An Analysis of Private Browsing Modes in Modern Browsers

Gaurav Aggarwal   Elie Bursztein   Collin Jackson   Dan Boneh
Stanford University   *CMU*   *Stanford University*

Abstract

We study the security and privacy of private browsing modes recently added to all major browsers. We first propose a clean definition of the goals of private browsing and survey its implementation in different browsers. We conduct a measurement study to determine how often it is used and on what categories of sites. Our results suggest that private browsing is used differently from how it is marketed. We then describe an automated technique for testing the security of private browsing modes and report on a few weaknesses found in the Firefox browser. Finally, we show that many popular browser extensions and plugins undermine the security of private browsing. We propose and experiment with a workable policy that lets users safely run extensions in private browsing mode.

1   Introduction

The four major browsers (Internet Explorer, Firefox, Chrome and Safari) recently added private browsing modes to their user interfaces. Loosely speaking, these modes have two goals. First and foremost, sites visited while browsing in private mode should leave no trace on the user's computer. A family member who examines the browser's history should find no evidence of sites visited in private mode. More precisely, a *local attacker* who takes control of the machine at time T should learn no information about private browsing actions prior to time T. Second, users may want to hide their identity from web sites they visit by, for example, making it difficult for web sites to link the user's activities in private mode to the user's activities in public mode. We refer to this as privacy from a *web attacker*.

While all major browsers support private browsing, there is a great deal of inconsistency in the type of privacy provided by the different browsers. Firefox and Chrome, for example, attempt to protect against a local attacker and take some steps to protect against a web attacker, while Safari only protects against a local attacker.

Even within a single browser there are inconsistencies. For example, in Firefox 3.6, cookies set in public mode are not available to the web site while the browser is in private mode. However, passwords and SSL client certificates stored in public mode are available while in private mode. Since web sites can use the password manager as a crude cookie mechanism, the password policy is inconsistent with the cookie policy.

Browser plug-ins and extensions add considerable complexity to private browsing. Even if a browser adequately implements private browsing, an extension can completely undermine its privacy guarantees. In Section 6.1 we show that many widely used extensions undermine the goals of private browsing. For this reason, Google Chrome disables all extensions while in private mode, negatively impacting the user experience. Firefox, in contrast, allows extensions to run in private mode, favoring usability over security.

Our contribution. The inconsistencies between the goals and implementation of private browsing suggests that there is considerable room for research on private browsing. We present the following contributions.

- **Threat model.** We begin with a clear definition of the goals of private browsing. Our model has two somewhat orthogonal goals: security against a local attacker (the primary goal of private browsing) and security against a web attacker. We show that correctly implementing private browsing can be non-trivial and in fact all browsers fail in one way or another. We then survey how private browsing is implemented in the four major browsers, highlighting the quirks and differences between the browsers.

- **Experiment.** We conduct an experiment to test how private browsing is used. Our study is based on a technique we discovered to remotely test if a browser is currently in private browsing mode. Using this technique we post ads on ad-networks and

determine how often private mode is used. Using ad targeting by the ad-network we target different categories of sites, enabling us to correlate the use of private browsing with the type of site being visited. We find it to be more popular at adult sites and less popular at gift sites, suggesting that its primary purpose may not be shopping for "surprise gifts." We quantify our findings in Section 4.

- **Tools.** We describe an automated technique for identifying failures in private browsing implementations and use it to discover a few weaknesses in the Firefox browser.

- **Browser extensions.** We propose an improvement to existing approaches to extensions in private browsing mode, preventing extensions from unintentionally leaving traces of the private activity on disk. We implement our proposal as a Firefox extension that imposes this policy on other extensions.

Organization. Section 2 presents a threat model for private browsing. Section 3 surveys private browsing mode in modern browsers. Section 4 describes our experimental measurement of private browsing usage. Section 5 describes the weaknesses we found in existing private browsing implementations. Section 6 addresses the challenges introduced by extensions and plug-ins. Section 7 describes additional related work. Section 8 concludes.

2 Private browsing: goal and threat model

In defining the goals and threat model for private browsing, we consider two types of attackers: an attacker who controls the user's machine (a local attacker) and an attacker who controls web sites that the user visits (a web attacker). We define security against each attacker in turn. In what follows we refer to the user browsing the web in private browsing mode as *the user* and refer to someone trying to determine information about the user's private browsing actions as *the attacker*.

2.1 Local attacker

Stated informally, security against a local attacker means that an attacker who takes control of the machine *after* the user exits private browsing can learn nothing about the user's actions while in private browsing. We define this more precisely below.

We emphasize that the local attacker has no access to the user's machine before the user exits private browsing. Without this limitation, security against a local attacker is impossible; an attacker who has access to the user's machine before or during a private browsing session can simply install a key-logger and record all user actions. By restricting the local attacker to "after the fact" forensics, we can hope to provide security by having the browser adequately erase persistent state changes during a private browsing session.

As we will see, this requirement is far from simple. For one thing, not all state changes during private browsing should be erased at the end of a private browsing session. We draw a distinction between four types of persistent state changes:

1. Changes initiated by a web site without any user interaction. A few examples in this category include setting a cookie, adding an entry to the history file, and adding data to the browser cache.
2. Changes initiated by a web site, but requiring user interaction. Examples include generating a client certificate or adding a password to the password database.
3. Changes initiated by the user. For example, creating a bookmark or downloading a file.
4. Non-user-specific state changes, such as installing a browser patch or updating the phishing block list.

All browsers try to delete state changes in category (1) once a private browsing session is terminated. Failure to do so is treated as a private browsing violation. However, changes in the other three categories are in a gray area and different browsers treat these changes differently and often inconsistently. We discuss implementations in different browsers in the next section.

To keep our discussion general we use the term *protected actions* to refer to state changes that should be erased when leaving private browsing. It is up to each browser vendor to define the set of protected actions.

Network access. Another complication in defining private browsing is server side violations of privacy. Consider a web site that inadvertently displays to the world the last login time of every user registered at the site. Even if the user connects to the site while in private mode, the user's actions are open for anyone to see. In other words, web sites can easily violate the goals of private browsing, but this should not be considered a violation of private browsing in the browser. Since we are focusing on browser-side security, our security model defined below ignores server side violations. While browser vendors mostly ignore server side violations, one can envision a number of potential solutions:

- Much like the phishing filter, browsers can consult a block list of sites that should not be accessed while in private browsing mode.
- Alternatively, sites can provide a P3P-like policy statement saying that they will not violate private browsing. While in private mode, the browser will not connect to sites that do not display this policy.

