On the Privacy of Private Browsing – A Forensic Approach

Kiavash Satvat, Matthew Forshaw, Feng Hao, Ehsan Toreini

School of Computing Science
Newcastle University

Abstract. Private browsing has been a popular privacy feature built into all mainstream browsers since 2005. However, despite its prevalent use, the security of this feature has received little attention from the research community. In this paper, we present an up-to-date and comprehensive analysis of private browsing across four most popular web browsers: IE, Firefox, Chrome and Safari. We report that all browsers under study suffer from a variety of vulnerabilities, many of which have not been reported or known before. Our work highlights the complexity of the subject and calls for more attention from the security community.

1 Introduction

In 2005, Safari first introduced private browsing, a feature that enables a user to surf the Internet without leaving traces on her local computer, such as history, cookies and temporary files [5]. All other mainstream browsers have since added the feature, including Internet Explorer (IE) [4], Chrome [1] and Firefox [2].

Although the basic aim of private browsing is the same, the implementations vary greatly across different browsers. This adds significant complexity to the subject. So far only few researchers have attempted to investigated the subject. In 2010, Aggarwal *et. al.* first initiated the security analysis of private browsing in [5]. In particular, they defined a threat model, surveyed the main usage of private browsing, reviewed the open source code of Firefox, and studied the effect of Firefox extension on private browsing. In 2011, Said *et al.* continued the investigation by examining the content in the volatile memory and they found artifacts left in memory about user activities in the private session even after the session had been closed [6]. Apart from these two publications [5,6], the subject of the security of private browsing seems to have been mostly neglected by the research community.

In this paper, we extend the earlier works in several aspects. First, we refine the threat model in [5] to capture more realistic threats in practice. Second, we carry out more extensive experiments than [5]: covering not only Firefox, but also IE, Chrome and Safari. Third, we scrutinise artefacts left from private browsing from all angles: not only in memory as in [6], but also in disk and network traffic.

2 Research methodology

In this research work, we took a forensic approach to collect and analyse residual data left on the host computer after the private browsing session. Virtualisation was used to prevent any cross-contamination between experiments. In particular, VMware Player (a free version of VMware) was installed [10]. Windows 7 was chosen based on its popularity among the Internet users. The latest versions of the four most popular browsers (as in April, 2013 [9]) were installed: Mozilla Firefox (19.0), Apple Safari (5.1.7), Google Chrome (25.0.1364.97) and IE (10.0.9200.16521).

For each experiment, a fresh Windows installation with a single web browser was used. The experiments were carried out for each browser to investigate possible residual data left in memory or disk after private navigation. To allow other researchers to easily replicate the experiments, we only used *freely* available forensic tools. Finally, all the software tools developed during the course of this research are released as open source (see [20]). We hope this would help browser vendors evaluate the security of their products and improve accordingly.

3 Threat model

Same as in [5], we categorise attackers into two types: local and remote. A local attacker is someone who has physical access to a user's machine. The threat model defined in [5] restricts the local attack to "after the fact" forensics. On the other hand, it is acknowledged in [5] that the user may have installed third-party browser extensions before the private session. Our model about a local attacker is essentially the same as that in [5] but with one difference: we explicitly assume at least one of the installed third-party extensions were written by an attacker. Instead of surveying the third-party extensions and speculating their behavior as in [5], we write our own extensions as if from an attacker's perspective. This allows capturing the exact impact of extensions more directly.

For remote attacks, we assume the attacker is capable to engage with the user in a web browsing session over HTTP(S). This typically happens when a user navigates to a web site that is controlled by an attacker, whose goal is to detect whether the user is in the private mode[1]. As compared with the model in [5], we have excluded the threat of remote websites tracking users (e.g., based on IP addresses [7] or unique browser fingerprints [14]). This is because private browsing has never been designed to prevent web tracking [1,2,3,4]. (We refer interested readers to other privacy-preserving tools such as TOR [15] for the prevention of web tracking.)

Against the defined threat model, we conducted a series of experiments to assess the security of private browsing among the four most popular browsers:

[1] Given the often negative connotation of using the private mode for viewing adult websites (see [5]), we consider the fact of using the private mode a privacy feature by itself. If the remote website learns the user is in the private mode, it may push more adult-oriented advertisement to the user.

