Computers & Security 115 (2022) 102626

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TC 11 Briefing Papers

Digital forensic analysis methodology for private browsing: Firefox and Chrome on Linux as a case study☆

Xosé Fernández-Fuentes[a,*], Tomás F. Pena[a,b], José C. Cabaleiro[a,b]

[a] Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Rúa de Jenaro de la Fuente Domínguez, 15782 Santiago de Compostela, Spain
[b] Departamento de Electrónica e Computación, Universidade de Santiago de Compostela, Rúa Lope Gómez de Marzoa, 15782 Santiago de Compostela, Spain

ARTICLE INFO

Article history:
Received 26 April 2021
Revised 23 December 2021
Accepted 22 January 2022
Available online 24 January 2022

Keywords:
Digital Forensics
Browsing artefacts
Private browsing
Internet privacy
Virtualization

ABSTRACT

The web browser has become one of the basic tools of everyday life. A tool that is increasingly used to manage personal information. This has led to the introduction of new privacy options by the browsers, including private mode. In this paper, a methodology to explore the effectiveness of the private mode included in most browsers is proposed. A browsing session was designed and conducted in Mozilla Firefox and Google Chrome running on four different Linux environments. After analyzing the information written to disk and the information available in memory, it can be observed that Firefox and Chrome did not store any browsing-related information on the hard disk. However, memory analysis reveals that a large amount of information could be retrieved in some of the environments tested. For example, for the case where the browsers were executed in a VMware virtual machine, it was possible to retrieve most of the actions performed, from the keywords entered in a search field to the username and password entered to log in to a website, even after restarting the computer. In contrast, when Firefox was run on a slightly hardened non-virtualized Linux, it was not possible to retrieve any browsing-related artifacts after the browser was closed.

1. Introduction

In recent years, there have been several events that have raised a lot of attention on the importance of privacy. Revelations such as Edward Snowden's (Greenwald, 2013a; 2013b), the Cambridge Analytica scandal[1], or the big Equifax data breach (Berghel, 2017) have given a big push to make the world aware of the level to which personal data collection reaches. Personal data are collected, shared, and sold by a large number of online services. Services that, most of the time, do not need to handle such a large amount of personal data to perform their function (Felt and Evans, 2008; Krasnova et al., 2013). However, they continue to collect them because it is very easy and very cheap to store them with the excuse that they could be useful in the future. Not only that, but they do not invest enough in adequately protecting the information, which leads to leaks. Leaks where the biggest victims are the users, whose personal data are exposed to anyone with access to the Internet.

In order to ensure greater control over the data collected by companies, the European Parliament introduced the General Data Protection Regulation (GDPR) in 2016. GDPR restricts and determines how personal data should be managed. For example, companies must now make users aware of what information they are collecting, they must delete the data once they are no longer needed, they must not collect more information than is strictly necessary, and they must ensure that the data are protected with appropriate security measures (Anderson and von Seck, 2020). Since the introduction of the GDRP, companies have reduced their use of cookies, web users have noticed an increase in the number of consents, and websites show users what information they collect and for what purpose (Anderson and von Seck, 2020; Kretschmer et al., 2021). New studies have also been published that aim to help companies adapt to the new regulation. For example,

☆ This work has received financial support from the Ministerio de Ciencia e Innovación, Spain within the project PID2019-104834GB-I00. It was also funded by the Consellería de Cultura, Educación e Ordenación Universitaria of Xunta de Galicia (accr. 2019–2022, ED431G 2019/04 and reference competitive group 2019–2021, ED431C 2018/19) and the European Regional Development Fund (ERDF). 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).

[1] https://abcnews.go.com/Business/facebook-agrees-pay-uk-fine-cambridge-analytica-scandal/story?id=66635145

https://doi.org/10.1016/j.cose.2022.102626

in Caruccio et al. (2020) a methodology is presented to detect possible privacy violations when performing operations with large amounts of data, such as data integration or record linking.

To help users realize how their personal data are shared between different web services, there are tools that graphically show how information flows in the network during a browsing session. For example, CHRAVAT (Cirillo et al., 2019) or VIPAT (Breve et al., 2020), which allow to obtain real–time graphs of the different providers with which the user interacts while surfing the web.

The more aware users are of the importance of privacy, the more they will demand tools that help them preserve their privacy. One example is the incorporation of the private mode in most current browsers. This new browsing mode is designed to prevent any information related to browsing from being stored on the device being used.

Different papers can be found in the literature exploring the effectiveness of the private mode of different browsers. However, in these papers, browsers are often tested in a superficial and unstructured way, making it difficult to verify the correct functioning of the private mode in different environments or to compare the level of privacy offered by different browsers. Therefore, the objective of this paper is to describe a consistent and thorough methodology for testing the private mode of Internet browsers. Furthermore, this methodology is completely independent of the browser and the operating system used.

Once the methodology has been presented, it will be applied, as an example, to Mozilla Firefox and Google Chrome running on four different Linux-based environments. In the first scenario, the browser will be run on a bare-metal computer with Ubuntu 20.04. In the second scenario, it will be executed on a slightly hardened Ubuntu. In the third scenario, it will be launched on a Ubuntu virtual machine, with Ubuntu as the host system and VirtualBox as the hypervisor. Finally, the fourth scenario is the same as the third one, but using VMware as hypervisor.

The reason for testing browsers in different environments is to verify that the private mode continues to work as it should, even when used in different or less common situations. As an operating system, it was decided to use a Linux distribution because, to the best of our knowledge, there is no work focused on a Linux-based operating system, apart from Anuradha et al. (2016) where the behavior of Chrome running on Ubuntu is tested but without studying the behavior of the private mode nor analyzing the RAM content.

The paper is structured as follows: Section 2 summarizes previous work related to forensic analysis of different browsers; Section 3 describes the proposed methodology; Section 4 shows the application of the methodology to Firefox and Chrome; the results obtained and their discussion are covered in Sections 5 and 6, respectively; finally, Section 7 contains the closing thoughts.

2. Related work

In the literature, there are previous works that study possible information leaks from the private mode included in the browsers. This section discusses the most relevant studies of the last few years ordered from oldest to newest.

Calum Findlay and Petra Leimich (Findlay and Leimich, 2014) studied the behavior of Firefox in four different situations: normal and private mode with Firefox installed on the system and normal and private mode using the portable version of Firefox. Their goal was to establish which conditions reduced the amount of data filtered, thus maximizing the user's privacy. They observed that no data were written to permanent storage during private browsing sessions, neither with the installed version nor with the portable version. The only exception where there was a possibility of writing to disk was if the operating system decided to move a memory page containing sensitive information to the swap memory. As good practice, they recommend restarting the computer after finishing a browsing session in order for the RAM to be restored.

Reza Montasari and Pekka Peltola (Montasari and Peltola, 2015) studied what information could be extracted if a forensic analysis were to be performed on a computer after browsing in private mode. For this purpose, they tested the following browsers: Chrome 26, Firefox 20, Internet Explorer 9, and Safari 5. All of them were run on a Windows operating system running on VirtualBox virtual machines. Various activities were executed with each of the browsers such as playing a video on YouTube, searching for a product on Amazon, or previewing a PDF file. To perform the analysis, they decided to dump the RAM just before closing the browser and to create a disk image just after closing the browser. The tools used during the analysis were FTK Imager, Autopsy, FTK, and WinHex. The results of analyzing the disk images revealed that Chrome was the only one that did not leave any artifacts on disk. However, the RAM analysis did reveal nearly all activities performed in private mode, regardless of the browser used.

Anuradha et al. (2016) studied what information could be recovered from a disk image after deleting browsing artefacts. The browser chosen for testing was Chrome running on Ubuntu 14.04. The browsing session consisted of watching videos in YouTube, searching for images in Google Images, searching for items in Amazon, and accessing Gmail. Once the navigation session had been ended using Chrome in normal mode (not incognito), they manually deleted all browsing artefacts. Then, they created an image of the hard drive used and performed a series of searches on the disk image with the AccessData Forensic Tool Kit. They were able to recover much of the information generated, including some of the images displayed. However, they were unable to recover the passwords or the videos played.

