Evaluating Private Modes in Desktop and Mobile Browsers and Their Resistance to Fingerprinting

Yuanyi Wu
ShanghaiTech University
Email: wu_yuanyi@icloud.com

Dongyu Meng
ShanghaiTech University
Email: mengdy@shanghaitech.edu.cn

Hao Chen
UC Davis
Email: chen@ucdavis.edu

***Abstract*—Modern browsers implement private mode to protect user privacy. However, they do not agree on what protection private mode should provide. We performed the first study on comparing private modes in popular desktop and mobile browsers and found many inconsistencies between different browsers and between the desktop and mobile versions of the same browser. We show that some inconsistencies result from the tradeoff between security and privacy. However, even if private mode leaks no information about the user, the attacker could still track the user by fingerprinting the browser. Recent work suggested that a browser could report randomized configurations, such as font sizes and installed plugins, to defeat fingerprinting. To show that randomizing configuration reports is insecure, we propose an attack that estimates the true configuration based on statistical methods. We demonstrated that this attack was easy and effective.**

***Keywords*—private mode, browsing history, fingerprint, statistics**

I. Introduction

Modern browsers introduced private browsing mode to protect user privacy. From the user's perspective, a browsing session in private mode looks like a session on a public computer: the browser isolates all the data created or modified during the session, and destroy them after the session ends. Private mode defends against two types of attackers: 1) Web attacker: a malicious website trying to link a private mode session to other current or future sessions of the same browser; 2) Local attacker: a local user trying to read information about a private mode session after the session ends.

In 2010, Aggarwal et al. analyzed the implementation of private browsing mode of four desktop browsers — Chrome, Firefox, Safari, and IE — and found several inconsistencies [2]. They showed that both web and local attackers can take advantage of these inconsistencies to read user data in private mode. Since their work, mobile browsers became popular and are commanding a growing market share. Major vendors support private mode in both desktop and mobile versions of their browsers.

We performed the first study on analyzing private modes in both the desktop and mobile versions of popular browsers. We found many implementation inconsistencies between different browsers as well as between the desktop and mobile versions of the same browser. These inconsistencies allow a web or local attacker to compromise user privacy even when the user browses in private mode. We will show that some inconsistencies result from the tradeoff between security and privacy.

978-1-5386-0683-4/17$31.00

Name	Version	Platform	Name of private browsing
Chrome Chrome (M)	52 52.0	Windows 8.1 Android 5.0 (HTC)	Incognito
Firefox Firefox (M)	48 48.0	Windows 8.1 Android 5.0 (HTC)	Private Browsing
Safari Safari (M)	10.0 10	MacOS 10.11.6 iOS 10 (iPhone 7)	Private Browsing
Edge	38	Windows 10	InPrivate

TABLE I: Browsers whose private modes we compared

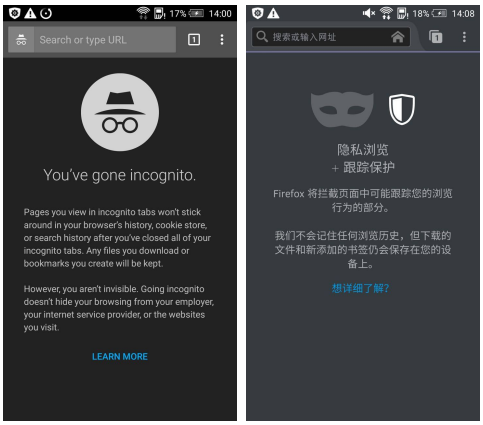
Even if private mode completely isolates users' private data, it may not completely protect user privacy. Researchers showed that a web attacker could fingerprint a browser to link different sessions in the same browser, including private sessions. Recently, Nikiforakis proposed a method to defeat browser fingerprinting by randomizing the reported font sizes and installed plugins [12]. We propose an attack that can fingerprint a browser accurately despite that defense. The gist of our attack is to take multiple measurements and use statistical methods to estimate the true configuration. We demonstrate that our attack is easy yet effective.