	Firefox	**Chrome**	**IE**	**Safari**
Domain name system	√	√	√	√
Memory inspection	√	√	√	√
File timestamp	−	√	−	√
Index.dat *	N/A	N/A	√	N/A
SQLite database crash *	√	√	N/A	√
SQLite added bookmark *	√	√	N/A	√
Extension *	√	√	−	√
Cross-mode Interference *	N/A	√	N/A	N/A
Hyperlink attack	√	√	√	√
Timing attack *	√	√	−	√

Table 1: List of attacks and their applicability to each browser. Those marked with * contain new results discovered by our study, while others correspond to attacks that have been previously known but validated again by our study.

Firefox, Chrome, IE and Safari. Table 1 summarises the attacks, and their applicability to specific browsers.

4 Local attacks

4.1 Summary of previously known attacks

Domain Name System (DNS). DNS caching has long been known as a major threat to private browsing [5]. This vulnerability is caused due to the operating system caching all DNS queries sent by a web browser. We confirm that this vulnerability still persists in all browsers three years after it was reported in [5]. Third-party extensions have been developed to address this issue [11,12], but none of them has been adopted by browser vendors.

Memory Inspection. In 2011, Said *et al.* reported that artifacts from a private browsing session were found in the main memory after the end of the session [6]. We have verified that the same vulnerability still exists in the latest versions of all fours browsers. After navigating a few websites in the private mode and closing the session, we inspected the content in RAM and discovered traces of private navigation, including visited URLs, password and cookies.

File timestamp. In [5], the authors compared the “last modified date” of files in the Firefox profile directory before and after private browsing. They found the timestamps had been changed while file sizes remained the same, which allows deducing the occurrence of a private session in the past. Our experiments show the vulnerability has been fixed in the latest version of Firefox (and also IE), but it still exists in Chrome and Safari.

4.2 Index.dat

The Index.dat files are binary format log files used by IE to store the user’s browsing history, cookies, temporary files, etc. We analyse these files in order to evaluate the correlation between IE’s InPrivate mode and Index.dat files. After the navigation of the targeted websites in the private mode, we scrutinise residual traces left in the files. Unlike in some earlier versions of IE, the latest version has successfully removed the traces of visited websites in the private mode.

However, we found that adding bookmarks in the IE private mode could lead to information leakage. Bookmarks added during a private session were stored as standalone files with corresponding creation timestamps. On the other hand, there is no matching URL for the added bookmark in `Histoy.IE5\index.dat`. A comparison between these files could allow an attacker to deduce that the bookmark was added in the private mode and when. False positives may occur if the user added a bookmark in the usual mode without visiting the page (e.g., right-click over a hyperlink to add it to the bookmarks). However, the false negatives are always zero.

4.3 SQLite Database

SQLite databases are used by Firefox, Chrome and Safari to store historical records of browsing activities [13]. We study the correlation between private browsing and the underlying SQLite database and reveal two vulnerabilities: one related to the application crash, and the other related to adding bookmarks.

Application Crash. There are many reasons why a browser program may terminate in an unexpected way, e.g., sudden power loss or system crash. The critical question is that: if the program terminates in an unexpected way, will it leave unexpected evidence on disk?

In Firefox, the SQLite database uses the Write Ahead Logging (WAL) mode to implement database transactions such as atomic commit and rollback. In the event of application crash, database connections are not closed cleanly and the WAL files will remain on disk until the browser is restarted. We observed that the WAL files left from the private mode always had the zero size (since there were not database updates), while the WAL files left from the usual mode had non-zero size. Hence, based on the size of a WAL file and its timestamp, an attacker will be able to deduce that a private session occurred at a specific time.

Chrome implements the SQLite database transactions using Journal files instead of the WAL files. To speed up the loading, the browser uses two SQLite databases to store the history records; a primary "`History`" database and monthly digests in the form of "`History Index YYYY-MM`". In the usual mode, the browser uses a journal file for each database. However, in the private mode, it just uses one journal file for the "`History`" database only. All journal files will remain on disk in the event of application crash or power loss. Based on the existence of only one journal file, an attacker can deduce that a private session occurred and the timestamp of the file reveals when. Similar to Firefox, restarting the browser in the usual mode will remove the evidence.