- A non-technical solution is to post a privacy seal at web sites who comply with private browsing. Users can avoid non-compliant sites when browsing privately.

Security model. Security is usually defined using two parameters: the attacker's capabilities and the attacker's goals. A local private browsing attacker has the following capabilities:

- The attacker does nothing until the user leaves private browsing mode at which point the attacker gets complete control of the machine. This captures the fact that the attacker is limited to after-the-fact forensics.

  In this paper we focus on persistent state violations, such as those stored on disk; we ignore private state left in memory. Thus, we assume that before the attacker takes over the local machine all volatile memory is cleared (though data on disk, including the hibernation file, is fair game). Our reason for ignoring volatile memory is that erasing all of it when exiting private browsing can be quite difficult and, indeed, no browser does it. We leave it as future work to prevent privacy violations resulting from volatile memory.

- While active, the attacker cannot communicate with network elements that contain information about the user's activities while in private mode (e.g. web sites the user visited, caching proxies, etc.). This captures the fact that we are studying the implementation of browser-side privacy modes, not server-side privacy.

Given these capabilities, the attacker's goal is as follows: for a set S of HTTP requests of the attacker's choosing, determine if the browser issued any of those requests while in private browsing mode. More precisely, the attacker is asked to distinguish a private browsing session where the browser makes one of the requests in S from a private browsing session where the browser does not. If the local attacker cannot achieve this goal then we say that the browser's implementation of private browsing is secure. This will be our working definition throughout the paper. Note that since an HTTP request contains the name of the domain visited this definition implies that the attacker cannot tell if the user visited a particular site (to see why set S to be the set of all possible HTTP requests to the site in question). Moreover, even if by some auxiliary information the attacker knows that the user visited a particular site, the definition implies that the attacker cannot tell what the user did at the site. We do not formalize properties of private browsing in case the user never exits private browsing mode.

Difficulties. Browser vendors face a number of challenges in securing private browsing against a local attacker. One set of problems is due to the underlying operating system. We give two examples:

First, when connecting to a remote site the browser must resolve the site's DNS name. Operating systems often cache DNS resolutions in a local DNS cache. A local attacker can examine the DNS cache and the TTL values to learn if and when the user visited a particular site. Thus, to properly implement private browsing, the browser will need to ensure that all DNS queries while in private mode do not affect the system's DNS cache: no entries should be added or removed. A more aggressive solution, supported in Windows 2000 and later, is to flush the entire DNS resolver cache when exiting private browsing. None of the mainstream browsers currently address this issue.

Second, the operating system can swap memory pages to the swap partition on disk which can leave traces of the user's activity. To test this out we performed the following experiment on Ubuntu 9.10 running Firefox 3.5.9:

1. We rebooted the machine to clear RAM and setup and mounted a swap file (zeroed out).
2. Next, we started Firefox, switched to private browsing mode, browsed some websites and exited private mode but kept Firefox running.
3. Once the browser was in public mode, we ran a memory leak program a few times to force memory pages to be swapped out. We then ran `strings` on the swap file and searched for specific words and content of the webpages visited while in private mode.

The experiment showed that the swap file contained some URLs of visited websites, links embedded in those pages and sometimes even the text from a page – enough information to learn about the user's activity in private browsing.

This experiment shows that a full implementation of private browsing will need to prevent browser memory pages from being swapped out. None of the mainstream browsers currently do this.

Non-solutions. At first glance it may seem that security against a local attacker can be achieved using virtual machine snapshots. The browser runs on top of a virtual machine monitor (VMM) that takes a snapshot of the browser state whenever the browser enters private browsing mode. When the user exits private browsing the VMM restores the browser, and possibly other OS data, to its state prior to entering private mode. This architecture is unacceptable to browser vendors for several reasons: first, a browser security update installed during private browsing will be undone when exiting private mode;

second, documents manually downloaded and saved to the file system during private mode will be lost when exiting private mode, causing user frustration; and third, manual tweaks to browser settings (e.g. the homepage URL, visibility status of toolbars, and bookmarks) will revert to their earlier settings when exiting private mode. For all these reasons and others, a complete restore of the browser to its state when entering private mode is not the desired behavior. Only browser state that reveals information on sites visited should be deleted.

User profiles provide a lightweight approach to implementing the VM snapshot method described above. User profiles store all browser state associated with a particular user. Firefox, for example, supports multiple user profiles and the user can choose a profile when starting the browser. The browser can make a backup of the user's profile when entering private mode and restore the profile to its earlier state when exiting private mode. This mechanism, however, suffers from all the problems mentioned above.

Rather than a snapshot-and-restore approach, all four major browsers take the approach of not recording certain data while in private mode (e.g. the history file is not updated) and deleting other data when exiting private mode (e.g. cookies). As we will see, some data that should be deleted is not.

2.2 Web attacker

Beyond a local attacker, browsers attempt to provide some privacy from web sites. Here the attacker does not control the user's machine, but has control over some visited sites. There are three orthogonal goals that browsers try to achieve to some degree:

- **Goal 1:** A web site cannot link a user visiting in private mode to the same user visiting in public mode. Firefox, Chrome, and IE implement this (partially) by making cookies set in public mode unavailable while in private mode, among other things discussed in the next section. Interestingly, Safari ignores the web attacker model and makes public cookies available in private browsing.

- **Goal 2:** A web site cannot link a user in one private session to the same user in another private session. More precisely, consider the following sequence of visits at a particular site: the user visits in public mode, then enters private mode and visits again, exits private mode and visits again, re-activates private mode and visits again. The site should not be able to link the two private sessions to the same user. Browsers implement this (partially) by deleting cookies set while in private mode, as well as other restrictions discussed in the next section.

- **Goal 3:** A web site should not be able to determine whether the browser is currently in private browsing mode. While this is a desirable goal, all browsers fail to satisfy this; we describe a generic attack in Section 4.

Goals (1) and (2) are quite difficult to achieve. At the very least, the browser's IP address can help web sites link users across private browsing boundaries. Even if we ignore IP addresses, a web site can use various browser features to fingerprint a particular browser and track that browser across privacy boundaries. Mayer [14] describes a number of such features, such as screen resolution, installed plug-ins, timezone, and installed fonts, all available through standard JavaScript objects. The Electronic Frontier Foundation recently built a web site called Panopticlick [6] to demonstrate that most browsers can be uniquely fingerprinted. Their browser fingerprinting technology completely breaks private browsing goals (1) and (2) in all browsers.

