are the words typed in the search box as well as the element that was reproduced in the captured evidence?
2. Open a new tab and access a website that hosts at least one cookie in the browser. The reason for this point is to determine if the cookies created by the website are in the collected evidence.
3. Open a website that hosts some kind of file that can be previewed in the browser itself without the need to download it. Is it possible to recover the preview file in the analysis phase?
The original methodology indicates, for reference, to preview some PDF available online. However, this type of file is not valid for Android. None of the browsers tested currently allow you to preview a PDF file. The only option they offer is to download the file and open it with another application, which is not a valid solution for this situation. Therefore, in the case of Android it is recommended to preview another type of file, such as an audio or video file, or even a photograph.
4. Open a new tab and type a URL in the address bar. Once typed, and without having visited the site, delete the entered URL. The reason for this point is to answer the following question: Is the URL typed in the evidence?
It is important to select a URL that does not generate false positives. For example, if the URL [google.com](https://www.google.com) is selected, it will be

easily found in the subsequent analysis because it is one of the default search engines integrated in the browsers. Therefore, it is advisable to select a URL that does not return any occurrences prior to conducting the browsing session.

5. Open a new tab and access the website whose credentials were stored in the [Browser setup](#) phase in the browser keychain. Log in with the saved credentials. The idea behind this point is to try to determine if it is possible to recover the complete database of users and passwords from the memory dumps.
6. Open a new tab and access a web page where login is required. Obviously this page has to be different from the previous one and for which no credentials are stored in the browser's keychain. After accessing the website, enter the information requested by the page (username, password...). Is it possible to retrieve the information typed in the various fields of the login form in the analysis phase?

Following the above scheme, the browser session designed to run in the three Android environments is as follows:

1. Go to [youtube.com](https://www.youtube.com) and search kernel bugs. Play the video with the title Syzbot and the Tale of Thousand Kernel Bugs - Dmitry Vyukov, Google. Pause the video after 15 seconds.
2. Open a new tab and access to github.com.
3. Open a new tab and access to <https://file-examples.com/index.php/sample-audio-files/sample-mp3-download/>. Click on the "Download sample MP3 file" button corresponding to the 700 KB option. Pause the audio after 5 seconds.
4. Open a new tab and write myurl.com in the address bar. Without accessing this website, delete the written URL.
5. Access to [reddit.com/login](https://www.reddit.com/login). Log in using the credentials stored in the browser keychain. Specifically, use the credentials of the user Test9861. It is important to point out that we only used the first (of the three) credentials shown in Table 1.
When the Tor Browser was used, this step could not be completed successfully in any of the runs. This is because it does not allow the use of the stored credentials. So, when Tor Browser was being tested, the [reddit.com/login](https://www.reddit.com/login) website was simply accessed and no further action was taken.
6. Open a new tab and go to [google.com/gmail](https://www.google.com/gmail). Click on Sign in. Try logging in using virtual112233@gmail.com as username and @thisis4testing1 as password. The login will fail.

3.4. Data acquisition

This phase is intended for the acquisition of evidence for subsequent analysis. Specifically, it is necessary to obtain several dumps of the entire memory at certain times as well as all the changes made by the browser in the file system. The runs where memory is dumped must be independent of the runs where disk changes are monitored. It is also important that only one browser is tested in each run.

3.4.1. Memory

To get a more comprehensive view of what information can be retrieved from memory at different points in time, it should be dumped in four different situations. Specifically, the steps to be performed to obtain each one of them are as follows:

- T1. Turn on the device, launch the browser in incognito mode, perform the browsing session, and dump the RAM.
- T2. Turn on the device, launch the browser in incognito mode, perform the browsing session, *close the browser, wait one minute*, and dump the RAM.
- T3. Turn on the device, launch the browser in incognito mode, perform the browsing session, *close the browser, restart the device*, and dump the RAM.

- T4. Turn on the device, launch the browser in incognito mode, perform the browsing session, *close the browser, turn off the device, wait 10 seconds, turn on the device, and, once started, dump the RAM*.

Each of the above runs must be completely independent. For example, it is not valid to create the first and the second dump in the same run. The reason is simply not to contaminate the results of one run with the next one.

3.4.2. File system

To obtain the changes made by the browser in the file system, the following procedure must be followed:

1. Launch the selected monitoring tool.
2. Perform the navigation session presented in Section 3.3.
3. Close the browser and stop the monitoring tool.

3.5. Analysis

This last phase is intended to process all the evidence captured in the previous phase. On the one hand, the different memory dumps obtained must be analyzed and, on the other hand, the changes made by the browser in the file system must be examined.

The task of extracting information from the memory dumps consists of two parts. In the first part, it is searched by a set of keywords associated with the navigation performed. For this purpose, a hexadecimal editor, such as wxHexEditor (2017), can be used or a script can be developed to automate the search process. In this case, a script was developed to search for a list of keywords. The output of this script is a CSV file indicating the number of occurrences found for each of the search terms in the different dumps.