The case of Safari is more serious. Unlike Firefox and Chrome that only use in-memory SQLite database for private browsing, Safari first writes records of the visited websites to the database file and then removes them after the browser is closed normally. We found that if the browser was closed in an abnormal way (e.g., manual termination), the records of visited websites in the private mode would remain in the database. The residual data persists on disk even after the browser is restarted, which poses a serious threat to the user's privacy. As a countermeasure, we recommend Safari to adopt in-memory SQLite updates, like Chrome and Firefox.

Adding Bookmarks. In Firefox, after visiting targeted websites and adding a bookmark in the private mode, we examine the `places.sqlite` SQLite database, which contains records of all visited URLs and added bookmarks. Our investigation revealed that a bookmark added during the private mode was recorded with empty "title" and "last_visit_date" fields, disclosing that the bookmark was added during the private mode at a specific time. It is worse than the earlier IE case, since the evidence is definite: i.e., zero false positive and zero false negative. The case of Chrome is similar. The URL for bookmarks added in the private mode could be found in the "`history`" SQLite database. Unlike a bookmark recorded in the usual mode, the "visit_count" field was always set to 0 and the "hidden" field set to 1.

The case of Safari is the most problematic. Under the normal operation, Safari removes the browsing history in the private mode when the program is closed. However, we found that as long as the user added one bookmark during the private navigation, all the websites that were visited during the private session would remain in the SQLite database. (A bug report on this issue has been filed to Apple.)

4.4 Extensions

Chrome Extension. We developed a Chrome extension (the source code in [20]) that, once enabled in the private mode, was able to record detailed user activities for the duration of a private browsing session. This includes when the tabs were opened and closed, which web pages were visited and at which time, how the user moved between tabs and windows, etc. In the latest version of Chrome, extensions are disabled in the private mode by default. This "disable-by-default" policy significantly alleviates the threat. However, the fact that Chrome allows the private and usual modes to run in parallel renders this policy ineffective, as we will explain in Section 4.5.

Internet Explorer Extension. We developed an IE extension [20] to obtain the URL and the content of the HTML pages based on using the Browser Helper Object (BHO) class. Like Chrome, IE disables extensions in the private mode by default. However, even after we manually enabled extensions in the private mode, we found the extension had only restricted privilege: in particular, it could no longer invoke the BHO class. Hence, our attack did not work on IE.

Safari and Firefox Extensions. We developed similar extensions for Safari and Firefox [20], which were able to record details of the user's activities within a private session [20]. In both Safari and Firefox, extensions are enabled by default in the private mode. Hence, they are vulnerable to extension attacks. The countermeasure we recommend is to disable extensions by default in the private mode, just like in IE and Chrome.

4.5 Cross-mode interference

While extensions in Chrome are disabled by default in the private mode, Chrome allows the usual and private modes to run in parallel, providing the attacker an opportunity to exploit cross-mode interference.

The attack was motivated by the following observation: the `Chrome://memory` page displays all the opened tabs in the browser regardless if they are in the usual or private mode. Accordingly, we developed an extension [20] using the standard Chrome extension APIs [18].

The attack works as follows. In the usual mode, the extension is enabled by default and it is able to invoke standard APIs to list all tabs, each having a unique ID. If the tab is in the usual mode, the extension can obtain further details about the tab, such as the page title and URL. However, if the tab is in the private mode, no response will be given. This lack of response provides an indication that the queried tab is in the private mode. By periodically polling the tabs, the extension can detect the existence of a private browsing session, the number of active tabs opened in the private mode, and when those tabs are opened and closed.

Chrome also provides experimental APIs (which are enabled in `chrome://flags`) to further enforce the extension's functionality [19]. In particular, it provides the following additional information about each tab: the CPU consumption, network bandwidth and Frames Per Section (FPS). This information is obtainable even for tabs in the private mode.

The extra information allows the attacker to draw an even more fine-grained profile about the user activities within a private session. Figure 1 shows how the user's activities are correlated with the CPU consumption and network bandwidth usage. Loading new pages increases the CPU and bandwidth usage at the same time while scrolling pages only affects the CPU consumption. When one is watching an HTML5 video, there is a substantial increase of both the CPU usage and network bandwidth. As a countermeasure, we recommend the browser should always be run in a single mode. This applies to all other browsers.