Nikolaos Tsalis et al. (Tsalis et al., 2017) studied the private mode of Chrome 47, Firefox 43, Internet Explorer 11, and Opera 34 running on a Windows 7 virtual machine. They performed the analysis from the point of view of an attacker who had temporary physical access to the computer after a user had browsed using the private mode and left the computer on. After conducting the experiments, they discovered situations where there were privacy violations that should not have occurred in private mode according to browser documentation. In one of the tests, they found that saving a bookmark in Firefox or in Chrome stored additional information indicating that it had been created using private mode. In another test, they discovered that Firefox stored OCSP protocol responses in the browser's cache folder, leaking the websites that had been accessed. As a solution to avoid this type of leaks, they suggest storing the browser profile in a virtual file system hosted on a volatile medium (such as RAM).

Graeme Horsman (Horsman, 2017) performed a process–level analysis of Chrome during a private browsing session. The Chrome version analyzed was 55 running on Windows 7. To show the interaction with the operating system, he used the different tools available in the Windows Sysinternals suite. In particular, one of the experiments carried out was to compare the number of system events generated by a normal session with an incognito session. The result was a significant decrease of events in the private session. After the analysis, and despite not having been able to verify the content of some of the temporary files created by Chrome, the author concludes that the best way to recover browsing actions is by analyzing the RAM.

There are also previous works that explore the private mode of less popular browsers. For example, Szu-Yuan Teng et al. (Teng and Wen, 2018) tested six different browsers with incognito functions running on a virtualized Windows 10: Epic Privacy Browser, Secure Browser, Comodo Dragon, SRWare Iron, Dooble, and Maxthon. Specifically, they analyzed the network traffic generated when each

browser was opened as well as retrieved the username and password used to log into a website from memory. They conclude that the private mode poses a challenge for forensic analysis, being only possible to recover valuable information when accessing the contents of the memory allocated to the browser.

Abid Khan Jadoon et al. (Jadoon et al., 2019) designed and implemented a series of scenarios to test the privacy and anonymity offered by Tor Browser. All tests were conducted on Windows 8.1 running on VMware Workstation. Data acquisition was performed at three different points in time: firstly, with the browser open; secondly, after the browsing activities have been performed; and, finally, after closing the browser. They conducted a comprehensive analysis of the registry, RAM, and hard drive, which led them to the conclusion that Tor Browser leaves a large number of artefacts, especially in memory, which allows many of the activities performed online to be discovered.

In a more recent study, Graeme Horsman et al. (Horsman et al., 2019) tested the private browsing mode of 30 different browsers. Each of them was deployed on a standalone Windows 10 virtual machine on VirtualBox. The test consisted of visiting five URLs with each of them and then using a search term for each URL in order to determine if the browser had written any browsing-related data to disk. The results show that only 5 of the 22 browsers that offered private mode leaked browsing session information to disk.

Rebecca Nelson et al. (Nelson et al., 2020) performed a thorough analysis of the following browsers: Firefox 55, Chrome 61, and Tor Browser 7. The main forensic analysis tool used was FTK and all browsers were tested on Windows 7 virtual machines. The results show that it was possible to retrieve practically all the information related to the browsing performed when using the browsers in normal mode, how the amount of retrievable information is drastically reduced when using the private mode, and how it was not possible to recover almost anything when using Tor Browser. The entire analysis is focused exclusively on the information that can be recovered from a hard disk image.

Comparing the present manuscript with these previous works, its main contributions can be summarized in the following points:

1. RAM is captured in more situations, not just when the browser is running or just after closing it, allowing a more complete picture of the information that can be retrieved at different times.
2. No previous work, to the best of our knowledge, has studied whether the password keychain of a browser can be recovered from a memory dump. In this work, it is shown that, in some situations, it is possible to recover the complete browser keyring.
3. Both, bare metal and virtual machines have been tested in this work, whereas all previous works mentioned in this section use virtualized or non-virtualized environments, but not both. In addition, some of the kernel hardening options were also tested to see how they affect the behavior of the browsers.
4. A browsing session has been designed that includes all the activities used in previous works and some additional ones not considered so far. The deployment, execution and test capture phases were designed and executed to cover a wide number of scenarios, which has made it possible to thoroughly study the behavior of private browsing.

3. Methodology

The methodology introduced in this paper aims to test the private mode included in most current browsers. With this methodology it is possible to determine the level of effectiveness of this privacy feature, as it will reveal what information can be retrieved after browsing in private mode.

The proposed methodology is divided into five phases: environment setup, monitoring changes, browsing, data acquisition, and analysis.

3.1. Environment setup

The objective of this phase is to design and deploy the environment (or environments) where the selected browser will be tested. It is essential to consider using more than one environment, as it allows to get a broader picture of how the browser behaves in diverse situations. In this case, when using multiple environments, it is important to take into account the following characteristics:

- Different operating systems or low-level options of the same operating system.
- A bare metal environment and a virtualized environment.

The inclusion of a bare metal environment is of vital importance since it should not be assumed that the information that can be retrieved from a virtualized environment is the same as from a bare metal environment. For example, memory management in the virtualized environment is different due to the addition of the hypervisor layer.

When setting up a computer for testing, the following suggestions should be considered:

- Use a dedicated computer with a freshly installed operating system. Thus, any artefacts found must have been created by the browser. It is also advisable to use another computer for all subsequent analysis and processing tasks, avoiding contamination of the test computer.
- Disable automatic updates. To prevent the browser version from changing between executions of the same experiment, it is necessary to disable automatic updates. To do this, there are two options: 1) Configure the system not to update the web browser but to update the rest of the packages or 2) Configure the system not to perform any updates. The problem with the first option is that more variables are introduced when performing the experiments. For example, if repeating a test yields a different result, it will be more difficult to determine whether the problem is in the test itself or in the change of version of a particular package. Therefore, it is recommended to configure the second option. Another benefit of disabling automatic updates is that the number of background processes is reduced, allowing better isolation of the browser's behavior.

3.1.1. Browser setup

Once the environment (or environments) to be used has been decided, it is time to prepare the selected browser. The only preparation necessary to apply this methodology is to add a username and password to the browser's keychain. However, the specific browser configuration will depend on what is to be tested. For example, this methodology could be used to test whether it is worth installing an extension that deletes the cookies created by a website after leaving it. In this case, two profiles would be required: one with the extension installed and one without. When the results were analyzed, it would be checked if the extension did its job correctly and, therefore, if it is worth installing.

3.2. Monitoring changes

In this phase, it must be established how the changes made to the file system by the browser will be monitored and how the system RAM will be dumped. The tools used will vary depending on the operating systems selected in the environment setup phase.

To monitor changes made to the file system, a tool that logs each time a file or folder is created, modified or deleted in the

monitored directories should be selected. The recommended option when specifying the directories to monitor is to include only the folders where the browser profile is stored as well as temporary folders. The other option would be to monitor all changes made to the entire file system during the execution of the tests. However, this option makes subsequent analysis much more complicated.

To obtain a dump of the RAM content, there are tools that allow to dump the memory space of a particular process. However, this type of tool is not valid for the purpose of this methodology for two reasons:

1. Dumping only the memory currently allocated to the browser process would not dump memory areas that were previously allocated to the browser, but were freed.
2. It would not be possible to dump the memory associated with the browser process after closing the browser or after restarting the computer, because the browser process would no longer exist and, therefore, it would not be possible to dump its associated memory.

Due to the above reasons, a tool that allows to obtain a complete dump of the computer's RAM must be selected. Both specialized hardware and software solutions are available (Kollár, 2010).