Given the browsing session described in Section 3.3, the set of keywords selected was as follows:

1. kernel bugs
2. kernel%20bugs
3. kernel+bugs
4. kernel%2Bbugs
5. Syzbot and the Tale of Thousand Kernel Bugs - Dmitry Vyukov, Google
6. _gh_sess=
7. _octo=
8. myurl.com
9. Test9861
10. oneRandomPassword41+
11. Test9862
12. oneRandomPassword42+
13. Test99172
14. oneRandomPassword46+
15. virtual112233@gmail.com
16. virtual112233%40gmail.com
17. @thisis4testing1
18. %40thisis4testing1

The first four terms are related to the search performed in [youtube.com](https://www.youtube.com). Numbers 2 and 4 are the same as numbers 1 and 3 but replacing, respectively, the blank space and the + symbol by their corresponding HTML codes. Number 5 is the full title of the video played. Numbers 6 and 7 correspond to the name of the cookies set by the github.com website. Item 8 is the URL that was typed into the address bar but was not accessed. Items 9, 11, and 13 correspond to the usernames stored in the browser keychain. Items 10, 12, and 14 are the passwords corresponding to each of the above usernames in the keychain. The number 15 is the email address entered to log in to [google.com/gmail](https://www.google.com/gmail) and 16 is the same address but with the @ replaced by its HTML code.

Item 17 corresponds to the Gmail password entered, and the last item is the same password but with the @ replaced by its HTML code.

The second part of the analysis of the memory determines whether it is possible to retrieve various files directly from the dumps. In this particular instance, the files to be searched were the following: the MP3 file previewed in the browser and the files that store the contents of the keychain of the different browsers. A possible option to perform this task is to use the Volatility Framework tool (Volatility, 2016). Another option, which is the one we actually use, is to implement a program that, given a file, indicates whether that file is found in its entirety in the dump or not.

Once the memory analysis has been completed, the last part of this phase consists of determining whether there are any traces of the actions carried out in private mode on the disk. The best way to tackle this task is to develop a script that searches for the aforementioned keywords in each of the files indicated by the file system monitoring tool (inotifywait in this case). In addition, it is also advisable to run this search script on the directories where the browser profiles are stored.

4. Results

Applying the described methodology requires repeating the browsing session several times and capturing evidence at different times in each run. This implies that it is necessary to take precautions to avoid contaminating the results of the current run with artifacts left over from a previous run. There are several ways to avoid this problem, from a radical and time-consuming solution to a simpler and faster one. The first solution is the most radical and consists of starting each run with a completely clean device. To do this, the following steps must be performed before starting a new test:

1. If a physical device is to be used, it must be reset to factory settings. However, if an emulator is to be used, simply create a new virtual device.
2. Install and configure the browser to be tested.
3. Turn off the device or emulator completely. If a physical device is being used, it is important to wait at least one minute before turning it back on to ensure that the device starts with “clean” RAM.
4. Turn on the device or emulator and start the corresponding test.

The second solution is to uninstall and reinstall the browser to be tested. On Android, when an application is uninstalled, it automatically deletes all user data. Therefore, when it is reinstalled, there is no trace of the previous configuration. The steps to perform before starting a new run with this solution are listed below:

1. Uninstall the browser.
2. Reinstall the browser.
3. Configure the browser.
4. Turn off the device or emulator completely. If a physical device is being used, it is important to wait at least one minute before turning it back on to ensure that the device starts with “clean” RAM.
5. Turn on the device or emulator and start the corresponding test.

The third and last option is to use the cleaning options included in the browsers themselves. This option is the simplest and requires the least setup time of the three. In this case, the steps to be performed before a new execution are as follows:

1. Clear the cookies, history and cache using the option available in the browser preferences. If the browser tested in the previous run is different from the one to be tested next, this cleaning must be done in both browsers (the previous one and the one to be used next).

2. Turn off the device or emulator completely. If a physical device is being used, it is important to wait at least one minute before turning it back on to ensure that the device starts with “clean” RAM.
3. Turn on the device or emulator and start the corresponding test.

After testing the three solutions with the different browsers, we found no difference in the results obtained. The results presented in the following subsections are the result of running the same tests several times in order to corroborate the results. In order to be as efficient as possible in setting up the environment, we used a hybrid of two of the presented solutions. The first solution was used every time we started testing a new browser. For example, if we had just tested Brave and we were going to test Tor Browser, then we would restore the devices completely. However, when we were doing multiple tests of the same browser, then we used the third option. This means that we would use the clearing options provided by the browser itself.

4.1. Findings on the memory dumps

Tables 3, 4, and 5 contain the results of processing the dumps obtained with the browsers tested in scenarios T1, T2, and T3, respectively. The first part of the tables (Keyword searches) contains the number of matches for each search term described above. The second part of the tables (File recovery) indicates whether it was possible to recover the previewed MP3 file as well as the files containing the browser keychains.

The results obtained when analyzing the dumps created after the device was turned off for 10 seconds (scenario T4) reveal that it was not possible to recover any artifacts in any of the cases. The only exception was when running Google Chrome on the emulator with Android 9. In this case, 2 matches were found for each of the usernames stored in the browser keychain. In other words, 2 occurrences of Test9861, 2 of Test9862, and another 2 of Test99172 were found.

4.2. Findings from file system analysis

After examining all the files indicated by the selected monitoring tool, and also after repeating the searches in the directories where the browsers store the user profiles, no artifacts were found that revealed the actions performed in private mode.

One detail that caught our attention when processing the directories with the browser profiles is that both Google Chrome and Brave leave the usernames (not the passwords) stored in the browser’s keychain in clear text on disk. Specifically, in the file Login Data located in the directory containing the user profile. However, repeating these searches in the Mozilla Firefox and Tor Browser directories yielded no results. This point is not related to the operation of the private mode, but it is a difference that was found when analyzing the changes in the file system.

5. Discussion

In this section we discuss in detail the results obtained with the four browsers in the different situations. To facilitate the discussion, we will first focus on the memory analysis and then on the file system.

5.1. Memory

The analysis of memory dumps revealed numerous artifacts that made it possible to recover, in some situations, all the activities performed in private browsing mode. The following subsections discuss the information that was possible to retrieve from the memory dumps according to when they were created.

