On the Privacy of Private Browsing - A Forensic Approach

Kiavash Satvat, Matthew Forshaw, Feng Hao, Ehsan Toreini[1]

School of Computing Science
Newcastle University

Abstract

Private browsing has been a popular privacy feature built into mainstream browsers since 2005. However, despite the prevalent use, the security of this feature has received little attention from the research community. To the best of our knowledge, no study has existed that systematically evaluates the security of private browsing across major browsers and from multiple angles: not only examining the memory, but also the underlying database structure on the disk and the web traffic. In this paper, we present an up-to-date and comprehensive analysis of private browsing across the four 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. The problems are generally caused by the following factors: a lax control of permission to allow extensions to run in the private mode with unrestricted privilege; inconsistent implementations of the underlying SQLite database between the private and usual modes; the neglect of the cross-mode interference when the two modes are run in parallel; a lack of attention to side-channel timing attacks, etc. All of the attacks have been experimentally verified with countermeasures proposed.

Keywords: private browsing, web security, user privacy, system security

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

[1]A preliminary short version of the paper was presented at DPM'13 [12].

as history, cookies and temporary files [1]. Since then, all other mainstream browsers have added the same feature, including Internet Explorer (IE) [18], Chrome [15] and Firefox [16].

Although the basic aim of private browsing is the same, the implementations vary greatly across different browsers. This adds significant complexity to the subject. In USENIX Security'10, Aggarwal, Burzstein, Jackson and Boneh first initiated the study of the security of private browsing in modern browsers and discovered several vulnerabilities [1]. In particular, they studied the dire impact of browser extensions on private browsing in Firefox (v3.6). A year later, Said *et al.* continued the study of private browsing [14]. They focused on examining the content in the volatile memory and found that artifacts from the private session remained in memory even after the session had been closed. Recently, in ESORICS'13, Lerner *et al.* presented a software tool that allows automatic verification of the browser extensions' compliance with the private mode [8]. The tool was mainly tested on Firefox extensions, although in principle it could be easily extended to other browsers.

Apart from these three papers, the security of private browsing seems to have been almost entirely neglected by the security research community. To date, no study has existed that systematically analyses the security of private browsing across major web browsers and from multiple angles: not just examining the memory, but also the underlying database structure on the disk and web traffic.

We believe this lack of attention is disproportionate to the importance of the subject. Over the past five years since 2008, private browsing has been extensively used by a significant portion of Internet users (19% according to a survey [1]) to protect their privacy during web navigation [8]. Given the prevalent use of private browsing and the fact that many users are relying on it for privacy, it is important to ensure that private browsing is really as "private" as the browser vendors have claimed.

1.1. Contributions

In this paper, we will present an independent and systematic evaluation of the current state of private browsing in popluar browsers. Our contributions are summarised below.

1. **Threat model**: We refine a threat model for private browsing based on adjusting a previous model (due to Aggarwal *et al.* [1]) in order to capture more realistic threats in practice. This new model provides a

concrete definition of security, which allows us to evaluate the security of private browsing in a systematic manner.

2. **Discovery of new attacks**: We have performed a series of concrete experiments and discovered a number of new vulnerabilities across all the web browsers under study. In particular, the attacks based on application crash, cross-mode interference and remote timing measurements are novel and are demonstrated to work in practice for the first time.
3. **Validation of known attacks**: We have tested all previously known vulnerabilities against the latest versions of web browsers and are able to confirm that some still remain unfixed.

Our preliminary research results were presented as a short paper (8 pages) at the ESORICS workshop on Data Privacy Management in September, 2013 (DPM'13) [12]. They were based on evaluating the latest versions of the mainstream web browsers as of April, 2013. However, being a short paper, only the main outcomes of the attacks are summarized. This journal paper includes full technical details for each of the attacks, especially the working and quantitative analysis of a novel remote timing attack in Section 5.2. Furthermore, suggestions for countermeasures are added in Section 6. We have informed the relevant browser vendors about the attacks and received useful responses that are also included in this paper. Some of the attacks have been fixed as a result. To inform the reader about the latest situation, we have re-tested all our attacks against the newest versions of browsers as of February 2014 with updates to the previous results included in this paper.

1.2. Outline

The rest of the paper is organised as follows. Section 2 explains the research methodology used for this study. Section 3 defines a threat model for private browsing. The next two sections, 4 and 5, present a series of experiments to expose vulnerabilities of private browsing against local and remote attackers respectively. Section 6 discusses countermeasures for discovered vulnerabilities. Finally, Section 7 concludes our study and suggests future research.

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.

Software	Firefox	Chrome	IE	Safari
VMware Player	√	√	√	√
WinHex	√	√	√	√
Index.dat Analyser	N/A	N/A	√	N/A
SQLite browser	√	√	N/A	√
SQLite manager	√	√	N/A	√

Table 1: List of third-party software used in each experiment

Virtualisation was used to prevent any cross-contamination between experiments. In particular, VMware Player (a free version of VMware) was installed [20]. In terms of the operating system, Windows 7 was chosen based on its popularity among the Internet users. The latest versions of the four popular browsers (as in April, 2013 [19]) 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. A set of freely distributed third party tools were chosen (see Table 1), which makes it possible for the reader to replicate the experiments. Finally, all the software tools developed during the course of this research work are released as open source (see [33]). This should help browser vendors to evaluate the security of their products and improve accordingly.

A group of targeted websites were chosen to imitate a real user's behavior in one browsing session, and to examine a variety of elements involved in web browsing. Table 2 lists the group of targeted websites and their characteristics. In each experiment, the targeted websites were visited in the private mode. The subsequent investigation involved closing the private session and searching for any evidence left on the computer. In most scenarios, this included searching for specific keywords such as URL, cookies, or other content of visited web pages.