3.3. Browsing

In this phase, a browsing session should be designed to generate data for subsequent analysis. Having this session pre-planned has two main advantages: 1) it is easily reproducible and 2) allows for a more targeted analysis in the next stage. This session should incorporate the most common activities performed when using a web browser. As mentioned in Montasari and Peltola (2015), some of the most common actions are: downloading a file, performing a search, watching a streaming video, or viewing a PDF document. Therefore, based on the methodologies described in Montasari and Peltola (2015); Muir et al. (2019), an outline for a browsing session that includes a variant of these actions as well as new actions has been created. As new activities, the following can be highlighted: logging into a website, entering a URL but not accessing it, or using login information stored in the browser's keychain.

The outline for the browsing session is as follows:

1. Access a web page that hosts multimedia content, such as music or videos. Use the search engine and play one of the items returned by the search. Motivation: Is it possible to retrieve the words entered in the search field and the name of the item played?
2. Open a new tab and access a web page that stores a cookie in the browser. Motivation: Is it possible to retrieve the cookie created by the website?
3. Open a new tab and access a website that hosts a PDF file. Preview the content of the PDF file with the browser. Motivation: Is it possible to recover the name and the content of the file?
4. Open a new tab and enter a URL in the address bar. Without accessing that website, delete the written URL. It is important not to use a URL that can be easily found in memory or on disk. For example, using a URL such as bing.com should be avoided, as it will produce many false positives due to the fact that it is one of the built-in search engines in the majority of browsers. Therefore, it is advisable to make sure that there is no match when searching for that URL before performing the tests. Motivation: Is it possible to retrieve the URL entered?
5. Open a new tab and access the website for which the login information is stored in the browser's keychain. Attempt to log in using the saved credentials. Motivation: Is it possible to retrieve the full database of logins and passwords from memory?
6. Open a new tab and access a different website from the previous one where there is also a login page. Try to log in by entering the information requested by the page (username, password...). Motivation: Is it possible to retrieve the information entered?

3.4. Data acquisition

The purpose of this phase is to gather the necessary data from the test machine for further analysis. Specifically, the aim is to obtain the list of changes made to the hard disk as well as a complete dump of the RAM. In order to make the test runs as "clean" as possible, the capture of the changes on disk has to be done in a completely independent run from when the RAM memory contents are dumped. In addition, only one browser can be tested at a time. The following subsections describe in detail how and when the different acquisitions should be made.

3.4.1. Hard disk

In this case, the steps to be taken are very simple:

1. Launch the monitoring tool chosen in Section 3.2.
2. Perform the navigation session designed according to the scheme described in Section 3.3.
3. Close the browser.
4. Stop the monitoring tool.

3.4.2. Memory

In the case of RAM, the process is more complex, as it varies depending on whether the browser is running on a virtual machine or directly on the machine. Generically, the memory should be dumped at four different times:

T1. With the browser running and after having completed the designed browsing session.
T2. After closing the browser.
T3. After rebooting the computer.
T4. After the computer has been turned off for 10 seconds.

As in the case of the acquisition of disk changes, the runs have to be completely separate. That is, in one run the memory is dumped in only one of the moments described above. In this way, the "cleanest" possible memory dumps are obtained.

As mentioned above, the RAM dump process depends on where the browser is being executed. When it is running directly on the machine, the steps to be performed to create the different dumps are as follows:

T1. Turn on the computer, launch the browser in private mode, perform the browsing session, and dump the RAM.
T2. Turn on the computer, launch the browser in private mode, perform the browsing session, *close the browser, wait one minute*, and dump the RAM.
T3. Turn on the computer, launch the browser in private mode, perform the browsing session, close the browser, *restart the computer, and, once started*, dump the RAM.
T4. Turn on the computer, launch the browser in private mode, perform the browsing session, close the browser, *turn off the computer, wait 10 seconds, turn on the computer, and, once started*, dump the RAM.

Fig. 1 shows an overview of the different actions to be done to obtain each of the dumps.

When the browser is running in a virtual machine, the procedure is slightly different. In this case, the steps to be performed to create the various dumps are:

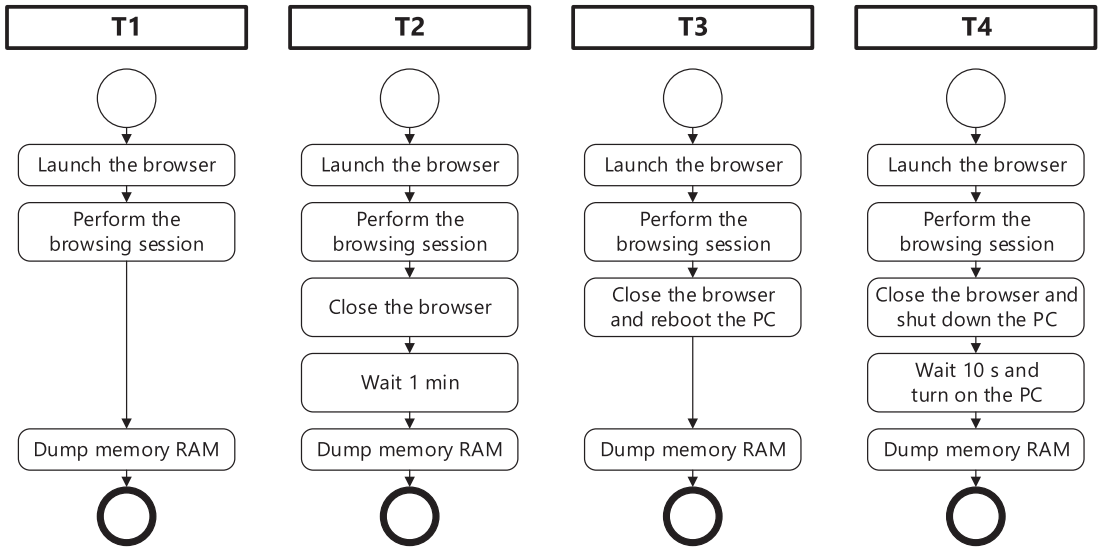

Fig. 1. Diagram of the steps to be performed to obtain the different memory dumps when the browser is running directly on the machine.

T1. Turn on the computer, start the virtual machine, launch the browser in private mode, perform the browsing session, and dump the RAM.
T2. Turn on the computer, start the virtual machine, launch the browser in private mode, perform the browsing session, *close the browser, turn off the virtual machine*, and dump the RAM.
T3. Turn on the computer, start the virtual machine, launch the browser in private mode, perform the browsing session, close the browser, turn off the virtual machine, *restart the computer, and, once started*, dump the RAM.
T4. Turn on the computer, start the virtual machine, launch the browser in private mode, perform the browsing session, close the browser, turn off the virtual machine, *turn off the computer, wait 10 seconds, turn on the computer, and, once started*, dump the RAM.

Fig. 2 shows a diagram of the steps to be followed with this setup.

It is important to mention that there are some BIOSes that wipe the contents of the RAM memory on reboot (Kollár, 2010), preventing potentially interesting information from being retrieved after restarting the computer.

3.5. Analysis

In this stage the aim is to retrieve as much information as possible from the browsing session. On the one hand, the list of files obtained by the tool that monitors changes on the hard disk has to be parsed. Each of these files has to be examined to check whether it is possible to obtain information about any of the activities performed. On the other hand, the different memory dumps have to be analyzed, trying to recover as much information as possible from each one of them.