Torbutton [29] — a Tor client implemented as a Firefox extension — puts considerable effort into achieving goals (1) and (2). It hides the client's IP address using the Tor network and takes steps to prevent browser fingerprinting. This functionality is achieved at a considerable performance and convenience cost to the user.

3 A survey of private browsing in modern browsers

All four majors browsers (Internet Explorer 8, Firefox 3.5, Safari 4, and Google Chrome 5) implement a private browsing mode. This feature is called *InPrivate* in Internet Explorer, *Private Browsing* in Firefox and Safari, and *Incognito* in Chrome.

User interface. Figure 1 shows the user interface associated with these modes in each of the browsers. Chrome and Internet Explorer have obvious chrome indicators that the browser is currently in private browsing mode, while the Firefox indicator is more subtle and Safari only displays the mode in a pull down menu. The difference in visual indicators has to do with shoulder surfing: can a casual observer tell if the user is currently browsing privately? Safari takes this issue seriously and provides no visual indicator in the browser chrome, while other browsers do provide a persistent indicator. We expect that hiding the visual indicator causes users who turn on private browsing to forget to turn it off. We give some evidence of this phenomenon in Section 4 where we show that the percentage of users who browse the web in private mode is greater in browsers with subtle visual indicators.

Another fundamental difference between the browsers is how they start private browsing. IE and Chrome spawn

a new window while keeping old windows open, thus allowing the user to simultaneously use the two modes. Firefox does not allow mixing the two modes. When entering private mode it hides all open windows and spawns a new private browsing window. Unhiding public windows does nothing since all tabs in these windows are frozen while browsing privately. Safari simply switches the current window to private mode and leaves all tabs unchanged.

Internal behavior. To document how the four implementations differ, we tested a variety of browser features that maintain state and observed the browsers' handling of each feature in conjunction with private browsing mode. The results, conducted on Windows 7 using a default browser settings, are summarized in Tables 1, 2 and 3.

Table 1 studies the types of data set in public mode that are available in private mode. Some browsers block data set in public mode to make it harder for web sites to link the private user to the pubic user (addressing the web attacker model). The Safari column in Table 1 shows that Safari ignores the web attacker model altogether and makes all public data available in private mode except for the web cache. Firefox, IE, and Chrome block access to some public data while allowing access to other data. All three make public history, bookmarks and passwords available in private browsing, but block public cookies and HTML5 local storage. Firefox allows SSL client certs set in public mode to be used in private mode, thus enabling a web site to link the private session to the user's public session. Hence, Firefox's client cert policy is inconsistent with its cookie policy. IE differs from the other three browsers in the policy for form field autocompletion; it allows using data from public mode.

Table 2 studies the type of data set in private mode that persists after the user leaves private mode. A local attacker can use data that persists to learn user actions in private mode. All four browsers block cookies, history, and HTML5 local storage from propagating to public mode, but persist bookmarks and downloads. Note that all browsers other than Firefox persist server self-signed certificates approved by the user while in private browsing mode. Lewis [35] recently pointed that Chrome 5.0.375.38 persisted the window zoom level for URLs across incognito sessions; this issue has been fixed as of Chrome 5.0.375.53.

Table 3 studies data that is entered in private mode and persists during that same private mode session. While in private mode, Firefox writes nothing to the history database and similarly no new passwords and no search terms are saved. However, cookies are stored in memory while in private mode and erased when the user exists private mode. These cookies are not written to persistent storage to ensure that if the browser crashes in

private mode this data will be erased. The browser's web cache is handled similarly. We note that among the four browsers, only Firefox stores the list of downloaded items in private mode. This list is cleared on leaving private mode.

3.1 A few initial privacy violation examples

In Section 5.1 we describe tests of private browsing mode that revealed several browser attributes that persist after a private browsing session is terminated. Web sites that use any of these features leave tracks on the user's machine that will enable a local attacker to determine the user's activities in private mode. We give a few examples below.

Custom Handler Protocol. Firefox implements an HTML 5 feature called *custom protocol handlers* (CPH) that enables a web site to define custom protocols, namely URLs of the form `xyz://site/path` where `xyz` is a custom protocol name. We discovered that custom protocol handlers defined while the browser is in private mode persist after private browsing ends. Consequently, sites that use this feature will leak the fact that the user visited these sites to a local attacker.

Client Certificate. IE, Firefox, and Safari support SSL client certificates. A web site can, using JavaScript, instruct the browser to generate an SSL client public/private key pair. We discovered that all these browsers retain the generated key pair even after private browsing ends. Again, if the user visits a site that generates an SSL client key pair, the resulting keys will leak the site's identity to the local attacker. When Internet Explorer and Safari encounter a self-signed certificate they store it in a Microsoft certificate vault. We discovered that entries added to the vault while in private mode remain in the vault when the private session ends. Hence, if the user visits a site that is using a self signed certificate, that information will be available to the local attacker even after the user leaves private mode.