[2]Siverlight Isolated Storage

[3]This includes HTML5 Session Storage, Local Storage, Global Storage and Database Storage

Website	Characteristics
http://www.samy.pl/evercookie	Ever-cookies include flash cookie, IsoData[2] and HTML 5 storage[3]
https://www.twitter.com/time	Tweets streams and sizeable photos
http://www.bbc.co.uk/persian	Unicode transmission and videos
http://www.youtube.com	Videos and online advertising
https://www.facebook.com	Online chat and online advertising

Table 2: List of targeted websites and their characteristics

3. Threat model

The threat model for private browsing is defined in terms of the attacker's capabilities and their goals. In 2010, Aggarwal *et al.* defined one threat model for private browsing in [1]. Our model is similar to theirs but with some differences, as we will explain. Same as in [1], we will categorise attackers into two types: local and remote attackers.

Local attack. A local attacker is someone who has physical access to a user's machine. The threat model defined in [1] restricts the local attack to "after the fact" forensics. In other words, it is assumed that an attacker cannot have physical access to the user's computer before the private browsing session (otherwise, the attacker may just install a key logger and the attack would be trivial). On the other hand, it is acknowledged in [1] 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 [1] but with one difference: we explicitly assume at least one of the installed third-party extensions were written by an attacker. Hence, instead of surveying the third-party extensions and speculating their behavior as in [1], we chose to write our own extensions as if from an attacker's perspective. This allows capturing the exact impact of browser extensions more directly.

Remote attack. 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. The goal of the attacker is to detect whether the user is in the private mode. Given the often negative connotation of using the private mode (also known as the porn mode) for viewing adult websites [1], we consider

	Firefox	**Chrome**	**IE**	**Safari**	***Information leakage***
Domain name system	√	√	√	√	*browsing history*
Memory inspection	√	√	√	√	*browsing hisotory, passwords, cookies*
File timestamp	–	√	–	√	*when private mode was last used*
Index.dat *	N/A	N/A	√	N/A	*when private mode was last used*
SQLite database crash *	√	√	N/A	√	*minor to serious depending on browsers*
SQLite added bookmark *	√	√	N/A	√	*minor to serious depending on browsers*
Extension *	√	√	–	√	*browsing history*
Cross-mode Interference *	N/A	√	N/A	N/A	*user activities in private mode*
Hyperlink attack	√	√	√	√	*if the user is in private mode*
Timing attack *	√	√	–	√	*if the user is in private mode*

Table 3: 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.

the fact of using the private mode a privacy feature by itself[4]. If the remote website learns the user is in the private mode, the user privacy may be invaded. For example, if the user signs up a newsletter mailing list when he is in the private mode, the remote server might customize the future delivery of the newsletter content and push more adult-oriented advertisements.

As compared with the model in [1], we have excluded the threat of the remote website tracking the user (e.g., based on IP addresses [13] or unique browser fingerprints [4]). This is because in all browsers, private browsing has never been designed to prevent web tracking (see specifications in [15, 16, 17, 18]). Hence, it is not reasonable to insist on a privacy feature that the product is not designed for. (We refer interested readers to other privacy-preserving tools such as TOR [29] 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 popular browsers: Firefox, Chrome, IE and Safari. Table 3 summarises the attacks, and their applicability to specific browsers. Details of each attack will be explained below.

[4]Although the private mode has been commonly used for viewing adult websites according to the study [1], we should note that there are other usages as well, e.g., a person may use the private mode to check emails on his friend's computer to avoid the browser remembering the password.

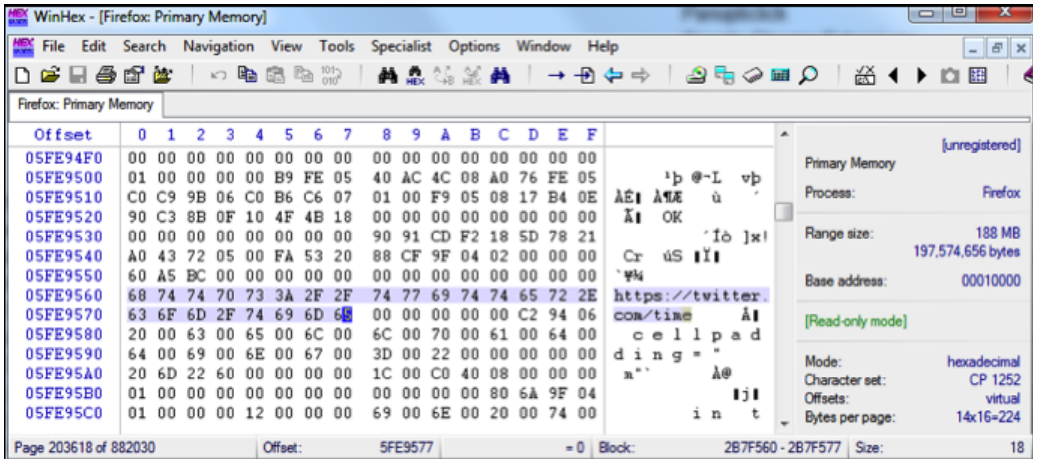


Figure 1: Results of memory inspection after the exit of private session in Firefox. The URL `https://twitter.com/time` was an artifact left from the session.

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 [1]. This vulnerability is caused due to the operating system caching the DNS queries sent by a web browser regardless if it is in the private mode or not. The results of our testing on DNS caching have confirmed that, three years after it was reported in [1], this vulnerability still persists in all browsers. Some third-party extensions have been developed to address this issue [21, 22], but none of them has been adopted by browser vendors.

Memory Inspection. Volatile memory can be a remarkable source of information for forensic investigations. In 2011, Said *et al.* reported that artifacts from a private browsing session were found left in the primary memory after the exit of the session [14]. We conducted experiments to verify if the same vulnerability still existed in the latest versions of browsers.

WinHex [23], a popular forensic tool, was used to extract content from RAM. The experimental results revealed that artifacts from private navigation persisted in memory even after the browser was closed. They included information such as visited URLs, password and cookies (see Fig. 1).