To parse the list of files obtained, it is possible to develop scripts that perform keyword searches related to the browsing performed. In addition, when the browser is executed in a virtual

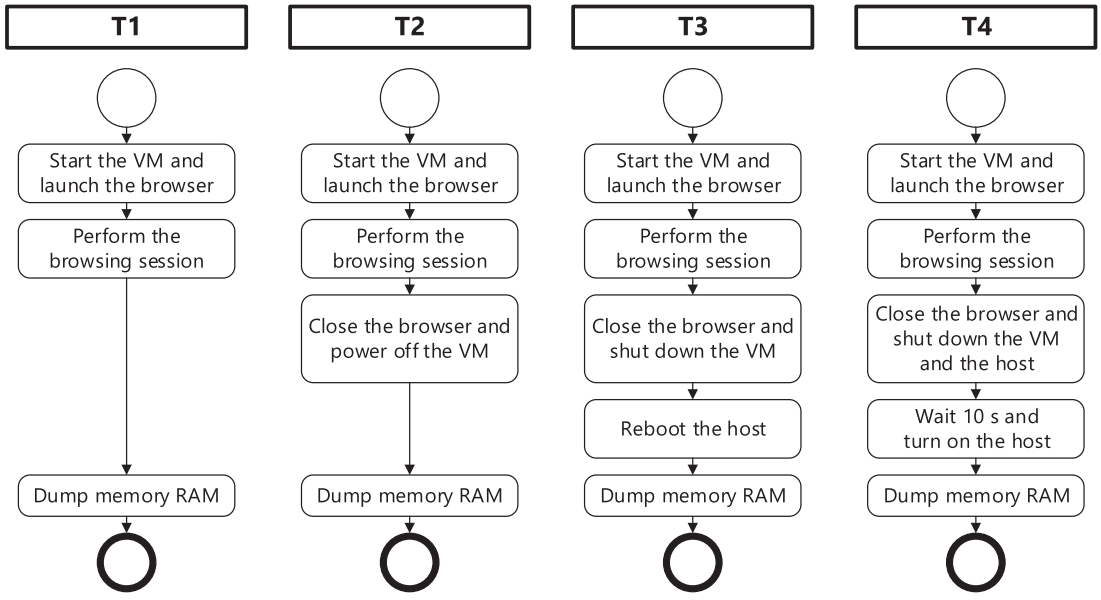

Fig. 2. Diagram of the steps to be performed to obtain the different memory dumps when the browser is running on a virtual machine.

machine, it should also be searched directly on the virtual disks. This ensures that the browser has not written to one of the directories that was not being monitored. In the case of memory dumps, the use of the following two cross-platform tools is recommended: Volatility Framework[2] and wxHexEditor[3]. In addition, for Volatility Framework there is the Actaeon[4] Graziano et al. (2013) plugin that facilitates the analysis of memory dumps containing running virtual machines. This plugin can be very useful when analyzing a memory dump where the browser is running in a virtual machine. As in the case of the files, scripts can be written to automate the search for information related to the navigation.

For the particular case of retrieving the complete content of the keychain, the tools to be used vary depending on the browser. However, there are cross-platform tools, such as HackBrowserData[5], that allow saved passwords, history, cookies, and bookmarks from different browsers to be recovered.

4. Firefox and Chrome as use cases

This section is dedicated to show the result of applying the methodology described in this work to two different browsers. Specifically, the browsers tested were Firefox and Chrome running on a Linux operating system. As mentioned in the introduction, the reason for focusing on Linux is because, although there are previous works that study the private mode of browsers, there is none focused on that particular operating system to the best of our knowledge. The only work that uses Linux is Anuradha et al. (2016), but it does not study the behavior of private mode or analyze the contents of RAM.

4.1. Environment setup

All experiments were tested in four different environments, which will be referred to as environment A, environment B, environment C, and environment D. They were deployed on a PC with a Core i7-4700K processor and 8 GB of DDR3-1600 MHz RAM. As recommended in Section 3.1, this equipment was used exclusively for the execution of the tests. All subsequent analysis and processing tasks were performed on a different machine.

In environment A, the browser was executed directly on the machine. The operating system chosen was Ubuntu 20.04 running the 5.4.0-26-generic version of the kernel. A clean install was performed and only three changes were made to the system:

1. Automatic updates were disabled.
2. Firefox was updated to version 95.0.
3. Chrome version 96.0.4664.110 was installed.

The versions of both browsers were the most recent at the time of testing.

Environment B is almost the same as environment A. The difference is that two boot options were added, `init_on_alloc=1` and `init_on_free=1`. As can be read in the commit description where these options were introduced[6], `init_on_alloc=1` zeroes new memory pages as well as heap objects, while `init_on_free=1` zeroes pages that have been freed as well as heap objects that have been deleted. The purpose of this environment was to test the effectiveness of these options and determine if they provide any improvement in terms of privacy.

In environment C, the browser was run in a VirtualBox virtual machine. For this, starting from environment A, VirtualBox 6.1.30 was installed and a virtual machine was created with 4 GB of RAM allocated. It ran the same operating system with the same changes as those mentioned for environment A.

In environment D, the browser was also run in a virtual machine but, in this case, using VMware as hypervisor. As in the previous case, starting from environment A, VMware Workstation Pro16.2.1 was installed and a virtual machine with 4 GB of RAM allocated was created. The operating system executed in the virtual machine and the modifications made were the same as in the environment C.

The purpose of the latter two environments was to determine whether, from a privacy standpoint, there was any advantage in isolating the browser in a virtual machine.

4.1.1. Browser setup

The browser configuration used was the default one. The only preparation done was to add a username and password to the browser's keychain. To do this, the https://mail.protonmail.com/login website was accessed and `test` was entered as the username and `1234` as the password. Once typed, this information was added to the keychain. This action was not done using private mode because, in the case of Chrome, it is not possible to add entries to the keychain from incognito mode. Once the login information was saved, the history, cache, and cookies were cleared.

4.2. Monitoring changes

This section describes the tools used to log the changes made by the browser. All the data obtained in this phase will be presented in the next section, where it will be checked what traces the browsers left behind.

4.2.1. Monitoring changes in the hard drive

To verify that the browser was not writing to disk any data that would reveal the activity carried out in the private mode, it was necessary to monitor the changes made to the file system. For this purpose, the `inotifywait` tool was used. It is part of the `inotify-tools`[7] package, which groups together a set of programs to monitor file system events.

This tool allows the capture a variety of events. For this particular use case, the capture was limited to the following ones:

- `create`. A file or directory was created within a watched directory.
- `modify`. A watched file or a file within a watched directory was written to.
- `delete`. A file or directory within a watched directory was deleted.

The mandatory parameter when configuring the tool is to specify the files or folders to be watched. In the case of Firefox, the directories monitored were:

- `~/.mozilla/`
- `~/.cache/mozilla/`

These are the two directories inside the home folder where Firefox writes data. These paths, which can be obtained by accessing the page with the address `about:profiles` in Firefox, are called "Root Directory" and "Local Directory", respectively.

In the case of Chrome, the directories monitored were:

- `~/.config/google-chrome/`
- `~/.cache/google-chrome/`

[2] https://github.com/volatilityfoundation/volatility.
[3] https://github.com/EUA/wxHexEditor.
[4] https://www.s3.eurecom.fr/tools/actaeon.
[5] https://github.com/moonD4rk/HackBrowserData.
[6] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=6471384af2a6530696fc0203bafe4de41a23c9ef.
[7] https://github.com/inotify-tools/inotify-tools.

The user data directory can be obtained by accessing `chrome://version` and looking for the "Profile Path" field. The user data directory is the parent of that path.

4.2.2. Memory dump

To dump the memory completely, LiME[8] (Sylve, 2012) (commit ID fa37b69) was used. LiME is a Loadable Kernel Module that, when it is loaded, dumps the memory content in the file that is passed to it as an argument. This file can be used later, for example, with the Volatility Framework tool. Another tool whose purpose is practically the same as LiME is AVML[9]. The main difference of AVML with respect to LiME is that it is not necessary to know, a priori, the Linux distribution or the kernel version on which it will be launched.

In all four environments, the entire memory of the host was always captured. In other words, the LiME kernel module was always loaded on the host.

4.3. Browsing

Using the scheme in Section 3.3 as a basis, the specific sequence of steps forming the browsing session used is as follows:

1. Go to 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`. After 15 seconds, pause the video.
2. Open a new tab and go to https://stackoverflow.com.
3. Open a new tab and go to https://meltdownattack.com. Click on `Spectre Paper`. The browser will display the contents of the `spectre.pdf` file.
4. Open a new tab and write myurl.com in the address bar. Without accessing this website, delete the written URL.
5. Access https://mail.protonmail.com/login. Attempt to log in using the credentials stored in the browser's keychain. The login will fail.
6. Open a new tab and go to 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.