SMB Query. Since Internet Explorer shares some underlying components with *Window Explorer* it understands SMB naming conventions such as `\\host\ mydir\myfile` and allows the user to browse files and directories. This feature has been used before to steal user data [16]. Here we point out that SMB can also be used to undo some of the benefits of private browsing mode. Consider the following code :
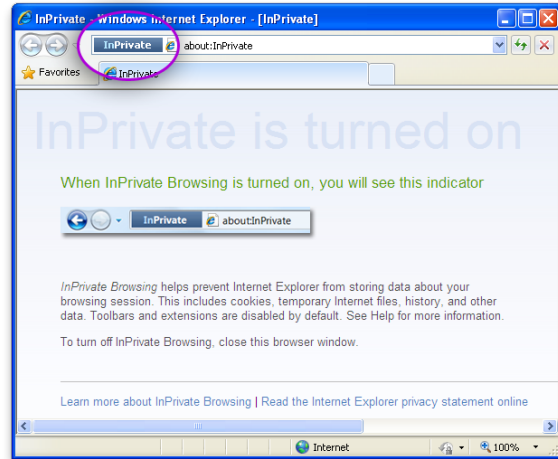
```
<img src="\\[WEB SERVER IP]\image.jpg">
```
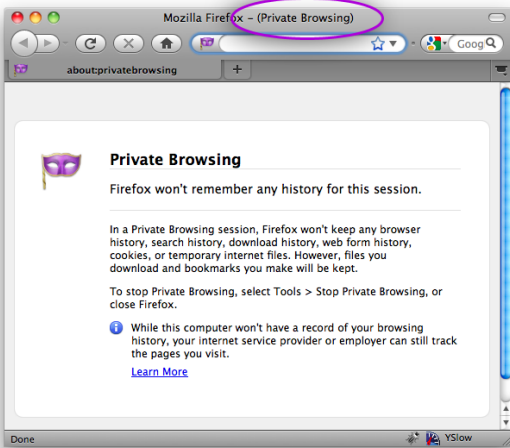
When IE renders this tag, it initiates an SMB request to the web server whose IP is specified in the image source. Part of the SMB request is an NTLM authentication that works as follows: first an anonymous connection is tried
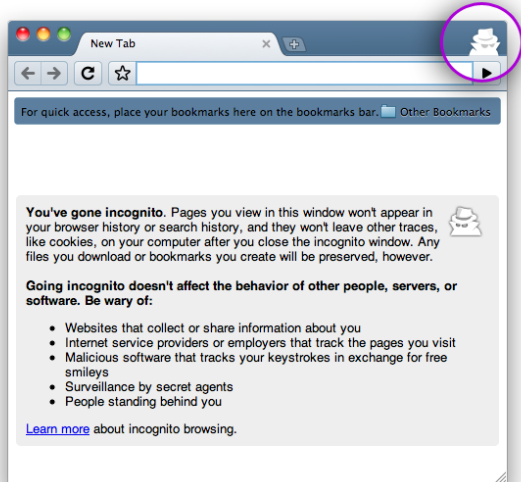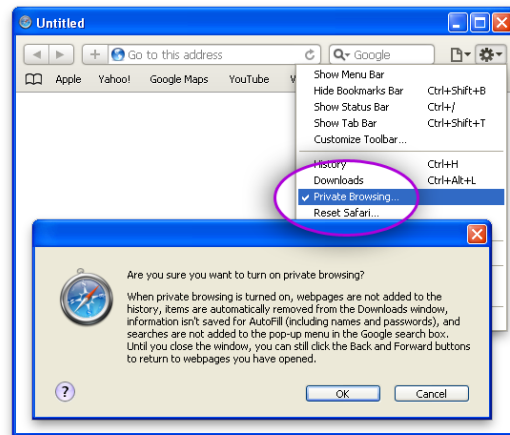
(a) Google Chrome 4

(b) Internet Explorer 8

(c) Firefox 3.6

(d) Safari 4

Figure 1: Private browsing indicators in major browsers

	FF	Safari	Chrome	IE
History	no	yes	no	no
Cookies	no	yes	no	no
HTML5 local storage	no	yes	no	no
Bookmarks	yes	yes	yes	yes
Password database	yes	yes	yes	yes
Form autocompletion	yes	yes	yes	no
User approved SSL self-signed cert	yes	yes	yes	yes
Downloaded items list	no	yes	yes	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	yes	yes	yes	yes
Browser's web cache	no	no	no	no
Client certs	yes	yes	yes	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	yes	n/a

Table 1: Is the state set in earlier public mode(s) accessible in private mode?

	FF	Safari	Chrome	IE
History	no	no	no	no
Cookies	no	no	no	no
HTML5 Local storage	no	no	no	no
Bookmarks	yes	yes	yes	yes
Password database	no	no	no	no
Form autocompletion	no	no	no	no
User approved SSL self-signed cert	no	yes	yes	yes
Downloaded items list	no	no	no	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	no	no	no	no
Browser's web cache	no	no	no	no
Client certs	yes	n/a	n/a	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	no	n/a

Table 2: Is the state set in earlier private mode(s) accessible in public mode?

	FF	Safari	Chrome	IE
History	no	no	no	no
Cookies	yes	yes	yes	yes
HTML5 Local storage	yes	yes	yes	yes
Bookmarks	yes	yes	yes	yes
Password database	no	no	no	no
Form autocompletion	no	no	no	no
User approved SSL self-signed cert	yes	yes	yes	yes
Downloaded items list	yes	no	no	n/a
Downloaded items	yes	yes	yes	yes
Search box search terms	no	no	no	no
Browser's web cache	yes	yes	yes	yes
Client certs	yes	n/a	n/a	yes
Custom protocol handlers	yes	n/a	n/a	n/a
Per-site zoom level	no	n/a	yes	n/a

Table 3: Is the state set in private mode at some point accessible later in the same session?

and if it fails IE starts a challenge-response negotiation. IE also sends to the server *Windows username, Windows domain name, Windows computer name* even when the browser is in InPrivate mode. Even if the user is behind a proxy, clears the browser state, and uses InPrivate, SMB connections identify the user to the remote site. While experimenting with this we found that many ISPs filter the SMB port 445 which makes this attack difficult in practice.

4 Usage measurement

We conducted an experiment to determine how the choice of browser and the type of site being browsed affects whether users enable private browsing mode. We used advertisement networks as a delivery mechanism for our measurement code, using the same ad network and technique previously demonstrated in [10, 4].

Design. We ran two simultaneous one-day campaigns: a campaign that targeted adult sites, and a campaign that targeted gift shopping sites. We also ran a campaign on news sites as a control. We spent $120 to purchase 155,216 impressions, split evenly as possible between the campaigns. Our advertisement detected private browsing mode by visiting a unique URL in an `<iframe>` and using JavaScript to check whether a link to that URL was displayed as purple (visited) or blue (unvisited). The technique used to read the link color varies by browser; on Firefox, we used the following code:

```
if (getComputedStyle(link).color ==
         "rgb(51,102,160)" )
  // Link is purple, private browsing is OFF
} else {
  // Link is blue, private browsing is ON
}
```

To see why this browser history sniffing technique [11] reveals private browsing status, recall that in private mode all browsers do not add entries to the history database. Consequently, they will color the unique URL link as unvisited. However, in public mode the unique URL will be added to the history database and the browser will render the link as visited. Thus, by reading the link color we learn the browser's privacy state. We developed a demonstration of this technique in February 2009 [9]. To the best of our knowledge, we are the first to demonstrate this technique to detect private browsing mode in all major browsers.

While this method correctly detects all browsers in private browsing, it can slightly over count due to false positives. For example, some people may disable the history feature in their browser altogether, which will incorrectly make us think they are in private mode. In Firefox, users can disable the `:visited` pseudotag using a Firefox preference used as a defense against history sniffing. Again, this will make us think they are in private mode. We excluded beta versions of Firefox 3.7 and Chrome 6 from our experiment, since these browsers have experimental visited link defenses that prevent our automated experiment from working. However, we note that these defenses are not sufficient to prevent web attackers from detecting private browsing, since they are not designed to be robust against attacks that involve user interaction [3]. We also note that the experiment only measures the presence of private mode, not the intent of private mode—some users may be in private mode without realizing it.