File timestamp. Timestamps are valuable evidence in any forensic investigation [2]. They often serve to reveal the occurrence of an event at a

precise time. In an operating system, each file or folder has a timestamp of the "last modified date". In [1], Aggarwal *et al.* compared the "last modified date" for files in the profile directory before and after the private browsing test using Firefox. They found the timestamps had been changed while the sizes of the files remained the same. Based on this observation, the authors suggested that an attacker could deduce the occurrence of a private session in the past. Our experiments showed that the reported vulnerability had been fixed in the latest version of Firefox. However, the same vulnerability was found to exist 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. A number of such files are dispersed under different paths in Windows 7, with each recording specific information about the visited web pages.

In order to evaluate the correlation between the private mode (also called InPrivate in IE) and 'Index.dat' files, Index.dat Analyser was installed in order to read and analyse binary format log files [24]. After the navigation of the targeted websites in the private mode, Index.dat Analyser was used to scrutinise residual traces left in the files. Our experiments showed that, unlike in some earlier versions of IE, the latest version successfully removed the traces of private navigation: no history, cookies or cache records were found.

However, we found that adding bookmarks in the IE private mode could lead to information leakage. The browser allows users to freely add bookmarks just as in the usual mode. Bookmarks added during a private session were stored as standalone files under the `%USERPROFILE%\Favorites` directory with timestamps of the file creation time. On the other hand, there is no matching URL for the added bookmark in `History.IE5\index.dat`. A cross comparison between these files could allow an attacker to deduce that the bookmark was added in the private mode; the timestamp of the bookmark file would indicate when the private browsing was used. The attacker may sometimes get false positives as the user may have added a bookmark in the usual mode without visiting the link (e.g., right-click over a hyperlink to add it to the bookmarks). However, the false negatives are always zero.

Browser	File directory
Firefox	%APPDATA%\Local\Mozilla\Firefox\Profiles\<Random Value>\<FileName>.sqlite
Chrome	%APPDATA%\Local\Google\Chrome\User Data\<FileName>.sqlite
Safari	%APPDATA%\Local\Apple Computer\Safari\<FileName>.db

Table 4: Paths of the SQLite database files in different browsers

4.3. SQLite Database

SQLite is ANSI-C-based open source database used by Firefox, Chrome and Safari to store historical records of browsing activities [6]. The database is structured as a single file saved under the browser's profile folder. The paths of the SQLite files used by different browsers are summarized in Table 4.

To study the correlation between the private browsing activities and the underlying SQLite database, we installed SQLite Browser [11] and SQLite Manager [25] to examine the records in detail. Our experiments reveal two vulnerabilities: one is related to the application crash, and the other one related to adding bookmarks.

Application Crash. There are many reasons why a browser program may terminate in an unexpected way. For example, the program may be manually terminated by the user due to unresponsiveness; there may be a sudden power loss; the web page contains too much HTML data to load in the browser; the Javascript or Java plug-in embedded in the web page may be too busy to handle the computational load, etc. 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 Write Ahead Logging (WAL) mode to implement database transactions such as atomic commit and rollback. In this mode, WAL files are created at the first connection to the SQLite database and are deleted when the last connection to the database is closed. However, 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 the timestamp of it, an attacker will be able to deduce that a private session occurred at a specific time. However, restarting the browser in the

usual mode will remove this evidence.

Chrome implements the SQLite database transactions using Journal files instead of WAL mode. To speed up the loading, the browser uses two SQLite databases to store the history records: one named "`History`" and the other in the form of "`History Index YYYY-MM`" (for example, "`History Index 2013-04''` for April of 2013). In the usual mode, the browser uses the two journal files for the two databases respectively. However in the private mode, it just uses one journal file (only for the "`History`" database). 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[5].

The case of Safari is more serious. Unlike Firefox and Chrome that only use in-memory SQLite database for private browsing and do not write the visited websites to the database file on disk, Safari actually writes records of the visited websites to the database file and removes them later. Only when the browser is closed normally will those history records be removed. Through experimentation, we found that if the browser was closed in an abnormal way (e.g., a crash or forced termination), the records of visited websites in the private mode would remain in the database. These records will persist on disk even after the browser is restarted. This flaw poses a serious threat to the user's privacy[6].

Adding Bookmarks. In Firefox, after visiting the targeted websites and adding a bookmark in the private mode, SQLite Manager and SQLite Browser [25] were used to examine the database file named `places.sqlite`. This database kepng records of all visited URLs and added bookmarks. Within this database, two are particularly relevant: `moz_places`, which stores the visited websites; and `moz_bookmarks`, which stores bookmarks.

Our investigation revealed that a bookmark added during the private mode was recorded in both tables. This was the same as the implementation

[5]As of February 2014, the Chrome browser (version 32.0.17) has changed the way the history files work. Instead of using two SQLite files with names "`History`" and "`History Index YYYY-MM`", it now uses just one SQLite file named "`Archived History`". Hence, the reported issue has been fixed.

[6]As of February 2014, the Safari browser (version 5.1.7) no longer writes records to the database file during the private browsing. Hence, the reported issue has been fixed.

in the usual mode, except that the "title" and "last_visit_date" fields were deleted (see Fig. 2). A comparison of these two tables would disclose 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.

Similarly, in Chrome, the URL for bookmarks added in the private mode could be found in the "`history`" SQLite database. Contrary to the implementation in the usual mode, the "visit_count" field was set to 0 and the "hidden" field set to 1 (see Fig. 3). Hence, a search for "visit_count=0 and hidden=1" would conveniently disclose websites bookmarked in the private mode with detailed timestamps, which in turn would indicate precisely when private browsing was used.

The case of Safari is the worst. 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 – more specifically, in the `PageURL` table of the `WebpageIcons.db` database file. We have reported this bug to Apple, and learned that it would be fixed in a newer version of Safari[7].