4.4. Data acquisition

The acquisition phase was performed following the steps described in Section 3.4. In the particular case of memory, to dump the RAM content in environments A and B, the steps described for the situation in which the browser runs directly on the machine were followed and, for environments C and D, the steps described for the situation in which the browser runs in a virtual machine were followed.

4.5. Analysis

As described in Section 3.5, this phase attempts to retrieve as much information as possible from both the hard disk and memory dumps.

In the case of the hard disk, to parse the list of files provided by `inotifywait`, several scripts were developed to perform keyword searches related to the browsing performed. In addition, in the instance of environments C and D, the virtual disks were also scanned to ensure that the browser had not written to one of the directories that was not being monitored. In the case of memory dumps, the following two tools were used: Volatility Framework (commit ID 703b29b) and wxHexEditor 0.24. As in the case of the files, several scripts were created to automate the search for information related to the navigation.

As for the tools used to retrieve the contents of the keychains, Firefox Decrypt[10] (commit ID 557bb60) tool was used in the case of Firefox and, in the case of Chrome, a custom script based on the Chrome-Password-Grabber[11] script was developed. For this specific case, different tools were used for each browser. However, as mentioned in Section 3.5, there are tools that support multiple browsers and are cross-platform.

5. Results

Each run, whether to monitor disk changes or to obtain a memory dump, was repeated several times in order to verify the results. Between each execution, the following precautions were taken to avoid contaminating the results:

- The contents of the `~/.mozilla` and `~/.config/google-chrome` folders were restored with a clean profile. As mentioned in Section 4.1.1, the only change made to the profiles was to add a username and password to the browser's keychain.
- The contents of the `~/.cache/mozilla/` and `~/.cache/google-chrome/` folders were deleted.
- In the case of environments C and D, a clean snapshot of the virtual machine was restored.
- The computer was turned off and the power cord was unplugged from the wall for a period of at least 1 minute. The goal was to try to start each execution with a "clean" RAM. As described in Gruhn and Müller (2013), the number of correct bits that can be physically recovered from DDR3 memory at room temperature is less than 50% after only 10 seconds. They also point out that with this type of memory, the only information recoverable after a cold reboot is noise patterns. More recent studies (Bauer et al., 2016; Yitbarek et al., 2017) show that it is possible to descramble the contents of DDR3 and DDR4 DRAM by performing a cold boot attack.

The findings in each of the scenarios described above are shown in detail in the following subsections.

5.1. Findings from hard disk analysis

While the browsing session was being conducted, changes made to the file system were monitored as described in Section 4.2.1. After analyzing the files indicated by the monitoring tool, and the virtual disks used in the environments C and D, no information associated with browsing was found in any of the files.

5.2. Findings on the memory dumps

This subsection shows the information that was possible to retrieve from the different memory dumps.

5.2.1. Environment A

The first part of the analysis consisted of loading the files containing the memory dumps into the hexadecimal editor mentioned in Section 4.5. Once opened, a series of searches were executed to check whether or not those keywords existed in memory, as well as to obtain the number of occurrences of those keywords.

The set of keywords searched in the memory dump was:

[8] https://github.com/504ensicsLabs/LiME.
[9] https://github.com/microsoft/avml.
[10] https://github.com/unode/firefox_decrypt.
[11] https://github.com/priyankchheda/chrome_password_grabber.

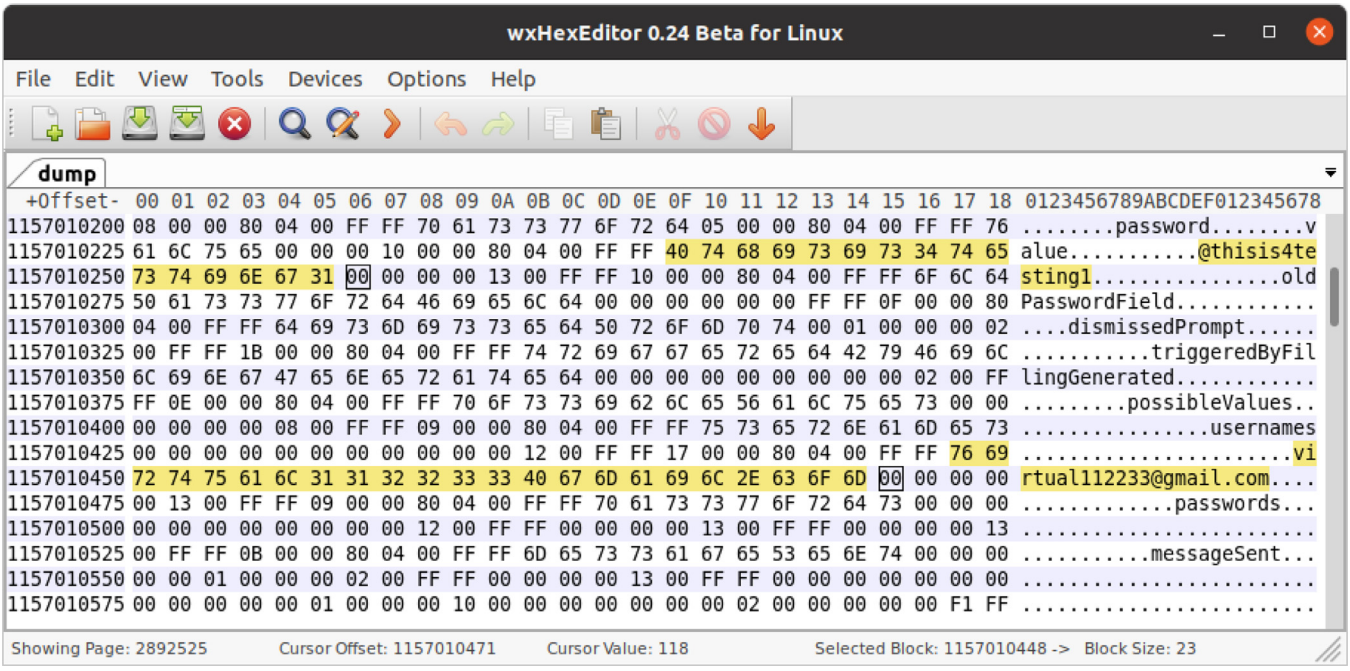

Fig. 3. Password and username entered on the Gmail login page found in the memory dump.

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. `prov=`
7. `spectre.pdf`
8. `myurl.com`
9. `virtual112233@gmail.com`
10. `virtual112233%40gmail.com`
11. `@thisis4testing1`
12. `%40thisis4testing1`
13. The Google Safe Storage key

The first four items are searches related to the words entered in the YouTube search field. Number 2 is the same as number 1 except that the space has been replaced by its HTML code (%20). The same was done in number 4, where the + has been replaced by its hexadecimal code (%2B). Item 5 is the full name of the YouTube video played. Number 6 corresponds to the cookie set by the https://stackoverflow.com site. Item 7 is the name of the PDF file previewed in the browser. Number 8 corresponds to the URL written in the address bar that was not accessed. The following four items are the username and password entered on the Gmail login page. As in the first items, the @ has been replaced by its HTML code (%40) in items 10 and 12. Finally, if the browser used was Chrome, the Google Safe Storage key would be searched for. This key is used to encrypt, among other things, the contents of the keychain. As an example, Fig. 3 shows the result of performing searches number 9 and 11 in the hexadecimal editor.