5 Remote attacks

5.1 Hyperlink attack

A conventional technique adopted by all browsers to distinguish visited links from unvisited ones is by changing colour, hence improving the user's browsing experience [8]. However, there are noticeable deviations for the same mechanism to work in the private mode. As we have tested, all browsers started a new private session with all hyperlinks displayed in blue. Furthermore, in Chrome, Firefox and Safari, the hyperlink never changes colour even after the user has clicked the link or visited the URL. (One might argue that this has the benefit of making it more difficult for the remote website to track the visited pages than in the usual mode since the color of the hyperlink does not change much; however,

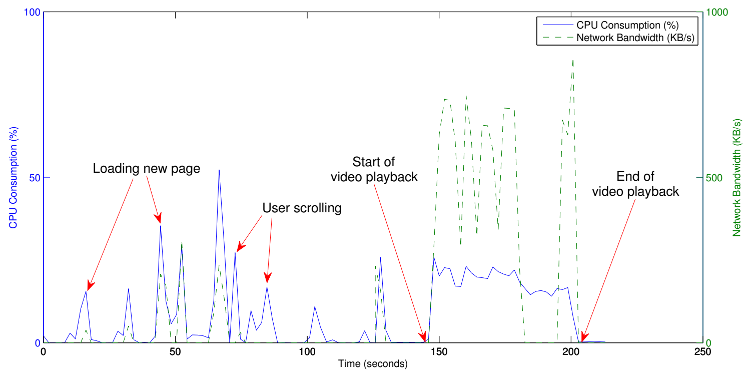

Fig. 1: Profiling the user activities in the private mode

it is worth noting that defence against web tracking is not within the threat model of private browsing.)

These deviations create an exploit path for a remote attacker. Based on the difference in the hyperlink colours, the remote attacker is able to tell a private mode apart from a usual mode. For example, since in Chrome, Firefox and Safari, the hyperlinks are persistently blue in the private mode, a remote website can use JavaScript to check the colour of the hyperlinks and easily tell if the user is currently in the private mode. This vulnerability was first reported in [5] and we find it still exists in the latest versions of the browsers. However, the case of IE is different from the rest browsers; the colour of the hyperlink does change based on the user's clicking just like in the usual mode. However, the private mode still deviates from the usual mode in that the former always displays all hyperlinks in blue in the beginning of the session. Hence, if the remote attacker is able to regularly engage with the user in more than one sessions (e.g., the remote attacker controls a news website), he can easily tell if the user is in the private mode.

As a countermeasure, we suggest to remove deviations of how hyperlinks are colored between the usual and private modes. The only difference should be that the private mode does not save any information about visited links after the session is closed.

5.2 Timing attack

In this section, we describe a novel timing attack, which is able to remotely detect the private mode based on measuring the time of writing a large number of cookies. We developed a simple PHP and MySQL application to measure the time taken to write a predefined number of cookies, and then store these results to a database for further analysis. The Selenium testing framework [17] was used to automate testing for large-scale experimentation.

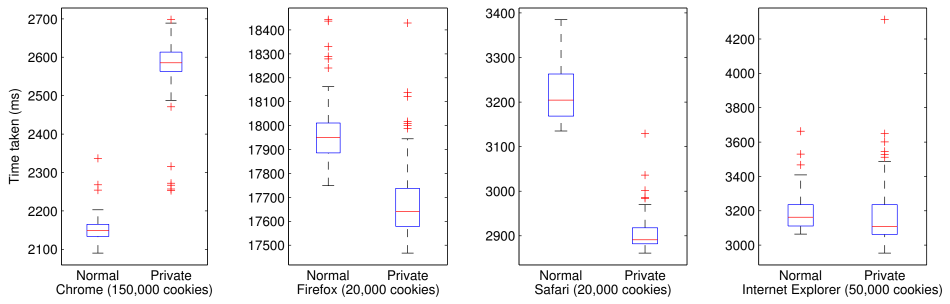

Fig. 2: Box plots representing timing data collected for browsers under test.