4.4. Extensions

The effects of browser extensions on private browsing were studied by Aggarwal *et al.* in 2010 [1]. Their study was mainly focused on Firefox extensions and the adopted approach was to survey the most popular 40 Firefox extensions and analyse their behavior. We extended their work in two aspects. First, instead of analysing third party extensions (and speculating their intentions), we wrote our own extensions; this allows us to more directly capture the impact of extensions on private browsing. Furthermore, the developed extensions cover not just Firefox, but also Chrome, Safari and IE. (Note that in 2010 when the paper [1] was published, Safari and IE did not yet support extensions.)

[7]After we filed a bug report (#14685058) about this flaw to Apple, we received a reply from Apple on 12 August, 2013: "Engineering has determined that this is not to be fixed". We further requested Apple to justify their decision. On 18 August, Apple replied again: "After much deliberation, engineering has removed this feature.". As of February 2014, we find the latest version of Safari (5.1.7) has fixed the reported issue; we can no longer find private browsing records in the history file.

Chrome Extension. Chrome extensions are designed to provide additional functionality and augment default browsing behaviour. Chrome extensions are based on web technologies, with Javascript used for providing functionality, and user interfaces based on HTML and CSS. [27]. For the demonstration purpose, we wrote a Chrome extension, called Incognito Inspector [33]. Once this extension was enabled in the Chrome private mode, it was able to record detailed user activities for the duration of a private browsing session, including when the tabs were opened and closed, which web pages were visited and at which time, how the user flipped between tabs and windows, etc (see Fig. 4). These details can be sent to a remote server in real time.

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.

Safari Extension. To illustrate the risks posed by extensions during private browsing in Safari, we developed a similar inspector extension [33] for Safari, which is able to record details of the user's activities within a private session. In Safari, extensions are enabled by default in the private mode. Figure 5 shows the Safari extension listing all websites that were visited in the private mode.

Firefox extension. Similarly, an inspector extension [33] was developed for Firefox to record detailed user activities in a private session. As with Safari, the Firefox extensions are enabled in the private mode by default. As shown in Figure 6, the extension displays all the visited website during the private navigation.

Internet Explorer Unlike other browsers, the development of IE extensions requires using .Net Framework based languages like C#. The extension we developed [33] uses the Browser Helper Object (BHO) class, which provides a rich set of APIs to extend the functionality of IE. In the usual mode, the extension can obtain the URL and the content of the HTML page, which is represented in a DOM (Document Object Model) structure. Like in Chrome, extensions are disabled by default in the private mode. They can be enabled by a manual operation. However, after we manually enabled extensions in the private mode, we found an enabled extension had only restricted privilege: in particular, it could no longer invoke the BHO class. Hence, our attack did not work on IE.

4.5. Cross-mode interference

In Chrome, extensions are disabled by default in the private mode. But extensions in the usual mode are enabled. Since Chrome allows the usual and private modes to run in parallel, this provides the attacker an opportunity to exploit the 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 (Figure 7). Accordingly, we developed an extension using the standard Chrome extension APIs [31], installed it in the browser, and left it disabled in the private mode.

The attack works as follows. In the usual mode, the extension 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. But, the precise lack of response provides 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, when those tabs are opened and when are they closed. Finally, the extension sends all the collected data to a remote website (under our control).

In fact, Chrome also provides experimental APIs (which are enabled in the `chrome://flags` interface) to further enforce the extension's functionality [32]. 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 8 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.

We do not test the cross-mode interference for Firefox and Safari, because extensions are enabled by default in the private mode for both browsers. Hence, an attacker is better off to steal sensitive information about private browsing directly through the extension rather than by exploiting the cross-mode interference.

5. Remote attacks

Based on the threat model explained in Section 3, the fact that a person used or is using the private mode is considered a privacy feature by itself. However, existing implementations of private browsing in several browsers allow a remote website to easily tell if the user is using the private mode. In this section, we will explain two attacks, based on checking the color of hyperlinks and the side-channel timing information of writing cookies.

5.1. Hyperlink attack

Normally a web browser displays an unvisited hyperlink in blue and changes it to purple after the user clicks the link or visits the URL. This is a conventional technique adopted by all browsers to distinguish visited links from unvisited ones, hence improving the user's browsing experience [3].

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. In several browsers (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, it is worth noting that defence against web tracking is not within the threat model of private browsing.)

These deviations clearly degrade the user experience; furthermore, they 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. As a result, the user may be presented with different web pages and advertisements (that may contain more adult content). 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 [1] and we find it still exists in the latest versions of the browsers. However, the case of IE is a bit different. In the latest version of IE, 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.

5.2. Timing attack

Timing attacks used to compromise the privacy of Internet users are presented in the literature [5]. In this section, we describe a novel client-side timing attack based on leveraging differences in time taken to write a large number of cookies between usual and private modes.

Implementation. 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 [30] was used to automate testing enabling large-scale experimentation.

Evaluation. We collected extensive timing measurements for the usual and private modes (100 samples per mode per browser) as training data (see Figure 9). Normal distributions are fitted to each training set.

We first assume the null hypothesis, namely: there is no significant difference in timing between writing cookies in the private and usual modes. Obviously, if statistical tests reject the null hypothesis, they will naturally support the alternative. The alternative would imply a weakness in the private browsing implementation as the remote attacker would be able to measure the difference and tell if the private mode is switched on. From Figure 9, one can intuitively tell that there is a significant timing difference between the private and usual modes for Chrome, Firefox and Safari. We now confirm that intuition in more concrete terms by employing the standard z-test [7]. Here the z-test is used instead of the Student's t-test mainly because the sample size in our experiments is relatively big (i.e., > 30). The z-test [7] is defined in Equation 1, where $\overline{x}$ is the sample mean, μ is the hypothesised mean, σ is the population standard deviation[8], and n is the sample size.

$$ztest(\mu, X) = \frac{\overline{x} - \mu}{\sigma/\sqrt{n}} \tag{1}$$

We calculate p-values representing the probability that a given observation belongs to usual or private browsing mode, as follows.

[8]Given our large sample size, we approximate the population standard deviation as the sample standard deviation.