Results. The results of our ad network experiment are shown in Figure 2. We found that private browsing was more popular at adult web sites than at gift shopping sites and news sites, which shared a roughly equal level of private browsing use. This observation suggests that some browser vendors may be mischaracterizing the primary use of the feature when they describe it as a tool for buying surprise gifts [8, 17].

We also found that private browsing was more commonly used in browsers that displayed subtle private browsing indicators. Safari and Firefox have subtle indicators and enforce a single mode across all windows; they had the highest rate of private browsing use. Google Chrome and Internet Explorer give users a separate window for private browsing, and have more obvious private browsing indicators; these browsers had lower rates of private browsing use. These observations suggest that users may remain in private browsing mode for longer if they are not reminded of its existence by a separate window with obvious indicators.

Ethics. The experimental design complied with the terms of service of the advertisement network. The servers logged only information that is typically logged by advertisers when their advertisements are displayed. We also chose not to log the client's IP address. We discussed the experiment with the institutional review boards at our respective institutions and were instructed that a formal IRB review was not required because the advertisement did not interact or intervene with individuals or obtain identifiable private information.

5 Weaknesses in current implementations: a systematic study

Given the complexity of modern browsers, a systematic method is needed for testing that private browsing modes adequately defend against the threat models of Section 2. During our blackbox testing in Section 3.1 it became clear that we need a more comprehensive way to en-
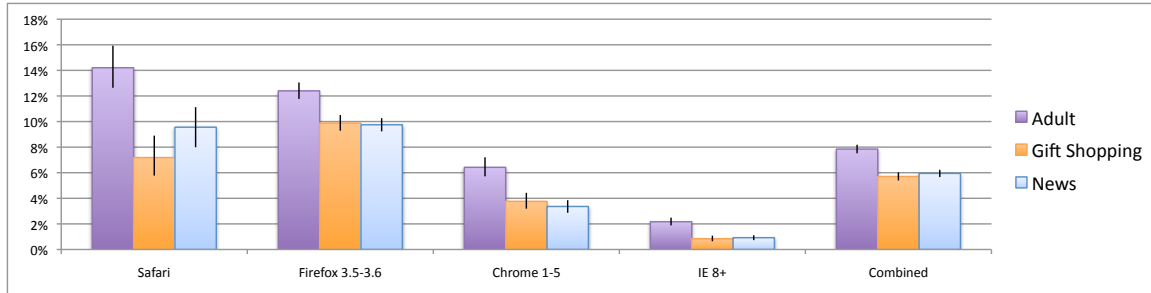
Figure 2: Observed rates of private browsing use

sure that *all* browser features behave correctly in private mode. We performed two systematic studies:

- Our first study is based on a manual review of the Firefox source code. We located all points in the code where Firefox writes to persistent storage and manually verified that those writes are disabled in private browsing mode.

- Our second study is an automated tool that runs the Firefox unit tests in private browsing mode and looks for changes in persistent storage. This tool can be used as a regression test to ensure that new browser features are consistent with private browsing.

We report our results in the next two sections.

5.1 A systematic study by manual code review

Firefox keeps all the state related to the user's browsing activity including preferences, history, cookies, text entered in forms fields, search queries, etc. in a *Profile* folder on disk [22]. By observing how and when persistent modifications to these files occur in private mode we can learn a great deal about how private mode is implemented in Firefox. In this section we describe the results of our manual code review of all points in the Firefox code that modify files in the *Profile* folder.

Our first step was to identify those files in the profile folder that contain information about a private browsing session. Then, we located the modules in the Mozilla code base that directly or indirectly modify these files. Finally, we reviewed these modules to see if they write to disk while in private mode.

Our task was greatly simplified by the fact that all writes to files inside the *Profile* directory are done using two code abstractions. The first is `nsIFile`, a cross-platform representation of a location in the filesystem used to read or write to files [21]. The second is `Storage`, a SQLite database API that can be used by other Firefox components and extensions to manipulate SQLite database files [23]. Points in the code that call these abstractions can check the current private browsing state by calling or hooking into the `nsIPrivateBrowsingService` interface [24].

Using this method we located 24 points in the Firefox 3.6 code base that control all writes to sensitive files in the *Profile* folder. Most had adequate checks for private browsing mode, but some did not. We give a few examples of points in the code that do not adequately check private browsing state.

- Security certificate settings (stored in file `cert8.db`): stores all security certificate settings and any SSL certificates that have been imported into Firefox either by an authorized website or manually by the user. This includes SSL client certificates.

  There are no checks for private mode in the code. We explained in Section 3.1 that this is a violation of the private browsing security model since a local attacker can easily determine if the user visited a site that generates a client key pair or installs a client certificate in the browser. We also note that certificates created outside private mode are usable in private mode, enabling a web attacker to link the user in public mode to the same user in private mode.

- Site-specific preferences (stored in file `permissions.sqlite`): stores many of Firefox permissions that are decided on a per-site basis. For example, it stores which sites are allowed or blocked from setting cookies, installing extensions, showing images, displaying popups, etc.

  While there are checks for private mode in the code, not all state changes are blocked. Permissions added to block cookies, popups or allow add-ons in private mode are persisted to disk. Consequently, if a user visits some site that attempts to open a popup, the popup blocker in Firefox blocks it and displays

a message with some actions that can be taken. In private mode, the "Edit popup blocker preferences" option is enabled and users who click on that option can easily add a permanent exception for the site without realizing that it would leave a trace of their private browsing session on disk. When browsing privately to a site that uses popups, users might be tempted to add the exception, thus leaking information to the local attacker.

- Download actions (stored in file `mimeTypes.rdf`): the file stores the user's preferences with respect to what Firefox does when it comes across known file types like pdf or avi. It also stores information about which protocol handlers (desktop-based or custom protocol handlers) to launch when it encounters a non-http protocol like `mailto` [26].

  There are no checks for private mode in the code. As a result, a webpage can install a custom protocol handler into the browser (with the user's permission) and this information would be persisted to disk even in private mode. As explained in Section 3.1, this enables a local attacker to learn that the user visited the website that installed the custom protocol handler in private mode.

5.2   An automated private browsing test using unit tests

All major browsers have a collection of unit tests for testing browser features before a release. We automate the testing of private browsing mode by leveraging these tests to trigger many browser features that can potentially violate private browsing. We explain our approach as it applies to the Firefox browser. We use MozMill, a Firefox user-interface test automation tool [20]. Mozilla provides about 196 MozMill tests for the Firefox browser.