Browser	Equal Error Rate (EER)	Threshold (t)
Google Chrome	1%	0
Mozilla Firefox	9%	0
Internet Explorer	63%	-0.0002
Apple Safari	1%	0.0055

Table 2: Equal Error Rates for detecting the private mode

We collected extensive timing measurements for the usual and private modes (100 samples per mode per browser) as training data (see Figure 2). We then collected further 100 timing measurements for each browser for each mode for evaluation. The evaluation is based on using a standard z-test [16] (details can be found in the full version of the paper [20]). There are two types of errors in the evaluation. One is the False Acceptance Rate (FAR), that is the rate of a usual session being characterised as the private mode. The other is False Rejection Rate (FRR), that is the rate of a private session being characterised as the usual mode. The two error rates vary according to the threshold. Hence, in the evaluation, we used the Equal Error Rate (EER) where the FRR and FAR curves intersect. In the ideal case, the EER should be close to 50%: i.e., the chance for the attacker to detect the private/usual mode is no better than tossing a coin. However, as shown in Table 2, with the exception of IE, a remote attacker is able to correctly identify the browsing mode with high accuracy.

6 Conclusion

We have revealed a range of vulnerabilities in the existing implementations of private browsing. The problems are generally caused by the following factors: a lack of understanding of the threat model (especially in relation to remote attacks), a lack of appropriate control of running extension in the private mode (and neglect of the cross-mode interference) and a lack of rigorous and systematic test (especially in edge cases such as program crash and adding bookmarks).

References

1. Chrome private browsing mode: https://support.google.com/chrome/bin/answer.py?hl=en&answer=95464&p=cpn_incognito (Accessed: April 2013)
2. Mozilla Firefox private browsing mode: http://support.mozilla.org/en-US/kb/private-browsing-browse-web-without-saving-info (Accessed: April 2013)
3. Safari private browsing mode: http://support.apple.com/kb/PH5000 (Accessed: April 2013)
4. Internet Explorer private browsing mode: http://windows.microsoft.com/en-us/windows-vista/what-is-inprivate-browsing (Accessed: April 2013)
5. G. Aggarwal, E. Burzstein, C. Jackson, D. Boneh, "An analysis of private browsing modes in modern browsers," the 19th USENIX Symphosium on Security, 2010.
6. H. Said, A.H. Mutawa, A.I. Awadhi, M. Guimaraes, "Forensic analysis of private browsing artifacts,". International Conference on Innovations in Information Technology (IIT), 2011.
7. A. Ruiz-Martínez, "A survey on solutions and main free tools for privacy enhancing Web communications," *Journal of Network and Computer Applications*, Vol. 35, No. 5, pp. 1473–1492, 2012
8. J. Collin, A. Bortz, D. Boneh, C.J. Mitchell, "Protecting browser state from web privacy attacks," the 15th international conference on World Wide Web (WWW), 2006.
9. Most popular web browsers: http://www.w3schools.com/browsers/browsers_stats.asp (Accessed: April 2013)
10. VMware Player Version 4.0.0: http://www.vmware.com/products/player/ (Accessed: April 2013)
11. Click & Clean: https://chrome.google.com/webstore/detail/ghgabhipcejejjmhhchfonmamedcbeod?utm_source=chrome-ntp-icon (Accessed: April 2013)
12. Clear DNS Cache: https://addons.mozilla.org/en-us/firefox/addon/clear-dns-cache/ (Accessed: April 2013)
13. S. Jeon, J. Bang, K. Byun, "A recovery method of deleted record for SQLite database," *Personal and Ubiquitous Computing*, Vol. 16, No. 6, pp. 707–715, 2011.
14. P. Eckersley, "How unique is your web browser?", Available at https://panopticlick.eff.org/browser-uniqueness.pdf (Accessed: April 2013)
15. The official website for the TOR project: https://www.torproject.org/ (Accessed: April 2013)
16. E. Kreyszig, "Introductory Mathematical Statistics," John Wiley & Sons Inc, 1970.
17. Selenium: http://seleniumhq.org/ (Accessed: April 2013)
18. Standard Chrome extension API: http://developer.chrome.com/extensions/ (Accessed: April 2013)
19. Experimental Chrome extension API: http://developer.chrome.com/extensions/experimental.html (Accessed: April 2013)
20. Open-source software tools developed for the research of private browsing: http://homepages.cs.ncl.ac.uk/m.j.forshaw1/privatebrowsing/