Browser	Equal Error Rate (EER)	Threshold (t)	Insecurity ($\|0.5 - EER\|$)
Google Chrome	1%	0	0.49
Mozilla Firefox	9%	0	0.41
Internet Explorer	63%	-0.0002	0.13
Apple Safari	1%	0.0055	0.49

Table 5: Equal Error Rates for detecting the browsing mode. The level of insecurity is expressed as $|0.5 - EER|$: the bigger the value is, the less secure.

$$p(x) = F(x|\mu = 0, \sigma = 1) = \frac{1}{\sigma\sqrt{2\pi}} \int_{|x|}^{\infty} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{2}$$

We then employ a simple threshold policy where any sample s from browser b whose p-value for private mode is greater than its p-value for usual mode by a threshold interval t is categorised as belonging to private mode. Our approach is outlined in Equation 3, where b_{usual} and $b_{private}$ are training sets for browser b in usual and private modes respectively.

$$f(s, b, t) = \begin{cases} Private & \text{if } p(ztest(s, b_{private})) > p(ztest(s, b_{usual})) + t \\ Usual & \text{else} \end{cases} \tag{3}$$

We collected further 100 timing measurements for each browser for each mode in order to evaluate the effectivenss of our approach. There are two types of errors in the evaluation. One is the False Acceptance Rate (FAR), that is the rate of a usual browsing session being characterised as private. The other is False Rejection Rate (FRR), that is the rate of a private browsing session being characterised as in 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 attacker detects the private/usual mode no better than tossing a coin. However, as shown in Table 5, with the exception of IE, a remote attacker is able to correctly identify the browsing mode with high accuracy.

6. Countermeasures

We divide the causes of attacks into two categories: internal elements that only concern the internal design of a browser, and external elements that involve external interactions with the rest of the system.

6.1. Internal elements

Adding bookmarks. Our research shows that adding bookmarks in the private mode can cause privacy violation in several popular browsers with a varying degree of severity. One countermeasure may be to disallow adding any bookmarks in the private mode (as implemented in an earlier version of Chrome 20.0). However, this would cause inconvenience to users; there exist valid use cases in which a user may want to add a bookmark while in the private mode. In the existing implementation of private browsing in popular browsers, users are completely uninformed about the potential information leakage associated with adding bookmarks in the private mode. Hence, browser vendors should at least add a mechanism to inform the user about the potential privacy violation before a bookmark is added, e.g., through the display of a warning dialogue.

SQLite database. We find SQLite temporary files usually remain on disk in the event of the program crash or power off. In Chrome and Firefox, the traces left on disk allow an attacker to deduce the occurrence of private browsing in the past. In Safari, the traces reveal the websites that were visited in the private session. The browser crash is an edge case as it happens infrequently in the normal usage. However, for a secure implementation, all edge cases should be considered. For example, the obvious flaw of the Safari browser retaining private navigation history in the event of program crash should have been easily detected in the in-house testing if Apple developers had included all edge cases into the test framework.

Extension. We have shown that browser extensions can easily reveal the user's activities in the private mode. So far only Chrome and IE disable extension by default in the private mode. We recommend other browsers to do the same. Furthermore, when an extension is enabled in the private mode, the browser should treat it differently as running in the usual mode, and restrict its privileged access to extension APIs accordingly. Recently, researchers have made progress in developing an automated tool to verify the extensions' compliance with the private browsing mode [8]. Such tool

can prove useful if it is integrated into the web browser, so that only the compliant extensions can be installed.

Cross-mode interference. Chrome allows the usual and private modes to run in parallel. However, our research shows that an attacker is able to obtain user-sensitive information in the private mode by exploiting cross-mode interference. A safer approach is to run the browser in a single mode only.

Hyperlinks. Our research shows that for all browsers, a remote attacker can easily tell if a user is in the private mode by checking the hyperlink colours. This is because the way a hyperlink is coloured in the private mode deviates from that in the usual mode. Such deviations clearly degrade the user experience in the private navigation mode. Furthermore, they do not seem to provide any real advantage in making private browsing more secure. On the contrary, based on such deviations, a remote attacker is able to easily tell a private mode apart from a usual mode. We therefore recommend browser vendors to remove these deviations. The mechanism for displaying hyperlinks should be same in the private mode as in the usual mode. The only difference should be that in the private mode the information about visited websites is not persistently saved on disk once the session is closed.

Cookie timing. For several browsers, we observed a noticeable difference in the time spent to write cookies between the usual and private modes. This difference can be exploited by a remote attacker to tell if the user is in the private mode. Usually, to mitigate the effects of timing attacks, one commonly adopted countermeasure is to insert random delays to reduce the timing difference (e.g, see [9, 10]). The same countermeasure may be applied here, but we believe it is important to first understand why such differences exist. Unfortunately, with our current state of knowledge, we cannot fully explain what caused the timing differences. (One may say writing cookies should be quicker in the private mode as it is a memory-only operation and no data is written to the disk. However, that intuitive explanation does not seem to apply to Chrome.) More investigation in this aspect is needed.

6.2. External elements

DNS. DNS cache retains the websites visited regardless if they are from the usual or private mode. Some researchers have already made progress in developing extensions to clean the DNS cache [21, 22]. It should be possible for browser vendors to integrate such solutions into the browser, so that

records in the DNS cache related to the private browsing can be sanitised in an automated manner.

Memory. We have found artifacts in memory left from the private navigation for all browsers under test. It seems browser vendors have all failed to address this vulnerability, despite that it was reported in 2011 [14]. Cleaning all individual data from RAM is a computationally expensive procedure, which may explain a lack of progress in this area. We are unable to suggest any solution at this stage; more research is needed in this area.

File timestamps. In the latest versions of Firefox and IE, we did not observe any timestamp change of files under the profile directory after a private browsing session. Both browsers chose to read databases into memory and do not write any new record to the file on the disk. This is a more secure approach, which we recommend other browsers to follow.