Table 3

Summary of the analysis of the memory dumps created with the browser still running (T1). The first part of the table (Keyword searches) shows the number of matches for each of the search terms. The second part (File recovery) shows whether the files containing the browser keychain and the MP3 file were found intact. To facilitate the reading of the table, instead of using a zero in cases where no match was found for that search term, a hyphen was used instead.

T1	Chrome			Brave			Firefox			Tor Browser		
	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13
Keyword searches												
kernel bugs	14	18	10	9	11	23	14	15	12	13	13	11
kernel%20bugs	4	4	-	6	-	11	5	5	5	5	-	5
kernel+bugs	40	42	130	35	31	46	15	14	11	15	23	18
kernel%2Bbugs	47	49	99	64	46	48	34	33	28	24	20	22
Syzbot and the Tale of...	27	26	23	24	25	27	16	13	14	14	14	13
_gh_sess =	-	1	1	-	1	3	2	2	2	2	2	2
_octo =	6	5	6	7	5	5	4	3	2	4	3	2
myurl.com	8	6	23	3	4	19	4	-	19	3	-	21
Test9861	39	38	44	29	74	73	27	34	44	-	-	-
oneRandomPassword41+	1	1	1	-	1	1	2	5	6	-	-	-
Test9862	2	2	2	2	2	2	4	9	17	-	-	-
oneRandomPassword42+	-	-	-	-	-	-	5	5	4	-	-	-
Test99172	2	2	2	2	2	2	-	-	-	-	-	-
oneRandomPassword46+	-	-	-	-	-	-	-	-	-	-	-	-
virtual112233@gmail.com	59	52	68	46	48	86	59	14	24	35	10	31
virtual112233%40gmail.com	6	5	6	5	7	7	9	9	12	10	9	10
@thisis4testing1	6	9	6	9	12	9	22	25	23	15	18	19
%40thisis4testing1	2	5	5	5	8	3	3	1	1	3	2	1
File recovery												
Browser keychain	Yes	No	No	No	No	No	No	No	No	No	No	No
file.mp3	No	No	No	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes

Table 4

Summary of the analysis of the memory dumps created one minute after closing the browser (T2). The first part of the table (Keyword searches) shows the number of matches for each of the search terms. The second part (File recovery) shows whether the files containing the browser keychain and the MP3 file were found intact. To facilitate the reading of the table, instead of using a zero in cases where no match was found for that search term, a hyphen was used instead.

T2	Chrome			Brave			Firefox			Tor Browser		
	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13
Keyword searches												
kernel bugs	9	13	18	7	12	10	12	11	3	12	12	13
kernel%20bugs	4	-	1	6	-	-	1	1	1	2	2	5
kernel+bugs	53	36	30	25	27	39	11	6	8	9	12	34
kernel%2Bbugs	56	51	43	69	49	51	28	18	15	7	8	20
Syzbot and the Tale of...	25	26	21	19	26	24	14	10	2	13	9	14
_gh_sess =	-	-	-	-	-	1	-	-	-	1	1	2
_octo =	3	5	3	5	6	6	2	1	1	2	2	2
myurl.com	7	3	18	3	-	19	3	-	19	3	-	21
Test9861	37	42	43	65	39	49	8	6	26	-	-	-
oneRandomPassword41+	1	1	1	-	1	1	2	1	2	-	-	-
Test9862	2	2	2	2	2	2	7	2	2	-	-	-
oneRandomPassword42+	-	-	-	-	-	-	2	2	2	-	-	-
Test99172	2	2	2	2	2	2	-	-	-	-	-	-
oneRandomPassword46+	-	-	-	-	-	-	-	-	-	-	-	-
virtual112233@gmail.com	51	27	27	18	65	42	56	9	4	18	4	11
virtual112233%40gmail.com	3	1	1	12	9	7	3	2	2	7	1	5
@thisis4testing1	10	4	-	9	10	4	13	9	1	11	16	7
%40thisis4testing1	4	2	2	5	4	6	2	1	-	2	1	-
File recovery												
Browser keychain	Yes	No	No	No	No	No	No	No	No	No	No	No
file.mp3	No	No	No	Yes	No	No	No	No	No	No	No	Yes

5.1.1. Memory dumps created in T1

In the T1 situation, the dumps are created after the entire browsing session has been completed and while the browser is still running. These dumps allow to establish a starting point with which to compare the

rest of the dumps, which are created when the browser is no longer running. In other words, the dumps created in T1 represent the best scenario when it comes to recovering the actions performed in private mode.

Table 5

Summary of the analysis of the memory dumps created after the device has been rebooted (T3). The first part of the table (Keyword searches) shows the number of matches for each of the search terms. The second part (File recovery) shows whether the files containing the browser keychain and the MP3 file were found intact. To facilitate the reading of the table, instead of using a zero in cases where no match was found for that search term, a hyphen was used instead.