The next part consisted of checking if it was possible to recover certain files from memory. First, the `spectre.pdf` file that was previewed in the browser was searched for. Then, if the browser used was Firefox, the following three files would be searched for: `cert9.db`, `key4.db`, and `logins.json`. On the contrary, if the browser used was Chrome, the following file would be searched for: `Login Data`. All these files are the ones that store the keychain. To check if they existed in memory, a script was developed to search for the complete content of these files and indicate if it was possible to retrieve them in their entirety. Another option to perform this part of the analysis would be to use the Volatility Framework `linux_find_file` plugin to retrieve the aforementioned files.

The result of performing the above procedure on the four memory dumps obtained with environment A can be seen in Table 1. This table shows the number of occurrences of each keyword as well as whether or not it was possible to recover the files mentioned above.

5.2.2. Environment B

The procedure performed in this environment was identical to the one executed in environment A. Table 2 summarized all the information that was possible to recover after analyzing the dumps obtained with this environment.

5.2.3. Environment C

As in environment B, only the summary table (Table 3) with the findings is included.

5.2.4. Environment D

As in environments B and C, only the summary table (Table 4) with the findings is included.

6. Discussion

The following subsections discuss the results obtained after analyzing the RAM and hard disk in the different configurations.

6.1. Hard disk

The purpose of this work was to find out how much information was possible to recover after having used the private mode. The results show that both Firefox and Chrome kept their word that no data associated with browsing in private mode are written to disk.

When using the browser's built-in keychains, it was noted that the behavior was slightly different between the two browsers. In

Table 1
Summary of the analysis of memory dumps performed in environment A. 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 or not it was possible to retrieve those files.

	Firefox				Chrome			
	T1	T2	T3	T4	T1	T2	T3	T4
Keyword searches								
kernel bugs	13	0	0	0	48	0	0	0
kernel%20bugs	18	0	0	0	14	0	0	0
kernel+bugs	40	0	0	0	58	0	0	0
kernel%2Bbugs	34	0	0	0	25	6	0	0
Syzbot and the Tale of...	17	0	0	0	38	3	0	0
prov=	2	0	0	0	1	0	0	0
spectre.pdf	167	2	0	0	72	1	0	0
myurl.com	3	0	0	0	0	0	0	0
virtual112233@gmail.com	11	11	0	0	17	0	0	0
virtual112233%40gmail.com	13	3	0	0	6	1	0	0
@thisis4testing1	25	2	0	0	2	0	0	0
%40thisis4testing1	4	2	0	0	3	0	0	0
Chrome Safe Storage key (*Chrome only*)	-	-	-	-	4	4	4	3
File recovery								
spectre.pdf	Yes	No	No	No	Yes	No	No	No
cert9.db (*Firefox only*)	No	Yes	No	No	-	-	-	-
key4.db (*Firefox only*)	No	Yes	No	No	-	-	-	-
logins.json (*Firefox only*)	No	Yes	No	No	-	-	-	-
Login Data (*Chrome only*)	-	-	-	-	Yes	No	No	No

Table 2
Summary of the analysis of memory dumps performed in environment B. 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 or not it was possible to retrieve those files.

	Firefox				Chrome			
	T1	T2	T3	T4	T1	T2	T3	T4
Keyword searches								
kernel bugs	8	0	0	0	42	0	0	0
kernel%20bugs	11	0	0	0	25	0	0	0
kernel+bugs	37	0	0	0	55	0	0	0
kernel%2Bbugs	27	0	0	0	27	0	0	0
Syzbot and the Tale of...	14	0	0	0	36	5	0	0
prov=	2	0	0	0	2	0	0	0
spectre.pdf	159	0	0	0	75	0	0	0
myurl.com	3	0	0	0	0	0	0	0
virtual112233@gmail.com	7	0	0	0	17	0	0	0
virtual112233%40gmail.com	7	0	0	0	6	0	0	0
@thisis4testing1	7	0	0	0	2	0	0	0
%40thisis4testing1	2	0	0	0	2	0	0	0
Chrome Safe Storage key (*Chrome only*)	-	-	-	-	5	4	2	2
File recovery								
spectre.pdf	Yes	No	No	No	Yes	No	No	No
cert9.db (*Firefox only*)	No	Yes	No	No	-	-	-	-
key4.db (*Firefox only*)	No	No	No	No	-	-	-	-
logins.json (*Firefox only*)	No	Yes	No	No	-	-	-	-
Login Data (*Chrome only*)	-	-	-	-	Yes	No	No	No

Firefox, it was possible to save new entries and use existing entries from the private mode. This could be a problem since saving a new entry from the private mode would be storing information on disk about the visited websites. In the case of Chrome, it was not possible to add new entries to the keychain from the incognito mode, but it was possible to use the existing ones. Firefox should probably add a warning message when adding new entries from private mode, at least.

In this work, the contents of the swap memory have not been analyzed. Swap memory contains memory pages that have been copied to disk due to a low amount of available RAM. It is important to realize that if a browser is being used in private mode, and the amount of RAM is very limited, there is a possibility that some of the memory pages allocated to the browser will be stored in the swap. This could be a problem as it would imply that memory pages with possible sensitive information would be stored on disk.

6.2. RAM

The tests performed show that neither Firefox nor Chrome stored browsing-related information on the hard disk. However, as can be seen in the results section, it was possible to retrieve artifacts related to the browsing session from memory.

It is important to note that if only the memory space associated with the browser had been dumped, regions of memory that were allocated to the browser process, but were freed, would have been left unanalyzed. This is a problem because those memory regions may still contain sensitive information. This behavior is clearly seen when, after restarting the machine, it was still possible to retrieve information from the activity performed during browsing.

To facilitate the discussion of this subsection, the memory dumps will be discussed according to when each was created.

Table 3
Summary of the analysis of memory dumps performed in environment C. 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 or not it was possible to retrieve those files.

	Firefox				Chrome			
	T1	T2	T3	T4	T1	T2	T3	T4
Keyword searches								
kernel bugs	13	4	0	0	43	0	0	0
kernel%20bugs	26	13	0	0	17	6	0	0
kernel+bugs	41	27	0	0	56	1	0	0
kernel%2Bbugs	47	18	0	0	31	11	0	0
Syzbot and the Tale of...	16	4	0	0	40	0	0	0
prov=	2	0	0	0	2	0	0	0
spectre.pdf	156	60	0	0	76	11	0	0
myurl.com	3	2	0	0	0	0	0	0
virtual112233@gmail.com	8	1	0	0	20	2	0	0
virtual112233%40gmail.com	10	4	0	0	10	1	0	0
@thisis4testing1	24	2	0	0	4	0	0	0
%40thisis4testing1	4	0	0	0	4	0	0	0
Chrome Safe Storage key (*Chrome only*)	-	-	-	-	2	0	0	0
File recovery								
spectre.pdf	Yes	No	No	No	Yes	No	No	No
cert9.db (*Firefox only*)	Yes	No	No	No	-	-	-	-
key4.db (*Firefox only*)	Yes	No	No	No	-	-	-	-
logins.json (*Firefox only*)	Yes	No	No	No	-	-	-	-
Login Data (*Chrome only*)	-	-	-	-	Yes	No	No	No

Table 4
Summary of the analysis of memory dumps performed in environment D. 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 or not it was possible to retrieve those files.