7. Conclusion

We have presented a range of vulnerabilities in the existing implementations of private browsing across four popular web browsers. The revealed problems highlight the complexity of the subject and call for more attention from the security community. They also show that ad-hoc efforts to implement private browsing – as currently adopted by browser vendors – can easily lead to important security considerations being ignored. A more systematic approach to design, implement and test the private browsing feature is needed – in particular, an appropriate threat model should be established so to provide a yardstick to measure security; edge cases that happen infrequently (e.g., application crash, user adding bookmarks) should be included in the test framework; extensions should be disabled in the private mode by default (otherwise, there must be some automated tool, like the one developed in [8], to strictly ensure the extensions' compliance with the private model); the browser should be run in a single mode to avoid any cross-mode interference; deviations between the private and usual modes should be minimized as much as possible so that a remote attacker cannot reliably tell the two modes apart.

Acknowledgement

The Firefox inspector extension was initially written by a previous MSc student, Nicoleta Nicolaou, in 2011 in the School of Computing Science,

Newcastle University. The initial idea of the remote attack based on writing cookies was inspired by a freely available on-line manuscript (`http://mocktest.net/paper.pdf`).

References

[1] G. Aggarwal, E. Burzstein, C. Jackson, D. Boneh, "An analysis of private browsing modes in modern browsers," the 19th USENIX Symphosium on Security, 2010.

[2] C. Boyd, P. Forster, "Time and date issues in forensic computing-a case study," *Digital Investigation*, Vol. 1, No. 1, pp. 18–23, 2004.

[3] 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.

[4] P. Eckersley, "How unique is your web browser?", Available at `https://panopticlick.eff.org/browser-uniqueness.pdf` (Accessed: April 2013)

[5] W.E. Felten, M.A. Schneider, "Timing attacks on Web privacy," the 7th ACM conference on Computer and Communications Security (CCS), 2000.

[6] 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.

[7] E. Kreyszig, "Introductory Mathematical Statistics," John Wiley & Sons Inc, 1970.

[8] B.S. Lerner, L. Elberty, N. Poole, S. Krishnamurthi, "Verifying Web Browser Extensions' Compliance with Private-Browsing Mode," Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS), 2013.

[9] P.C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '96), pp. 104-113, 1996.

[10] J. Nadhem, A. Fardan, K.. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols," Proceedings of 2013 IEEE Symposium on Security and Privacy (S&P), pp. 526-540, 2013.

[11] M.T. Pereira, "Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records," *Digital Investigation*, Vol. 5, No. 3, pp. 93–103, 2009.

[12] K. Satvat, M. Forshaw, F. Hao, E. Toreini, "On The Privacy of Private Browsing - A Forensic Approach (short paper)", Proceedings of the 8th International Workshop on Data Privacy Management (DPM'13), 2013.

[13] 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

[14] 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.

[15] Chrome private browsing mode: `https://support.google.com/chrome/bin/answer.py?hl=en&answer=95464&p=cpn_incognito` (Accessed: April 2013)

[16] Mozilla Firefox private browsing mode: `http://support.mozilla.org/en-US/kb/private-browsing-browse-web-without-saving-info` (Accessed: April 2013)

[17] Safari private browsing mode: `http://support.apple.com/kb/PH5000` (Accessed: April 2013)

[18] Internet Explorer private browsing mode: `http://windows.microsoft.com/en-us/windows-vista/what-is-inprivate-browsing` (Accessed: April 2013)

[19] Most popular web browsers: `http://www.w3schools.com/browsers/browsers_stats.asp` (Accessed: April 2013)

[20] VMware Player Version 4.0.0: `http://www.vmware.com/products/player/` (Accessed: April 2013)

[21] Click & Clean: https://chrome.google.com/webstore/detail/ghgabhipcejejjmhhchfonmamedcbeod?utm_source=chrome-ntp-icon (Accessed: April 2013)

[22] Clear DNS Cache: https://addons.mozilla.org/en-us/firefox/addon/clear-dns-cache/ (Accessed: April 2013)

[23] WinHex, Computer Forensics & Data Recovery Software:. http://www.winhex.com/winhex/ (Accessed: April 2013)

[24] Index.dat Analyzer: http://www.systenance.com/indexdat.php (Accessed: April 2013)

[25] SQLite Manager: https://addons.mozilla.org/en-us/firefox/addon/sqlite-manager/ (Accessed: April 2013)

[26] SQLite Database Browser: http://sqlitebrowser.sourceforge.net/ (Accessed: April 2013)

[27] Google Chrome Extensions: http://code.google.com/chrome/extensions/overview.html (Accessed: April 2013)

[28] Safari Extensions: https://developer.apple.com/library/safari/#documentation/Tools/Conceptual/SafariExtensionGuide/Introduction/Introduction.html (Accessed: April 2013)

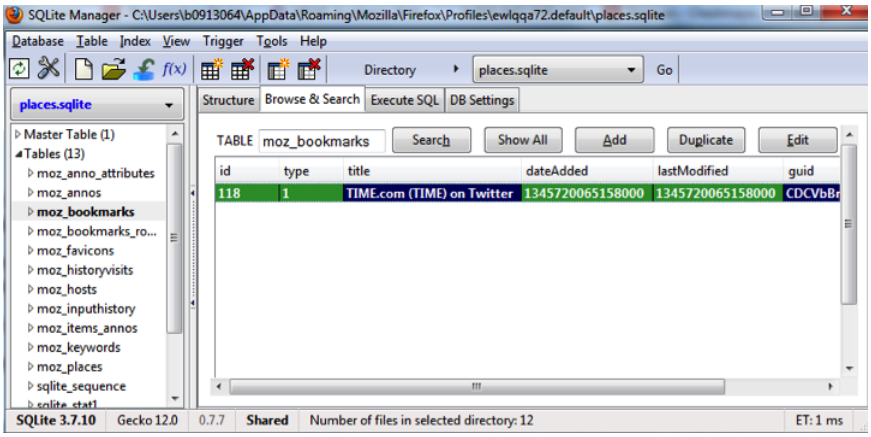
[29] The official website for the TOR project: https://www.torproject.org/ (Accessed: April 2013)