T3	Chrome			Brave			Firefox			Tor Browser		
	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13	Samsung	Android 9	Android 13
Keyword searches												
kernel bugs	2	5	-	-	5	-	2	2	-	-	3	-
kernel%20bugs	1	4	-	3	4	-	-	-	-	-	-	-
kernel+bugs	3	20	1	2	16	-	-	2	1	-	-	-
kernel%2Bbugs	1	24	2	1	15	1	1	5	-	-	1	-
Syzbot and the Tale of...	7	5	-	-	6	-	-	1	-	-	1	-
_gh_sess =	-	-	-	-	1	-	-	-	-	-	-	-
_octo =	1	-	-	-	1	-	-	-	-	-	-	-
myurl.com	2	-	-	-	-	-	-	-	-	-	-	-
Test9861	9	8	-	1	4	-	2	5	-	-	-	-
oneRandomPassword41+	-	1	-	-	-	-	-	2	-	-	-	-
Test9862	-	4	-	-	2	-	1	1	-	-	-	-
oneRandomPassword42+	-	-	-	-	-	-	-	1	-	-	-	-
Test99172	-	4	-	-	2	-	-	-	-	-	-	-
oneRandomPassword46+	-	-	-	-	-	-	-	-	-	-	-	-
virtual112233@gmail.com	1	2	-	3	-	-	2	-	-	1	1	-
virtual112233%40gmail.com	-	-	-	-	5	-	2	-	-	4	-	-
@thisis4testing1	1	-	-	-	-	-	4	2	-	-	1	-
%40thisis4testing1	-	-	-	-	-	-	-	-	-	-	-	-
File recovery												
Browser keychain	No	No	No	No	No	No	No	No	No	No	No	No
file.mp3	No	No	No	No	No	No	No	No	No	No	No	No

The first activity performed was to access YouTube and watch a video. Regardless of the browser used, it was possible to retrieve the keywords typed into the search box as well as the full title of the video played. When designing the browsing session, it was decided to use the YouTube website. However, this test indicates that it is possible to retrieve the terms typed into a search box as well as to determine which result was selected, regardless of the website visited.

The second activity was to access github.com for the purpose of subsequently attempting to retrieve the cookies it stored in the browser. Specifically, when github.com is accessed, it creates three cookies: `_gh_sess`, `_octo`, and `logged_in`. In the analysis phase it was decided to search only for the first two cookies. The problem with the third cookie is the name. It is very generic and produces numerous false positives when searching by that name. This does not mean that the value of the third cookie is not in the dump, it would simply require a more targeted analysis, using regular expressions to retrieve the value, for example. Regarding the first cookie, it was not possible to recover it when using Google Chrome or Brave on the tablet, but it was possible to recover it in all other cases. Regarding the second cookie, it was possible to retrieve it in all situations.

The third activity consisted of listening to a MP3 file directly in the browser. As can be seen in Table 3, it was possible to retrieve the file when using Mozilla Firefox or Tor Browser. With Google Chrome, the file was not intact in the dumps obtained and, with Brave, it was only possible to recover it undamaged when running on the tablet, not on the emulators. Although a MP3 file was used for this case, this test demonstrates that it is possible to recover, in some cases, files preserved directly in the browser that were not written to disk.

The fourth activity was typing a URL in the address bar but not accessing it. Interestingly, it was possible to retrieve this URL in all cases except when Mozilla Firefox and Tor Browser were executed in the emulator with Android 9. One potential reason why, in some situations, it could be possible to retrieve this URL is that the browser constructs the entire URL in the background, for example, by appending `https://`. Another possible explanation could be that the browser constructs the

URL necessary to use the words entered as search terms in a search engine. For example, in the case of Google Chrome, one of the results obtained when searching for myurl.com in the dump was the URL that the browser built to search for that word in the Google search engine: <https://www.google.com/search?q=myurl.com&client=chrome-mobile&sourceid=chrome-mobile&ie=UTF-8>.

The next step performed with the browsers was to access reddit.com and log in with the credentials that had been saved in the keychains. As previously mentioned, Tor Browser does not allow the use of the saved credentials, which is perfectly reflected in the results, where it was not possible to retrieve any of the credentials stored in the browser. As indicated in Table 1, three credentials were stored in the keychain of each of the browsers, two for reddit.com (Test9861 and Test9862) and one for forums.linuxmint.com (Test99172). With the other three browsers that do allow use of the keychain, the credentials of user Test9861 were used to log in to reddit.com. In the case of Google Chrome, it was possible to retrieve all three usernames stored in the keychain but only the password for user Test9861 was found in clear text. In the case of Brave, it was also possible to recover the usernames. However, it was only possible to recover the password of the logged-in user when Brave was running in the emulators. Lastly, for Mozilla Firefox, the saved usernames and passwords for the reddit.com website were recovered from the dumps, but neither the username nor the password for the forums.linuxmint.com website was found. Therefore, it appears that Mozilla Firefox decrypts from the keychain all the credentials associated with a particular website, even if only one of them is actually used.

Continuing with the recovery of credentials, the next part of the analysis consisted of determining whether it was possible to recover the file that stores the browser's keychain. In the case of Google Chrome, it was possible to recover the entire file (Login Data) when it was running on the tablet. Looking at the source code of Chromium (2022) (the open-source project on which Google Chrome is based), it can be seen that, in the case of Android, all the credentials stored in the keychain are encrypted using the AES 128-bit algorithm in CBC mode,

using peanuts as password and saltysalt as salt. Therefore, in the case of Google Chrome, having access only to the memory dump, we were able to recover all the credentials stored in the keychain, including the two passwords, the other one for [reddit.com](#) and the one for [forums.linuxmint.com](#), which were not in clear text in the dump. Brave is also based on Chromium, so if the `Login Data` file had been recovered intact in any of the environments, it would also have been possible to obtain all stored credentials in the same way as in Google Chrome. In the case of Mozilla Firefox and Tor Browser, the file containing the keychain (`logins2.sqlite`) was not complete in any of the memory dumps, so obtaining all the information stored in the browser's keychain was not possible in this case either.

The last activity of the designed browsing session consisted of logging into Gmail. Regardless of the environment and the browser used, it was possible to recover both the username and password entered. Although we have specifically used Gmail, this test indicates that it is possible to retrieve the login information entered on any website.