Our approach. We start by creating a fresh browser profile and set preferences to always start Firefox in private browsing mode. Next we create a backup copy of the profile folder and start the MozMill tests. We use two methods to monitor which files are modified by the browser during the tests:

- `fs_usage` is a Mac OSX utility that presents system calls pertaining to filesystem activity. It outputs the name of the system call used to access the filesystem and the file descriptor being acted upon. We built a wrapper script around this tool to map the file descriptors to actual pathnames using `lsof`. We run our script in parallel with the browser and the script monitors all files that the browser writes to.

- We also use the "last modified time" for files in the profile directory to identity those files that are changed during the test.

Once the MozMill test completes we compare the modified profile files with their backup versions and examine the exact changes to eliminate false positives. In our experiments we took care to exclude all MozMill tests like "testPrivateBrowsing" that can turn off private browsing mode. This ensured that the browser was in private mode throughout the duration of the tests.

We did the above experiment on Mac OSX 10.6.2 and Windows Vista running Firefox 3.6. Since we only consider the state of browser profile and start with a clean profile, the results should not depend on OS or state of the machine at the time of running the tests.

Results. After running the MozMill tests we discovered several additional browser features that leak information about private mode. We give a few examples.

- Certificate Authority (CA) Certificates (stored in `cert8.db`). Whenever the browser receives a certificate chain from the server, it stores all the certificate authorities in the chain in `cert8.db`. Our tests revealed that CA certs cached in private mode persist when private mode ends. This is significant privacy violation. Whenever the user visits a site that uses a non-standard CA, such as certain government sites, the browser will cache the corresponding CA cert and expose this information to the local attacker.

- SQLite databases. The tests showed that the last modified timestamps of many SQLite databases in the profile folder are updated during the test. But at the end of the tests, the resulting files have exactly the same size and there are no updates to any of the tables. Nevertheless, this behavior can exploited by a local attacker to discover that private mode was turned on in the last browsing session. The attacker simply observes that no entries were added to the history database, but the SQLite databases were accessed.

- Search Plugins (stored in `search.sqlite` and `search.json`). Firefox supports auto-discovery of search plugins [19, 25] which is a way for web sites to advertise their Firefox search plugins to the user. The tests showed that a search plugin added in private mode persists to disk. Consequently, a local attacker will discover that the user visited the web site that provided the search plugin.

- Plugin Registration (stored in `pluginreg.dat`). This file is generated automatically and records information about installed plugins like Flash and

Quicktime. We observed changes in modification time, but there were only cosmetic changes in the file content. However, as with search plugins, new plugins installed in private mode result in new information written to `pluginreg.dat`.

Discovering these leaks using MozMill tests is much easier than a manual code review.

Using our approach as a regression tool. Using existing unit tests provides a quick and easy way to test private browsing behavior. However, it would be better to include testcases that are designed specifically for private mode and cover all browser components that could potentially write to disk. The same suite of testcases could be used to test all browsers and hence would bring some consistency in the behavior of various browsers in private mode.

As a proof of concept, we wrote two MozMill testcases for the violations discovered in Section 5.1:

- Site-specific Preferences (stored in file `permissions.sqlite`): visits a fixed URL that open up a popup. The test edits preferences to allow a popup from this site.
- Download Actions (`mimeTypes.rdf`): visits a fixed URL that installs a custom protocol handler.

Running these tests using our testing script revealed writes to both profile files involved.

6 Browser addons

Browser addons (extensions and plug-ins) pose a privacy risk to private browsing because they can persist state to disk about a user's behavior in private mode. The developers of these addons may not have considered private browsing mode while designing their software, and their source code is not subject to the same rigorous scrutiny that browsers are subjected to. Each of the different browsers we surveyed had a different approach to addons in private browsing mode:

- **Internet Explorer** has a configurable "Disable Toolbars and Extensions when InPrivate Browsing Mode Starts" menu option, which is checked by default. When checked, extensions (browser helper objects) are disabled, although plugins (ActiveX controls) are still functional.

- **Firefox** allows extensions and plugins to function normally in Private Browsing mode.

- **Google Chrome** disables most extension functionality in Incognito mode. However, plugins (including plugins that are bundled with extensions) are enabled. Users can add exceptions on a per-extension basis using the extensions management interface.

- **Safari** does not have a supported extension API. Using unsupported APIs, it is possible for extensions to run in private browsing mode.

In Section 6.1, we discuss problems that can occur in browsers that allow extensions in private browsing mode. In Section 6.2 we discuss approaches to address these problems, and we implement a mitigation in Section 6.3.

6.1 Extensions violating private browsing

We conducted a survey of extensions to find out if they violated private browsing mode. This section describes our findings.

Firefox. We surveyed the top 40 most popular add-ons listed at `http://addons.mozilla.org`. Some of these extensions like "Cooliris" contain binary components (native code). Since these binary components execute with the same permissions as those of the user, the extensions can, in principle, read or write to any file on disk. This arbitrary behavior makes the extensions difficult to analyze for private mode violations. We regard all *binary extensions* as unsafe for private browsing and focus our attention only on *JavaScript-only extensions*.

To analyze the behavior of JavaScript-only extensions, we observed all persistent writes they caused when the browser is running in private mode. Specifically, for each extension, we install that extension and remove all other extensions. Then, we run the browser for some time, do some activity like visiting websites and modifying extension options so as to exercise as many features of the extension as possible and track all writes that happen during this browsing session. A manual scan of the files and data that were written then tells us if the extension violated private mode. If we find any violations, the extension is *unsafe* for private browsing. Otherwise, it may or may not be safe.

Tracking all writes caused by extensions is easy as almost all JavaScript-only extensions rely on either of the following three abstractions to persist data on disk:

- `nsIFile` is a cross-platform representation of a location in the filesystem. It can be used to create or remove files/directories and write data when used in combination with components such as `nsIFileOutputStream` and `nsISafeOutputStream`.

- `Storage` is a SQLite database API [23] and can be used to create, remove, open or add new entries to SQLite databases using components like `mozIStorageService`, `mozIStorageStatement` and `mozIStorageConnection`.

- `Preferences` can be used to store preferences containing key-value (boolean, string or integer) pairs using components like `nsIPrefService`, `nsIPrefBranch` and `nsIPrefBranch2`.

We instrumented Firefox (version 3.6 alpha1 pre, co-denamed *Minefield*) by adding log statements in all functions in the above Mozilla components that could write data to disk. This survey was done on a Windows Vista machine.

Out of the 32 JavaScript-only extensions, we did not find any violations for 16 extensions. Some of these extensions like "Google Shortcuts" did not write any data at all and some others like "Firebug" only wrote boolean preferences. Other extensions like "1-Click YouTube Video Download" only write files that users want to download whereas "FastestFox" writes bookmarks made by the user. Notably, only one extension ("Tab Mix Plus") checks for private browsing mode and disables the UI option to save session if it is detected.