[30] Selenium: http://seleniumhq.org/ (Accessed: April 2013)

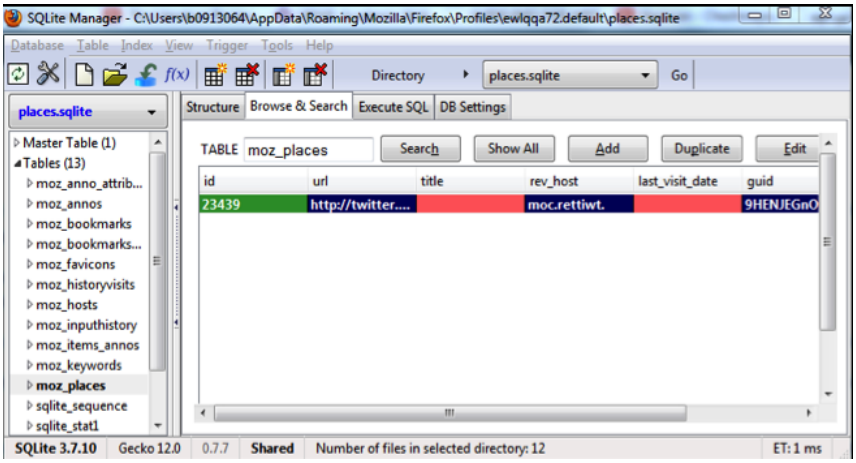
[31] Standard Chrome extension API: http://developer.chrome.com/extensions/ (Accessed: April 2013)

[32] Experimental Chrome extension API: http://developer.chrome.com/extensions/experimental.html (Accessed: April 2013)

[33] Open source code of the software tools developed during the research work of this paper: http://homepages.cs.ncl.ac.uk/m.j.forshaw1/privatebrowsing/



(a) Bookmark record in `moz_bookmarks`



(b) Record in `moz_places` with deleted last_visit_date

Figure 2: Adding a bookmark in the Firefox private mode

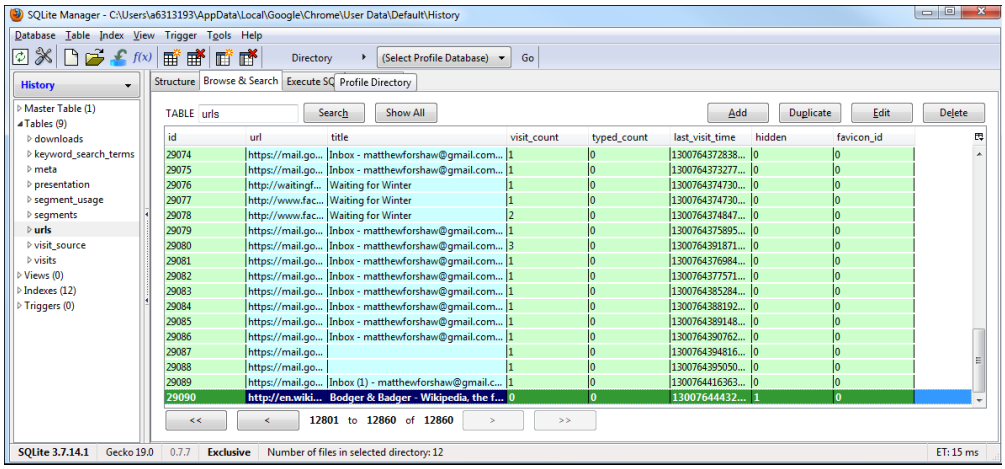


Figure 3: Chrome history SQLite. The highlighted record corresponds to a bookmark added in the private mode. The record persists in the database with the "visit_count" field merely set to 0 and "hidden" set to 1.

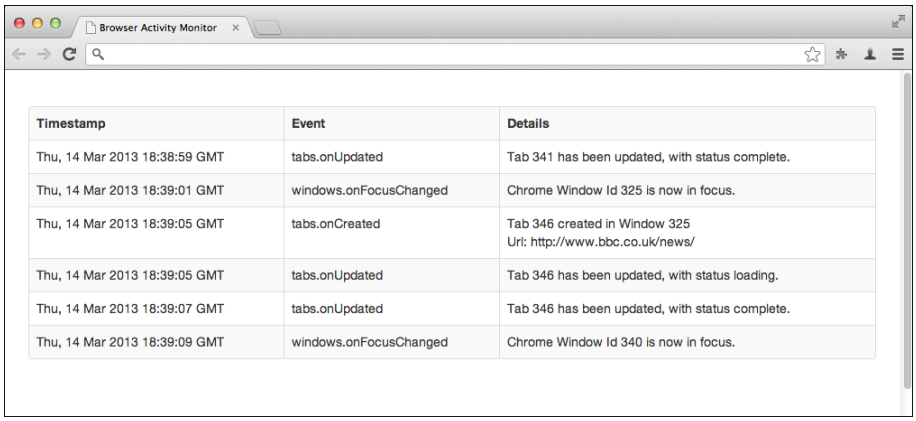


Figure 4: Chrome extension recording user activities in private browsing session

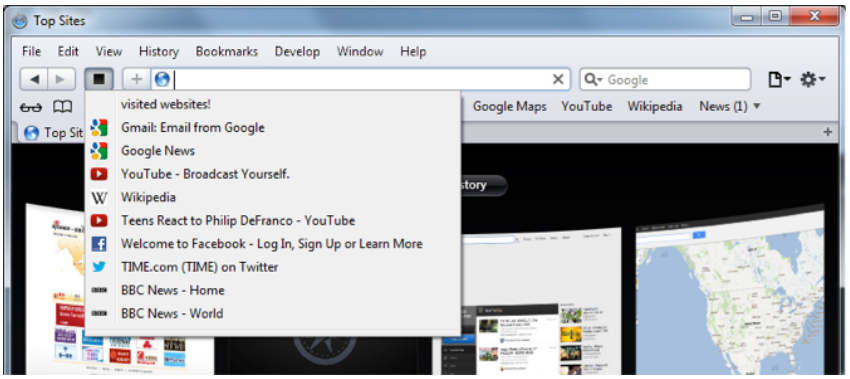


Figure 5: Safari extension showing the retrieved history of visited website during a private browsing session in a drop-down menu