5.1.2. Memory dumps created in T2

These dumps were created one minute after closing the browser. Comparing the results obtained at T1 (Table 3) with those obtained at T2 (Table 4), there are no major differences, being possible to recover practically all the browsing session performed.

Regardless of the environment and the browser, it was possible to retrieve, as in T1, the words typed in the YouTube search field as well as the title of the video played. When it comes to retrieving the cookies set by the [github.com](#) website, there are indeed differences in the results. In T2 only the first cookie (`_gh_sess`) could be retrieved when running Brave on the Android 13 emulator and in all Tor Browser runs in the different environments. However, the second cookie could be retrieved in all cases. Regarding the URL written in the address bar but not accessed ([myurl.com](#)), this one could be retrieved in the same cases as in T1 except that in this situation it was not possible to retrieve it when using Brave in the emulator with Android 9. The results obtained after searching for the usernames and passwords stored in the browser keychains after logging in to [reddit.com](#) are exactly the same as in T1. And the same is true when retrieving the login information entered in Gmail, it was possible to retrieve it in all cases. Finally, regarding file recovery, the result of recovering the files containing the browser keychain is identical to T1. However, when it comes to recovering the previewed MP3 file, it could only be found in the following two situations: when running Brave on the tablet and when using Tor Browser in the Android 13 emulator.

5.1.3. Memory dumps created in T3

As might be expected, the amount of information that could be recovered from the dumps created after rebooting the device (Table 5) is significantly less than in the previous cases, but it is by no means negligible.

The words typed in the YouTube search field could be retrieved in all cases except when Tor Browser was running on the tablet or on the emulator with Android 13. The title of the played video could only be retrieved in the following cases: with Google Chrome running on the tablet or on the Android 9 emulator and with Brave, Mozilla Firefox, and Tor Browser running on the Android 9 emulator. The first cookie set by [github.com](#) could only be retrieved when using Brave on the Android 9 emulator and, the second cookie, when browsing with Google Chrome on the tablet and with Brave on the Android 9 emulator. The URL typed in the address bar but not accessed could only be retrieved in the case of Google Chrome running on the tablet. The username used to log into [reddit.com](#) (Test9861) could be retrieved in the same cases as in T2, except that in T3 it was not possible to retrieve it in any of the runs with the Android 13 emulator. The password for this Reddit user could only be retrieved when using the Google Chrome and Mozilla Firefox browsers running on the Android 9 emulator. The name of the other Reddit user stored in the keychain (Test9862) could be

recovered when using Google Chrome, Brave and Mozilla Firefox in Android 9 emulator and with Mozilla Firefox running on the tablet. And, Test99172's password could only be retrieved when running Mozilla Firefox on the Android 9 emulator. Regarding the third user stored in the keychain for the [forums.linuxmint.com](#) site, it could only be retrieved with Google Chrome and Brave running on the Android 9 emulator. And, Test99172's password was not found in any scenario, just like in T1 and T2. Regarding the email address used to log in to Gmail, it could not be retrieved in the following cases: when using the Android 13 emulator and when running Mozilla Firefox on the Android 9 emulator. And, the password entered could only be retrieved when using the tablet with Google Chrome and Mozilla Firefox and when running the Android 9 emulator with Mozilla Firefox and Tor Browser. Finally, and unlike T1 and T2, in T3 no files could be recovered in any of the environments.

5.1.4. Memory dumps created in T4

As mentioned in Section 4, after having the device or emulator turned off for 10 seconds, the only situation where it was possible to retrieve anything was when running Google Chrome on the emulator with Android 9. This information does not provide any hints of the activities performed in private mode and is not particularly useful either, as this information is stored in clear text on disk.

5.2. File system

The main purpose of this work was to determine what information can be retrieved from an Android device after using a browser in private mode. The results show that none of the browsers tested wrote any browsing-related information to the file system. One point that is important to mention is that we cannot guarantee that the browsers have not actually stored anything related to the browsing performed, since there is a remote possibility that the browsers have stored some data using an encoding that we are unaware of. What we can state is that the only information we found were the usernames (not the passwords) stored in the keychains of the Google Chrome and Brave browsers. Mozilla Firefox and Tor Browser did not store usernames in clear text, so searching their directories did not yield any results.

Continuing with the keychains integrated in the browsers tested, it is worth noting the different behavior of each one of them. In the case of Google Chrome and Brave, they do not allow you to manually add passwords to the password manager. To save the login information for a particular site, it is necessary to log in to that site without using the incognito mode. After logging in, these browsers will ask if you want to save the login information. If you repeat these steps in incognito mode, the browsers will not allow the credentials to be stored in the keychain. This is the expected behavior, because if they allowed saving this information from incognito mode, it would leave a permanent record on disk that this website was visited. Another measure that Google Chrome and Brave include is that they do not allow passwords to be displayed in clear text in the password manager if the device does not have a screen lock mechanism.

The Mozilla Firefox browser allows you to manually add login information from the browser's preferences. It also allows you to add information to the keychain at the time of logging in to a website. The problem is that Mozilla Firefox asks if the login information should be saved to the keychain from both normal mode and private mode. Although Mozilla's support page (Mozilla, 2023) clearly indicates that "New passwords and bookmarks you create while using Private Browsing will be saved", perhaps Mozilla could add an informative message in Mozilla Firefox stating that storing such information from the private mode may be compromising the user's privacy. It must be taken into account that the main purpose of the private mode is that no trace of the browsing done is left on the device. If it allows saving login information, it creates a record on the hard disk that indicates that this website was visited. In addition, Mozilla Firefox allows you to view saved passwords

without having to configure a screen lock mechanism. When there is no PIN or unlock password configured, Mozilla Firefox displays a warning and advises you to configure it to prevent anyone from viewing the passwords. The only problem is that this message only appears the first time the password manager is accessed.