For 16 extensions, we observed writes to disk that can allow an attacker to learn about private browsing activity. We provide three categories of the most common violations below:

- **URL whitelist/blocklist/queues.** Many extensions maintain a list of special URLs that are always excluded from processing. For instance, "NoScript" extension blocks all scripts running on visited webpages. User can add sites to a whitelist for which it should allow all scripts to function normally. Such exceptions added in private mode are persisted to disk. Also, downloaders like "DownThemAll" maintain a queue of URLs to download from. This queue is persisted to disk even in private mode and not cleared until download completes.

- **URL Mappings.** Some extensions allow specific features or processing to be enabled for specific websites. For instance, "Stylish" allows different CSS styles to be used for rendering pages from different domains. The mapping of which style to use for which website is persisted to disk even in private mode.

- **Timestamp.** Some extensions store a timestamp indicating the last use of some feature or resource. For instance, "Personas" are easy-to-use themes that let the user personalize the look of the browser. It stores a timestamp indicating the last time when the theme was changed. This could potentially be used by an attacker to learn that private mode was turned on by comparing this timestamp with the last timestamp when a new entry was added to the browser history.

It is also interesting to note that the majority of the extensions use `Preferences` or `nsIFile` to store their data and very few use the SQLite database. Out of the 32 JavaScript-only extensions, only two use the SQLite database whereas the rest of them use the former.

Google Chrome. Google launched an extension platform for Google Chrome [5] at the end of January 2010. We have begun a preliminary analysis of the most popular extensions that have been submitted to the official extensions gallery. Of the top 100 extensions, we observed that 71 stored data to disk using the `localStorage` API. We also observed that 5 included plugins that can run arbitrary native code, and 4 used Google Analytics to store information about user behavior on a remote server. The significant use of local storage by these extensions suggests that they may pose a risk to Incognito.

6.2 Running extensions in private browsing

Current browsers force the user to choose between running extensions in private browsing mode or blocking them. Because not all extensions respect private browsing mode equally, these policies will either lead to privacy problems or block extensions unnecessarily. We recommend that browser vendors provide APIs that enable extension authors to decide which state should be persisted during private browsing and which state should be cleared. There are several reasonable approaches that achieve this goal:

- **Manual check.** Extensions that opt-in to running in private browsing mode can detect the current mode and decide whether or not to persist state.

- **Disallow writes.** Prevent extensions from changing any local state while in private browsing mode.

- **Override option.** Discard changes made by extensions to local state while in private browsing mode, unless the extension explicitly indicates that the write should persist beyond private browsing mode.

Several of these approaches have been under discussion on the Google Chrome developers mailing list [28]. We describe our implementation of the first variant in Section 6.3. We leave the implementation of the latter variants for future work.

6.3 Extension blocking tool

To implement the policy of blocking extensions from running in private mode as described in section 6.2, we built a Firefox extension called *ExtensionBlocker*

in 371 lines of JavaScript. Its basic functionality is to disable all extensions that are not *safe* for private mode. So, all unsafe extensions will be disabled when the user enters private mode and then re-enabled when the user leaves private mode. An extension is considered safe for private mode if its manifest file (`install.rdf` for Firefox extensions) contains a new XML tag `<privateModeCompatible/>`. Table 4 shows a portion of the manifest file of ExtensionBlocker declaring that it is safe for private browsing.

ExtensionBlocker subscribes to the `nsIPrivateBrowsingService` to observe transitions into and out of private mode. Whenever private mode is enabled, it looks at each enabled extension in turn, checks their manifest file for the `<privateModeCompatible/>` tag and disables the extension if no tag is found. Also, it saves the list of extensions that were enabled before going to private mode. Lastly, when the user switches out of private mode, it re-enables all extensions in this saved list. At this point, it also cleans up the saved list and any other state to make sure that we do not leave any traces behind.

One implementation detail to note here is that we need to restart Firefox to make sure that appropriate extensions are completely enabled or disabled. This means that the browser would be restarted at every entry into or exit from private mode. However, the public browsing session will still be restored after coming out of private mode.

7 Related work

Web attacker. Most work on private browsing focuses on security against a web attacker who controls a number of web sites and is trying to determine the user's browsing behavior at those sites. Torbutton [29] and Fox-Tor [31] are two Firefox extensions designed to make it harder for web sites to link users across sessions. Both rely on the Tor network for hiding the client's IP address from the web site. PWS [32] is a related Firefox extension designed for search query privacy, namely preventing a search engine from linking a sequence of queries to a specific user.

Earlier work on private browsing such as [34] focused primarily on hiding the client's IP address. Browser fingerprinting techniques [1, 14, 6] showed that additional steps are needed to prevent linking at the web site. Torbutton [29] is designed to mitigate these attacks by blocking various browser features used for fingerprinting the browser.

Other work on privacy against a web attacker includes Janus [7], Doppelganger [33] and Bugnosis [2]. Janus is an anonymity proxy that also provides the user with anonymous credentials for logging into sites. Doppelganger [33] is a client-side tool that focuses on cookie privacy. The tool dynamically decides which cookies are needed for functionality and blocks all other cookies. Bugnosis [2] is a Firefox extension that warns users about server-side tracking using web bugs. Millet et al. carry out a study of cookie policies in browsers [18].

P3P is a language for web sites to specify privacy policies. Some browsers let users configure the type of sites they are willing to interact with. While much work went into improving P3P semantics [13, 27, 30] the P3P mechanism has not received widespread adoption.

Local attacker. In recent years computer forensics experts developed an array of tools designed to process the browser's cache and history file in an attempt to learn what sites a user visited before the machine was confiscated [12]. *Web historian*, for example, will crawl browser activity files and report on all recent activity done using the browser. The tool supports all major browsers. The Forensic Tool Kit (FTK) has similar functionality and an elegant user interface for exploring the user's browsing history. A well designed private browsing mode should successfully hide the user's activity from these tools.

In an early analysis of private browsing modes, McKinley [15] points out that the Flash Player and Google Gears browser plugins violate private browsing modes. Flash player has since been updated to be consistent with the browser's privacy mode. More generally, NPAPI, the plugin API, was extended to allow plugins to query the browser's private browsing settings so that plugins can modify their behavior when private browsing is turned on. We showed that the problem is more complex for browser extensions and proposed ways to identify and block problematic extensions.