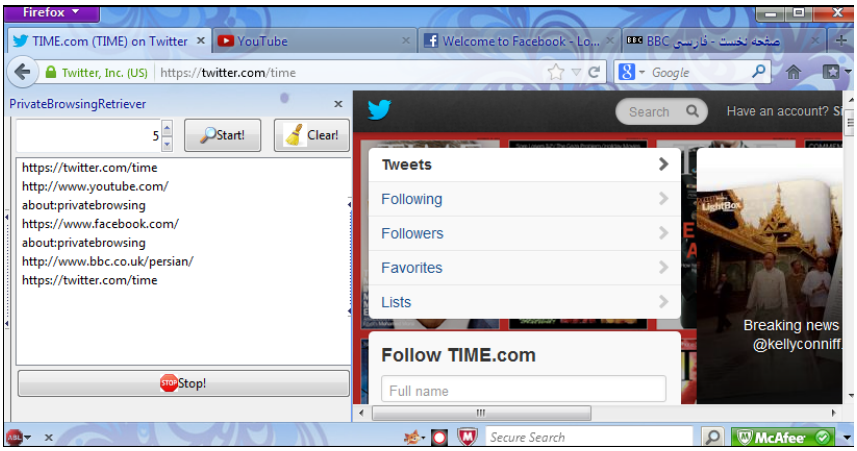


Figure 6: Firefox extension recording visited websites in the private mode

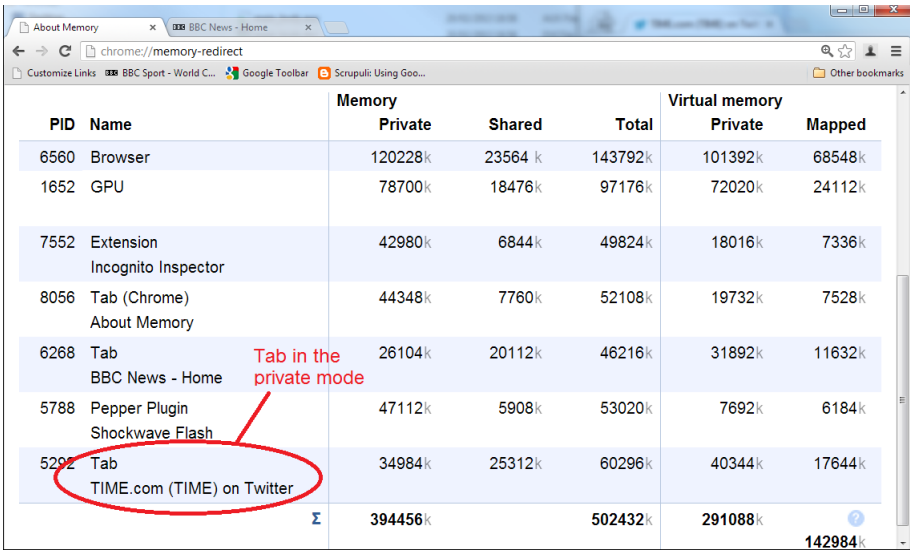


Figure 7: Chrome://memory interface

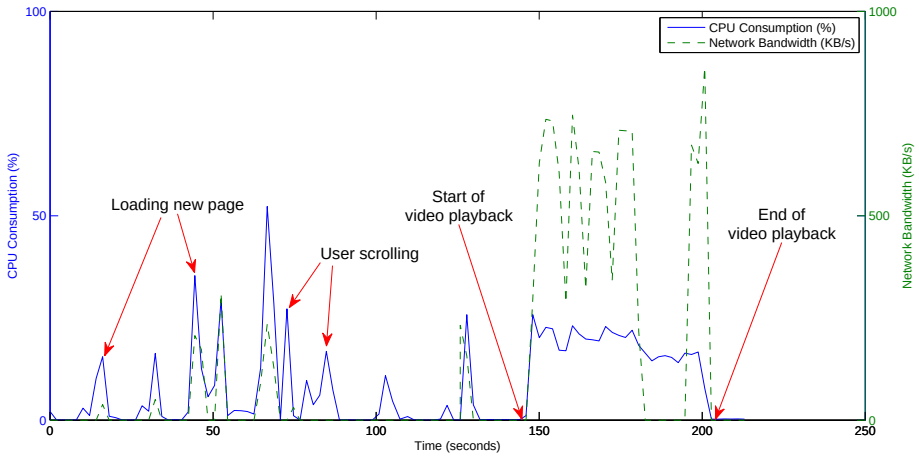


Figure 8: Profiling the user activities in the private mode based on the CPU and network usage

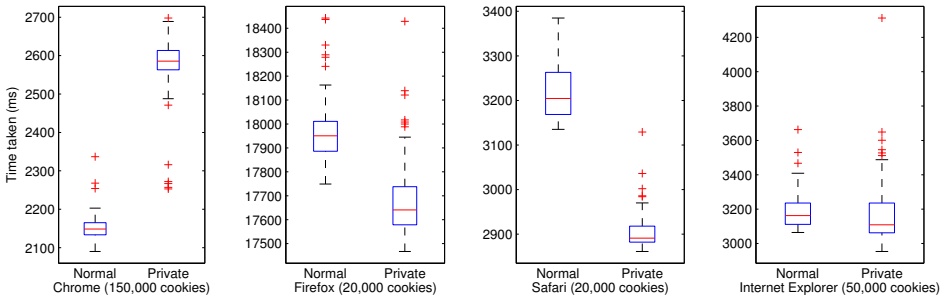


Figure 9: Box plots representing timing data collected for browsers under test.