Tor Browser, like Mozilla Firefox, allows you to manually add login information from the browser preferences. However, it is not possible to use such information to log in when accessing the corresponding website. It also does not offer the possibility to save the login information when logging into a website. We have verified this behavior on the desktop version of Tor Browser using a PC running both Windows and Linux. On both operating systems, we discovered that it does not allow to manually add entries to the browser keychain and also does not ask if the login information should be saved when logging in to a website. In other words, in the desktop version, the Tor Browser keychain is completely disabled. So, it's a bit confusing that it allows to manually add entries to the keychain in the Android version, if they are not going to be usable. Finally, when no locking mechanism is configured and the keychain is accessed to view the passwords, Tor Browser displays the same warning as Mozilla Firefox, indicating that one should be configured to prevent anyone from viewing the passwords.

To conclude this section, it is of interest to note that the memory analysis was the one that actually allowed to retrieve information about the activity performed in private mode. And, in the particular case of the tablet used for testing, this means that it is not necessary to modify or alter the system by rooting it, since thanks to the debugging functions included, it is really easy to dump the memory content, as described in Section 3.2.1. This makes these results applicable "to everyday life", where it is not common for users to use their devices rooted or with a customized OS version.

6. Conclusion

This paper presents the results of conducting a forensic analysis of the private mode integrated in four different browsers running on three separate Android environments. The environments used consisted of an emulator running Android 13, a tablet with Android 13, and an emulator running Android 9. To perform this analysis, we apply the methodology presented in (Fernández-Fuentes et al., 2022), making minor adjustments to carry it out on a mobile platform.

The browsers tested were Google Chrome, Brave, Mozilla Firefox, and Tor Browser. The memory and file system were analyzed for any artifacts that would reveal the activities performed using the respective private mode of each of these browsers. All of them kept their word that no activity performed in private mode would be stored in the file system. However, memory analysis allowed retrieval of all or virtually all the browsing session, with sensitive information (such as usernames, email addresses, and passwords) being retrieved even after the devices were rebooted. Therefore, the only way to ensure that no artifacts remain in memory is to completely power off the device and wait some time before restarting it, as this was the only scenario in which it was not possible to recover any activity performed in private mode.

The analysis also included the retrieval of files previewed during the browsing session, the acquisition of the complete keychain content integrated in the browsers, as well as different tests that revealed the different behavior of the keychains themselves.

Of the four browsers tested, it can be highlighted that Mozilla Firefox and Tor Browser were the ones that left the fewest artifacts in memory, with Google Chrome being the one that allowed the greatest amount of information to be retrieved. As for the environments, the Android 13 emulator was the one from which the highest number of artifacts could be obtained, being the Android 9 emulator the most "private" environment. This also shows that the use of emulators for performing forensic analysis to Android apps might not be the best environment, as it may give a false sense of privacy or, on the contrary,

reflect that much more information can be obtained than is actually possible on a real device.

As a future work it can be highlighted to apply the methodology to more browsers to have a broader picture of how each of them manage the private mode. Another possible line would be to repeat all the tests performed here but on different Android devices or even test different ROMs to compare the results. And, to complete the analysis of the most widely used mobile operating systems globally, the methodology would have to be applied to iOS devices.

CRedit authorship contribution statement

Xosé Fernández-Fuentes: Conceptualization, Investigation, Methodology, Writing – original draft. **Tomás F. Pena:** Conceptualization, Methodology, Writing – review & editing. **José C. Cabaleiro:** Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

- Ababneh, Ahmad, Awwad, Mohammad Abu, Al-Saleh, Mohammed I., 2017. IMO forensics in Android and Windows systems. In: 2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA). IEEE, pp. 1–6.
- Afzal, Asmar, Hussain, Mehdi, Saleem, Shahzad, Shahzad, M. Khuram, Ho, Anthony T.S., Jung, Ki-Hyun, 2021. Encrypted network traffic analysis of secure instant messaging application: a case study of Signal messenger app. *Applied Sciences* 11 (17), 7789.
- Al Barghouthy, Nedaa, Marrington, Andrew, 2014. A comparison of forensic acquisition techniques for Android devices: a case study investigation of Orweb browsing sessions. In: 2014 6th International Conference on New Technologies, Mobility and Security (NTMS). IEEE, pp. 1–4.
- Al Barghouthy, Nedaa, Marrington, Andrew, Baggili, Ibrahim, 2013. The forensic investigation of Android private browsing sessions using Orweb. In: 2013 5th International Conference on Computer Science and Information Technology. IEEE, pp. 33–37.
- Almomani, Iman M., Al Khayer, Aala, 2020. A comprehensive analysis of the Android permissions system. *IEEE Access* 8, 216671–216688.
- Android-x86, 2022. Run Android on your PC. <https://www.android-x86.org>. (Accessed 17 April 2023).
- Android Developers, 2023. Run apps on the Android Emulator. <https://developer.android.com/studio/run/emulator>. (Accessed 17 April 2023).
- Apostolopoulos, Dimitris, Marinakis, Giannis, Ntantogian, Christoforos, Xenakis, Christos, 2013. Discovering authentication credentials in volatile memory of Android mobile devices. In: Conference on e-Business, e-Services and e-Society. Springer, pp. 178–185.
- Artenstein, Nitay, Goldman, Gilad, 2017. Exploiting Android S-Boot: getting arbitrary code exec in the Samsung bootloader. <https://hexdetective.blogspot.com/2017/02/exploiting-android-s-boot-getting.html>. (Accessed 17 April 2023).
- Asim, Muhammad, Amjad, Muhammad Faisal, Iqbal, Waseem, Afzal, Hammad, Abbas, Haider, Zhang, Yin, 2019. Androkit: a toolkit for forensics analysis of web browsers on Android platform. *Future Generation Computer Systems* 94, 781–794.
- Berghel, Hal, 2017. Equifax and the latest round of identity theft roulette. *Computer* 50 (12), 72–76.
- Brinkmann, Martin, 2023. Android 13's upcoming Enhanced Pin Privacy feature explained. <https://www.ghacks.net/2023/03/31/android-13s-upcoming-enhanced-pin-privacy-feature-explained>. (Accessed 17 April 2023).
- Chen, Long, Mao, Yue, 2016. Forensic analysis of email on Android volatile memory. In: 2016 IEEE Trustcom/BigDataSE/ISPA. IEEE, pp. 945–951.
- Chromium, 2022. Source code of the GetEncryptionKey function of the Chromium project. https://source.chromium.org/chromium/chromium/src/+main:components/os_crypt/os_crypt_posix.cc;l=44-45;drc=c497cafe9858c478aa9daa3061e345f0683e61d. (Accessed 17 April 2023).
- DuckDuckGo, 2017. A study on private browsing: consumer usage, knowledge, and thoughts. https://duckduckgo.com/download/Private_Browsing.pdf. (Accessed 17 April 2023).
- Endeley, Robert E., 2018. End-to-end encryption in messaging services and national security—case of WhatsApp messenger. *Journal of Information Security* 9 (01), 95–99.