8 Conclusions

We analyzed private browsing modes in modern browsers and discussed their success at achieving the desired security goals. Our manual review and automated testing tool pointed out several weaknesses in existing implementations. The most severe violations enable a local attacker to completely defeat the benefits of private mode. In addition, we performed the first measurement study of private browsing usage in different browsers and on different sites. Finally, we examined the difficult issues of keeping browser extensions and plug-ins from undoing the goals of private browsing.

Future work. Our results suggest that current private browsing implementations provide privacy against some local and web attackers, but can be defeated by determined attackers. Further research is needed to design

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7−c20a−464f−9b0e−13a3a9e97384}</em:id>
    <em:minVersion>1.5</em:minVersion>
    <em:maxVersion>3.*</em:maxVersion>
    <em:privateModeCompatible/>
  </Description>
</em:targetApplication>
```

Table 4: A portion of the manifest file of ExtensionBlocker

stronger privacy guarantees without degrading the user experience. For example, we ignored privacy leakage through volatile memory. Is there a better browser architecture that can detect all relevant private data, both in memory and on disk, and erase it upon leaving private mode? Moreover, the impact of browser extensions and plug-ins on private browsing raises interesting open problems. How do we prevent uncooperative and legacy browser extensions from violating privacy? In browsers like IE and Chrome that permit public and private windows to exist in parallel, how do we ensure that extensions will not accidentally transfer data from one window to the other? We hope this paper will motivate further research on these topics.

Acknowledgments

References

[1] 0x000000. Total recall on Firefox. http://mandark.fr/0x000000/articles/Total_Recall_On_Firefox..html.

[2] Adil Alsaid and David Martin. Detecting web bugs with Bugnosis: Privacy advocacy through education. In *Proc. of the 2002 Workshop on Privacy Enhancing Technologies (PETS)*, 2002.

[3] David Baron et al. :visited support allows queries into global history, 2002. https://bugzilla.mozilla.org/show_bug.cgi?id=147777.

[4] Adam Barth, Collin Jackson, and John C. Mitchell. Robust defenses for cross-site request forgery. In *Proc. of the 15th ACM Conference on Computer and Communications Security. (CCS)*, 2008.

[5] Nick Baum. Over 1,500 new features for Google Chrome, January 2010. http://chrome.blogspot.com/2010/01/over-1500-new-features-for-google.html.

[6] Peter Eckersley. A primer on information theory and privacy, January 2010. https://www.eff.org/deeplinks/2010/01/primer-information-theory-and-privacy.

[7] E. Gabber, P. B. Gibbons, Y. Matias, and A. Mayer. How to make personalized web browing simple, secure, and anonymous. In *Proceedings of Financial Cryptography'97*, volume 1318 of *LNCS*, 1997.

[8] Google. Explore Google Chrome features: Incognito mode (private browsing). http://www.google.com/support/chrome/bin/answer.py?hl=en&answer=95464.

[9] Jeremiah Grossman and Collin Jackson. Detecting Incognito, Feb 2009. http://crypto.stanford.edu/~collinj/research/incognito/.

[10] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from DNS rebinding attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, 2007.

[11] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *Proc. of the 15th International World Wide Web Conference (WWW)*, 2006.

[12] Keith Jones and Rohyt Belani. Web browser forensics, 2005. www.securityfocus.com/infocus/1827.

[13] Stephen Levy and Carl Gutwin. Improving understanding of website privacy policies with fine-grained policy anchors. In *Proc. of WWW'05*, pages 480–488, 2005.

[14] Jonathan R. Mayer. *"Any person... a pamphleteer": Internet Anonymity in the Age of Web 2.0.* PhD thesis, Princeton University, 2009.

[15] Katherine McKinley. Cleaning up after cookies, Dec. 2008. https://www.isecpartners.com/files/iSEC_Cleaning_Up_After_Cookies.pdf.

[16] Jorge Medina. Abusing insecure features of internet explorer, Febuary 2010. http://www.blackhat.com/presentations/bh-dc-10/Medina_Jorge/BlackHat-DC-2010-Medina-Abusing-/insecure-features-of-Internet-/Explorer-wp.pdf.

[17] Microsoft. InPrivate browsing. http://www.microsoft.com/windows/internet-explorer/features/safer.aspx.

[18] Lynette Millett, Batya Friedman, and Edward Felten. Cookies and web browser design: Toward realizing informed consent online. In *Proce. of the CHI 2001*, pages 46–52, 2001.

[19] Mozilla Firefox - Creating OpenSearch plugins for Firefox. https://developer.mozilla.org/en/Creating_OpenSearch_plugins_for_Firefox.

[20] Mozilla Firefox - MozMill. http://quality.mozilla.org/projects/mozmill.

[21] Mozilla Firefox - nsIFile. https://developer.mozilla.org/en/nsIFile.

[22] Mozilla Firefox - Profiles. http://support.mozilla.com/en-US/kb/Profiles.

[23] Mozilla Firefox - Storage. https://developer.mozilla.org/en/Storage.

[24] Mozilla Firefox - Supporting private browsing mode. https://developer.mozilla.org/En/Supporting_private_browsing_mode.

[25] OpenSearch. http://www.opensearch.org.

[26] Web-based protocol handlers. https://developer.mozilla.org/en/Web-based_protocol_handlers.

[27] The platform for privacy preferences project (P3P). http://www.w3.org/TR/P3P.

[28] Matt Perry. RFC: Extensions Incognito, January 2010. http://groups.google.com/group/chromium-dev/browse_thread/thread/5b95695a7fdf6c15/b4052bb405f2820f.

[29] Mike Perry. Torbutton. http://www.torproject.org/torbutton/design.

[30] J. Reagle and L. Cranor. The platform for privacy preferences. *CACM*, 42(2):48–55, 1999.

[31] Sasha Romanosky. FoxTor: helping protect your identity while browsing online. cups.cs.cmu.edu/foxtor.

[32] F. Saint-Jean, A. Johnson, D. Boneh, and J. Feigenbaum. Private web search. In *Proc. of the 6th ACM Workshop on Privacy in the Electronic Society (WPES)*, 2007.

[33] Umesh Shankar and Chris Karlof. Doppelganger: Better browser privacy without the bother. In *Proceedings of ACM CCS'06*, pages 154–167, 2006.

[34] Paul Syverson, Michael Reed, and David Goldschlag. Private web browsing. *Journal of Computer Security (JCS)*, 5(3):237–248, 1997.

[35] Lewis Thompson. Chrome incognito tracks visited sites, 2010. www.lewiz.org/2010/05/chrome-incognito-tracks-visited-sites.html.