	Firefox				Chrome			
	T1	T2	T3	T4	T1	T2	T3	T4
Keyword searches								
kernel bugs	17	0	10	0	81	0	8	0
kernel%20bugs	51	15	26	0	41	8	36	0
kernel+bugs	62	30	16	0	109	2	16	0
kernel%2Bbugs	67	26	49	0	57	22	68	0
Syzbot and the Tale of...	21	4	12	0	60	0	10	0
prov=	3	0	0	0	3	0	0	0
spectre.pdf	254	216	230	0	110	23	45	0
myurl.com	6	6	10	0	0	0	0	0
virtual112233@gmail.com	26	10	16	0	34	27	26	0
virtual112233%40gmail.com	16	20	18	0	11	8	8	0
@thisis4testing1	54	26	63	0	8	5	4	0
%40thisis4testing1	3	6	6	0	7	7	14	0
Chrome Safe Storage key (*Chrome only*)	-	-	-	-	6	5	2	0
File recovery								
spectre.pdf	Yes	No	No	No	Yes	No	No	No
cert9.db (*Firefox only*)	Yes	Yes	No	No	-	-	-	-
key4.db (*Firefox only*)	Yes	Yes	No	No	-	-	-	-
logins.json (*Firefox only*)	Yes	Yes	No	No	-	-	-	-
Login Data (*Chrome only*)	-	-	-	-	Yes	Yes	No	No

6.2.1. Memory dump in T1

The first activity performed was to access YouTube and watch a video. Regardless of the environment or browser, it was possible to retrieve both the keywords entered in the search field and the full name of the video played. It is worth mentioning that in the pre-planned browsing session, it was decided to use YouTube as an example. However, this demonstrates that it is possible to retrieve the words entered in a search field as well as the chosen result, whatever the web page used.

The second activity was to access a website in order to later attempt to retrieve the name and content of the cookie created. As in the previous activity, regardless of the environment, it was possible to retrieve the cookie created. The value of this cookie is not particularly interesting. However, it demonstrates that it is possible to retrieve cookies set by any website. For example, it would be possible to retrieve the cookies associated with a web mail service and copy them to another computer to impersonate that user, as long as the cookie is still valid.

Activity number three consisted of viewing a PDF file directly in the browser. As can be seen in the previous section, both the file name and the complete contents were found in memory.

One of the tests that may seem irrelevant is number four of the pre-planned browsing session. In it, a URL was entered in the address bar but was not accessed. The results show that it was only possible to retrieve the words entered in the case of Firefox. Taking into account that search suggestions are disabled by default in private mode, the two most likely reasons why this behavior was observed are: 1) the browser probably constructs the entire URL in the background (adding, for example, `http://`) as it is typed and 2) the browser allows words entered in the address bar to be used

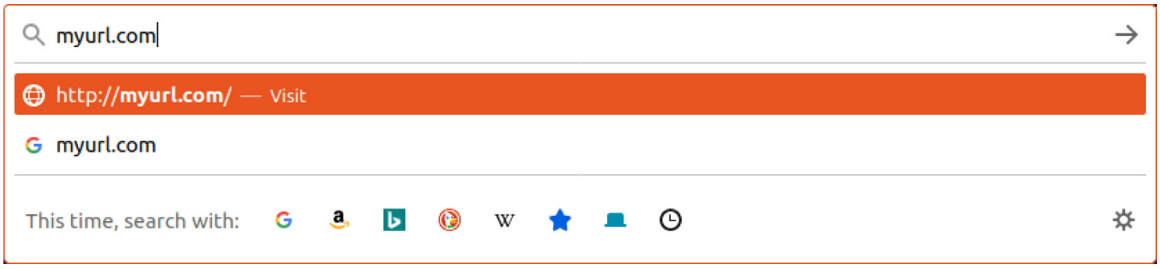

Fig. 4. Suggested options when typing in the Firefox address bar.

as keywords to be searched for in a web search engine (by default Google). Fig. 4 shows this behavior in Firefox when typing in the address bar.

In a similar way to activity three, the idea behind activity number five was to see if it was possible to retrieve the files needed to obtain all the passwords stored in the browser. This may seem irrelevant considering that these files were already permanently stored on disk. However, this test becomes relevant if the case where the browser profile was stored on an removable device is considered. This device could be plugged into a shared computer in order to check email, for example. Once finished, the device would be disconnected and, in theory, it should not be possible to retrieve the login information entered. Moreover, even if the computer used had a keylogger installed, either hardware or software, the login information would not have been leaked. This is due to the fact that this information would have been entered automatically by the browser's keychain and would not have been entered by keyboard. The problem is that the files where the browser stores the passwords could have remained in memory and not yet been overwritten. This means that if those files were to be retrieved from memory, it would be possible to access not only the username and password used at that moment but all the usernames and passwords stored in the keyring. As can be seen in the results tables, in the case of Firefox, the files associated with the keychain were not found intact in memory in environments A and B. However, they were found when the browser was executed in environments C and D. In the case of Chrome, it was possible to recover the `Login Data` file in all the environments.

The last activity was to try to log in to Gmail. The results show that, regardless of the environment or browser, it was possible to recover, in full, both the username and password entered. Although the login fails, the objective of this point was to see if it was possible to recover the login data.

The results of analyzing the memory dumps with the browser running show that it was possible to recover practically all the activities performed. In the case of Firefox it was not possible to recover the files containing the keychain in the first two environments and, in the case of Chrome, it was not possible to recover the myurl.com address.

6.2.2. Memory dump in T2

Table 1 shows that, in environment A, the amount of information that was possible to retrieve is drastically reduced after the browser was closed. In the case of Firefox, it was only possible to obtain the name of the previewed PDF and the Gmail username and password, as well as the Firefox profile files containing the logins database. In the case of Chrome, it was possible to retrieve the words entered in the YouTube search field, the title of the played video, the name of the previewed PDF file, the Gmail username, and the Chrome Safe Storage key. Although it was possible to retrieve the key, the `Login Data` file, which actually contains the keychain, was not found in memory.

Table 2, associated with environment B, reveals a whole different situation. In the case of Firefox, it was only possible to recover two of the three files needed to obtain the contents of the keychain. The rest of the information generated by Firefox seems to have been successfully deleted from memory thanks to the `init_on_free=1` boot option. However, in the case of Chrome, it was possible to retrieve the title of the played video and the Chrome Safe Storage key. Therefore, the situation seems to have improved with respect to the previous environment from a user privacy point of view.

Table 3, corresponding to the analysis of the environment C, presents a totally different situation from that of environment B. The amount of information that could be retrieved after shutting down the virtual machine is substantial. In the case of Firefox, it was possible to recover all the activities performed with the exception of the cookie set by stackoverflow.com, the previewed PDF file, and the profile files containing the keychain. In the case of Chrome, the situation seems slightly better. Only the words entered in the YouTube search field, the name of the previewed PDF file, and the Gmail username could be retrieved.

Table 4, corresponding to the analysis of the environment D, shows a slightly worse situation than the previous one. In the case of Firefox, the only information that could not be retrieved was the cookie set by stackoverflow.com and the previewed PDF file. In the case of Chrome, the title of the video played, the stackoverflow.com cookie, the myurl.com address, and the previewed PDF file could not be retrieved. It is noteworthy that in both browsers it was possible to recover the complete keychain after the virtual machine was completely shut down.

6.2.3. Memory dump in T3

After the computer was restarted, no artifacts associated with the navigation performed in any of the first three environments could be recovered. It is true that the Chrome Safe Storage key could be recovered in the first two environments. This is because the key is stored in the system keychain, which is unlocked when the user logs in. As in environment C the virtual machine was not started after rebooting the computer, the associated key was not loaded.

The situation presented in the environment where the browser is executed in a VMware virtual machine is totally opposite to the rest of the environments. The only artifacts that could not be retrieved in the case of Firefox were the stackoverflow.com cookie, the previewed PDF file and the profile files containing the keychain. In the case of Chrome, the artifacts that could not be recovered were the stackoverflow.com cookie, the myurl.com address, the previewed PDF file and the `Login Data` file.

These results show that adding a hypervisor in between may affect memory management, which can work against user privacy. This is a curious result since the virtual machine represents an environment where RAM is more limited, so information in memory would be expected to be more easily overwritten and harder to retrieve.

It is also possible to see that a simple reboot did not prevent sensitive information from being recovered in environment D. As mentioned in Section 3.4.2, some BIOSes erase the contents of the RAM on restart. If the BIOS of the computer used for testing had erased the RAM memory on reboot, the results obtained in T3 would have been the same as in T4.

6.2.4. Memory dump in T4

After having the computer turned off for 10 seconds, it was not possible to recover any information associated with the navigation, regardless of the environment or browser used. The only information that could be retrieved was the Chrome Safe Storage key in the first two environments. As explained above, this is because the key is stored in the system's keychain, which is unlocked when the user logs in.