- endoflife.date, 2023. Support schedule for Android OS. <https://endoflife.date/android>. (Accessed 17 April 2023).
- Exterro, 2023. FTK Imager. <https://www.exterro.com/ftk-imager>. (Accessed 17 April 2023).
- Fernández-Fuentes, Xosé, Pena, Tomás F., Cabaleiro, José C., 2022. Digital forensic analysis methodology for private browsing: Firefox and Chrome on Linux as a case study. *Computers & Security*, 102626.
- Greenwald, Glenn, 2013. NSA collecting phone records of millions of Verizon customers daily. *The Guardian*. <https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order>. (Accessed 17 April 2023).
- Greenwald, Glenn, MacAskill, Ewen, 2013. NSA Prism program taps in to user data of Apple, Google and others. *The Guardian*. <https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data>. (Accessed 17 April 2023).
- Habib, Hana, Colnago, Jessica, Gopalakrishnan, Vidya, Pearman, Sarah, Thomas, Jeremy, Acquisti, Alessandro, Christin, Nicolas, Cranor, Lorrie Faith, 2018. Away from prying eyes: analyzing usage and understanding of private browsing. In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pp. 159–175.
- Imran, Shujaa, 2023. Should you be worried about thieves stealing your iPhone passcode? <https://www.makeuseof.com/thieves-stealing-iphone-passcode-worry>. (Accessed 17 April 2023).
- inotify-tools, 2022. inotify-tools. <https://github.com/inotify-tools/inotify-tools>. (Accessed 17 April 2023).
- inotifywait for Android, 2016. inotifywait for Android. <https://github.com/dstmath/inotifywait-for-Android>. (Accessed 17 April 2023).
- Kerler, B., 2023. Samsung Upload Client. https://github.com/bkerler/sboot_dump. (Accessed 17 April 2023).
- Keshvadi, Sina, Karamollahi, Mehdi, Williamson, Carey, 2020. Traffic characterization of instant messaging apps: a campus-level view. In: *2020 IEEE 45th Conference on Local Computer Networks (LCN)*. IEEE, pp. 225–232.
- Kollnig, Konrad, Shuba, Anastasia, Van Kleek, Max, Binns, Reuben, Shadbolt, Nigel, 2022. Goodbye tracking? Impact of iOS app tracking transparency and privacy labels. *arXiv preprint*. [arXiv:2204.03556](https://arxiv.org/abs/2204.03556).
- Krishnan, Sundar, Zhou, Bing, An, Min Kyung, 2019. Smartphone forensic challenges. *International Journal of Computer Science and Security* 13 (5), 183–200.
- Let's Encrypt, 2023. Let's Encrypt stats. Percentage of web pages loaded by Firefox using HTTPS (data source: Firefox telemetry). <https://letsencrypt.org/stats/#percentage-loads>. (Accessed 17 April 2023).
- Miles, Kathleen, 2014. Glenn Greenwald on why privacy is vital, even if you 'have nothing to hide'. https://www.huffpost.com/entry/glenn-greenwald-privacy_n_5509704. (Accessed 17 April 2023).
- Mozilla, 2023. Private Browsing - Use Firefox without saving history. <https://support.mozilla.org/en-US/kb/private-browsing-use-firefox-without-history>. (Accessed 17 April 2023).
- Mullin, Joe, 2015. "I have secrets": Ross Ulbricht's private journal shows Silk Road's birth. <https://arstechnica.com/tech-policy/2015/01/silk-road-trial-fbi-reveals-whats-on-ross-ulbrichts-computer-in-open-court>. (Accessed 17 April 2023).
- Nox, 2023. NoxPlayer official website. <https://www.bignox.com/>. (Accessed 17 April 2023).
- Rathi, Khushboo, Karabiyik, Umit, Aderibigbe, Temilola, Chi, Hongmei, 2018. Forensic analysis of encrypted instant messaging applications on Android. In: *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, pp. 1–6.
- Reidt, Teresa, 2022. Android x86: How to choose the right Android OS for x86 systems. <https://emteria.com/blog/android-x86>. (Accessed 17 April 2023).
- Satrya, Gandeva B., Daely, Philip T., Shin, Soo Young, 2016. Android forensics analysis: private chat on social messenger. In: *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, pp. 430–435.
- sboot_dump, 2017. sboot_dump. https://github.com/nitayart/sboot_dump. (Accessed 17 April 2023).
- Schütz, David, 2022. Accidental \$70k Google Pixel lock screen bypass. <https://bugs.xdavidhu.me/google/2022/11/10/accidental-70k-google-pixel-lock-screen-bypass>. (Accessed 17 April 2023).
- Shortall, Adam, Azhar, M.A. Hannan Bin, 2015. Forensic acquisitions of WhatsApp data on popular mobile platforms. In: *2015 Sixth International Conference on Emerging Security Technologies (EST)*. IEEE, pp. 13–17.
- Sims, Gary, 2021. 32-bits is dead: here's what it means for Android, Apple, and more. <https://www.androidauthority.com/arm-32-vs-64-bit-explained-1232065>. (Accessed 17 April 2023).
- Statcounter, 2023. Statcounter Global Stats. <https://gs.statcounter.com>. (Accessed 17 April 2023).
- Sudozai, M.A.K., Saleem, Shahzad, Buchanan, William J., Habib, Nisar, Zia, Haleemah, 2018. Forensics study of IMO call and chat app. *Digital investigation* 25, 5–23.
- Sylve, Joe, 2012. LiME-Linux memory extractor. In: *Proceedings of the 7th ShmooCon Conference*. <https://github.com/504ensicsLabs/LiME>. (Accessed 17 April 2023).
- Sylve, Joe, Case, Andrew, Marziale, Lodovico, Richard, Golden G., 2012. Acquisition and analysis of volatile memory from Android devices. *Digital Investigation* 8 (3–4), 175–184.
- Thompson, Warren, 2022. A forensic analysis of Android mobile private browsing artifacts. White paper. SANS Institute. (Accessed 17 April 2023).
- Thorbecke, Catherine, 2019. Facebook agrees to pay UK fine over Cambridge Analytica scandal while admitting no liability. <https://abcnews.go.com/Business/facebook-agrees-pay-uk-fine-cambridge-analytica-scandal/story?id=66635145>. (Accessed 17 April 2023).
- Tri, Muchamad Kukuh, Riadi, Imam, Prayudi, Yudi, 2018. Forensics acquisition and analysis method of IMO messenger. *International Journal of Computer Applications* 179 (47), 9–14.
- Umar, Rusydi, Riadi, Imam, Muthohirin, Bashor Fauzan, 2019. Live forensics of tools on Android devices for email forensics. *TELKOMNIKA (Telecommunication Computing Electronics and Control)* 17 (4), 1803–1809.
- Vidas, Timothy, Zhang, Chengye, Christin, Nicolas, 2011. Toward a general collection methodology for Android devices. *Digital Investigation* 8, S14–S24.
- Voigt, Paul, Von dem Bussche, Axel, 2017. *The EU General Data Protection Regulation (GDPR). A Practical Guide*, vol. 10(3152676), 1st ed. Springer International Publishing, Cham, pp. 10–5555.
- Volatility, 2016. Android Volatility Framework. <https://github.com/volatilityfoundation/volatility/wiki/Android>. (Accessed 17 April 2023).
- Wang, Jules, 2022. Google explains the Pixel 7 is 64-bit only, says 32-bit apps still have roles to play. <https://www.androidpolice.com/google-pixel-7-pro-64-bit-info>. (Accessed 17 April 2023).
- Wijesekera, Primal, Baokar, Arjun, Tsai, Lynn, Reardon, Joel, Egelman, Serge, Wagner, David, Beznosov, Konstantin, 2017. The feasibility of dynamically granted permissions: aligning mobile privacy with user preferences. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 1077–1093.
- Wu, Songyang, Zhang, Yong, Wang, Xupeng, Xiong, Xiong, Du, Lin, 2017. Forensic analysis of WeChat on Android smartphones. *Digital investigation* 21, 3–10.
- Wu, Songyang, Sun, Wenqi, Liu, Xin, Zhang, Yong, 2018. Forensics on Twitter and WeChat using a customised Android emulator. In: *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*. IEEE, pp. 602–608.
- wxHexEditor, 2017. wxHexEditor. <https://www.wxhexeditor.org>. (Accessed 17 April 2023).
- Yang, Haiyu, Zhuge, Jianwei, Liu, Huiming, Liu, Wei, 2016. A tool for volatile memory acquisition from Android devices. In: *IFIP International Conference on Digital Forensics*. Springer, pp. 365–378.
- Yang, Seung Jei, Choi, Jung Ho, Kim, Ki Bom, Bhatia, Rohit, Saltaformaggio, Brendan, Xu, Dongyan, 2017. Live acquisition of main memory data from Android smartphones and smartwatches. *Digital Investigation* 23, 50–62.
- Younis, Lojin Bani, Sweda, Safa, Alzu'bi, Ahmad, 2021. Forensics analysis of private web browsing using Android memory acquisition. In: *2021 12th International Conference on Information and Communication Systems (ICICS)*. IEEE, pp. 273–278.
- Zhou, Fan, Yang, Yitao, Ding, Zhaokun, Sun, Guozi, 2015. Dump and analysis of Android volatile memory on Wechat. In: *2015 IEEE International Conference on Communications (ICC)*. IEEE, pp. 7151–7156.