7. Conclusion

This paper presents a methodology to test the effectiveness of the private mode included in the different web browsers. This methodology consists of performing a comprehensive forensic analysis of different machine configurations under different conditions after having completed a predefined browsing session using the private mode.

As an example of application of the proposed methodology, it was applied to Firefox and Chrome running on four different Linux environments. The memory and hard disk were analyzed for any artifacts that were generated during the private browsing sessions. Despite being a targeted analysis, it shows the amount of information that can be recovered from a complete memory dump when a comprehensive and in-depth analysis is performed. It has also been found that running Firefox or Chrome in a VMware virtual machine can decrease the level of privacy, allowing sensitive information to be recovered even after rebooting the computer, being necessary to turn off the computer for a minimum amount of time to guarantee the memory is emptied.

As future work it can be highlighted to apply this methodology to new combinations of operating system, hypervisor, and web browser. In addition, this methodology could be easily adapted to mobile platforms such as Android, since LiME can dump the memory of an Android device. One area where the scope of the methodology could be extended would be by adding a new environment with a very limited amount of RAM. This would allow to test what information can be retrieved from the swap, since swap memory was not analyzed in this work.

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.

CRediT authorship contribution statement

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

References

Anderson, D., von Seck, R., 2020. The GDPR and its impact on the web. In: Carle, G., Gõnther, S., Jaeger, B. (Eds.), Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM). Chair of Network Architectures and Services, Munich, pp. 1–5.

Anuradha, P., Kumar, T.R., Sobhana, N., 2016. Recovering deleted browsing artifacts from web browser log files in Linux environment. In: 2016 Symposium on Colossal Data Analysis and Networking (CDAN). IEEE, pp. 1–4.

Bauer, J., Gruhn, M., Freiling, F.C., 2016. Lest we forget: cold-boot attacks on scrambled DDR3 memory. Digital Invest. 16, S65–S74.

Berghel, H., 2017. Equifax and the latest round of identity theft roulette. Computer (Long Beach Calif) 50 (12), 72–76.

Breve, B., Caruccio, L., Cirillo, S., Desiato, D., Deufemia, V., Polese, G., 2020. Enhancing user awareness during Internet browsing. In: Proceedings of the Fourth Italian Conference on Cyber Security, ITASEC 2020. In: CEUR Workshop Proceedings, Vol. 2597, pp. 71–81.

Caruccio, L., Desiato, D., Polese, G., Tortora, G., 2020. GDPR Compliant information confidentiality preservation in big data processing. IEEE Access 8, 205034–205050.

Cirillo, S., Desiato, D., Breve, B., 2019. CHRAVAT-Chronology awareness visual analytic tool. In: 2019 23rd International Conference Information Visualisation (IV). IEEE, pp. 255–260.

Felt, A., Evans, D., 2008. Privacy protection for social networking platforms. Workshop on Web 2.0 Security and Privacy. Oakland, CA

Findlay, C., Leimich, P., 2014. An assessment of data leakage in Firefox under different conditions. In: 7th International Conference on Cybercrime Forensics Education & Training, CFET 2014. Canterbury, UK

Graziano, M., Lanzi, A., Balzarotti, D., 2013. Hypervisor memory forensics. In: International Workshop on Recent Advances in Intrusion Detection. Springer, pp. 21–40.

Greenwald, G., 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

Greenwald, G., 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

Gruhn, M., Müller, T., 2013. On the practicability of cold boot attacks. In: 2013 International Conference on Availability, Reliability and Security. IEEE, pp. 390–397.

Horsman, G., 2017. A process-level analysis of private browsing behavior: A focus on Google Chromes incognito mode. In: 2017 5th International Symposium on Digital Forensic and Security (ISDFS). IEEE, pp. 1–6.

Horsman, G., Findlay, B., Edwick, J., Asquith, A., Swannell, K., Fisher, D., Grieves, A., Guthrie, J., Stobbs, D., McKain, P., 2019. A forensic examination of web browser privacy-modes. Forensic Science International: Reports 1, 100036.

Jadoon, A.K., Iqbal, W., Amjad, M.F., Afzal, H., Bangash, Y.A., 2019. Forensic analysis of Tor browser: a case study for privacy and anonymity on the web. Forensic Sci. Int. 299, 59–73.

Kollár, I., 2010. Forensic RAM dump image analyzer. Charles University in Prague, Faculty of Mathematics and Physics.

Krasnova, H., Eling, N., Schneider, O., Wenninger, H., Widjaja, T., Buxmann, P., 2013. Does this app ask for too much data? The role of privacy perceptions in user behavior towards Facebook applications and permission dialogs. In: Proceedings of the 21st European Conference on Information Systems, ECIS 2013. Association for Information Systems, pp. 1–12.

Kretschmer, M., Pennekamp, J., Wehrle, K., 2021. Cookie banners and privacy policies: measuring the impact of the GDPR on the web. ACM Trans. Web 15 (4), 1–42.

Montasari, R., Peltola, P., 2015. Computer forensic analysis of private browsing modes. In: International Conference on Global Security, Safety, and Sustainability. Springer, pp. 96–109.

Muir, M., Leimich, P., Buchanan, W.J., 2019. A forensic audit of the Tor Browser Bundle. Digital Invest. 29, 118–128.

Nelson, R., Shukla, A., Smith, C., 2020. Web Browser Forensics in Google Chrome, Mozilla Firefox, and the Tor Browser Bundle. In: Digital Forensic Education. Springer, pp. 219–241.

Sylve, J., 2012. LiME-Linux memory extractor. In: Proceedings of the 7th ShmooCon Conference.

Teng, S.-Y., Wen, C.-Y., 2018. A forensic examination of anonymous browsing activities. Forensic Science Journal 17 (1), 1–8.

Tsalis, N., Mylonas, A., Nisioti, A., Gritzalis, D., Katos, V., 2017. Exploring the protection of private browsing in desktop browsers. Computers & Security 67, 181–197.

Yitbarek, S.F., Aga, M.T., Das, R., Austin, T., 2017. Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, pp. 313–324.

Xosé Fernández-Fuentes obtained his B.Sc. Computer Science degree from the Universidade de Santiago de Compostela (Spain) in 2016 and a MSc Degree in Information and Communication Technologies Security from the Universitat Oberta de Catalunya, Universitat Rovira I Virgili, and Universitat Autşnoma de Barcelona (Spain) in 2017. Currently, he is a PhD Student at the Research Center in Intelligent Technologies (CiTIUS) of the Universidade de Santiago de Compostela. His main research interests include computer security in general, cloud computing, data privacy, and virtualization.

Tomás F. Pena got his Ph.D. in Physics in 1994 in the University of Santiago de Compostela (Spain). From 1994, he is an associate professor in the Department of Electronics and Computer Science of the University of Santiago de Compostela. From 2010, he is a member of the Research Center in Intelligent Technologies (CiTIUS) of this University. His main research lines include the high performance computing in general, the architecture of parallel systems, the development of parallel algorithms for clusters and supercomputers, the optimization of the performance in irregular

codes and with sparse matrices, the prediction, and improvement of the performance of parallel applications in general, the development of applications and middleware for Grid and Cloud, and the use of Big Data technologies for scientific and NLP applications.

José C. Cabaleiro got his Ph.D. in Physics in the University of Santiago de Compostela (Spain). From 1994 he is an associate professor in the area of Computer Architecture in the Department of Electronics and Computer Science in the University of Santiago de Compostela. From 2010, he is a member of the Research Center in Intelligent Technologies (CiTIUS) of this University. His main lines of interest include the architecture of parallel systems, the development of parallel algorithms for irregular problems and with sparse matrices, prediction, and improvement of the performance of parallel applications, optimization of the memory hierarchy in irregular problems and development of applications and middleware for Cloud and Big Data.