II. Comparison of Private Modes in Modern Browsers

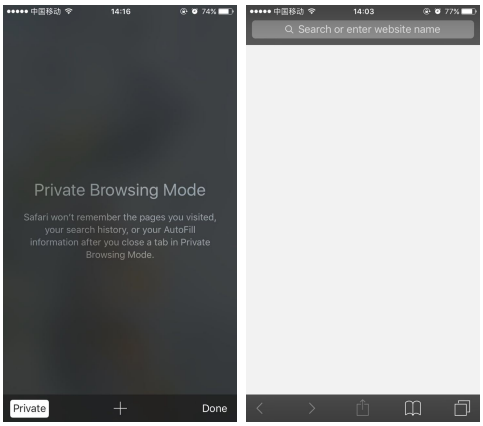
We evaluated the implementation of private mode of seven major browsers in Table I.

A. User Interface

1) Visual indicators: Figure 1a, Figure 1b, Figure 1c, show the user interface when user enters private mode in mobile Chrome, Firefox, and Safari, respectively. All of them show notification of private mode in the main window. However, when user opens a new tab, while Chrome and Firefox show a prominent indicator of private mode in browser window, Safari only changes the top and bottom bars in the browser chrome to a unique color (Figure 1d). Prior research showed that inadequate visual indicator may cause users in private mode to leave private mode on when they intend to quit private mode [2]. Same problem shows up in desktop Safari too.



(a) Chrome 52.0 (b) Firefox 48.0



(c) Safari 10 (d) Safari new tab

Fig. 1: Private browsing indicators in major mobile browsers

2) Starting private mode: In desktop Chrome and desktop Firefox, users can start private mode directly. However, in desktop Safari and desktop Edge, users have to start private mode from existing public mode sessions. The latter design makes it more burdensome to enter private mode and hence may discourage its uses.

On mobile Safari, if users are in private mode when they close the browser, the browser will remain in private mode when started next time. By contrast, on mobile Chrome and Firefox, the browser will restart in public mode.

3) Security exceptions: Once users add a security exception in mobile Firefox or mobile Safari, they cannot remove this exception through UI interface. In desktop and mobile Chrome as well as desktop Firefox, users can remove the exception by clicking an icon next to the address bar. Desktop Safari and desktop Edge do not remember security exceptions after users quit browsers' window in public or private mode.

4) Certificates: In desktop Safari, if users want to delete a certificate, they need to do it in another application named *Keychain Access*. In mobile Firefox and mobile safari, users cannot delete certificates in browsers. Edge currently has no support for certificates.

5) Helper for external protocols: In desktop Chrome, if users chose *remember choice for all links of this type when Chrome launch an external protocol request* in private mode or public mode, they cannot modify this setting anymore, unless they edit Chrome's *Local State* file in a hidden folder.

6) Settings: In desktop Firefox and desktop Safari, users can open *Preferences* page in private mode. In desktop Chrome, when users click *Settings* in private mode, the browser will open it in public mode.

B. Information Flow

We examined information flow in three scenarios.

- Information flow from public to private mode.
- Information leftover from private sessions.
- Information flow among pages in the same tab or throughout different tabs within private mode.

1) Information flow from public to private mode: Table II shows which states set in public mode are available in private mode even after users close their sessions in public mode. Such information could allow websites to link users' sessions in public mode to those in private mode.

2) JavaScript API: In all the browsers, cookies set in public mode are inaccessible in private mode. However, these browsers differ in how they authorize JavaScript API. In private mode, Chrome blocks `webkitRequsetFileSystem`, Firefox and Edge block `indexedDB`, and Safari blocks `LocalStorage` and `SessionStorage`. The purpose is to prevent web pages from accessing cookies via these API calls, but this mechanism inadvertently allows web pages to detect private mode. This is an example of tradeoff between security and privacy. We suggest that instead of blocking these calls, browsers should allow these calls but return empty data, just like how they handle unauthorized cookie requests.

3) Information leftover from private sessions: Table III shows what states set in private mode remain persistent after the user closes all private sessions. Such information allows a local attacker to discover the previous user's data created in private sessions.

All browsers delete history, cookies, password database, and auto-completed forms in private mode after the user closes private sessions.

a) Certificates: In mobile Chrome, desktop Firefox, mobile Firefox, desktop Safari, and mobile Safari, website certificates imported in private mode remain accessible after the session ends. In mobile Firefox, if the user deletes certificates in private mode, the browser removes the certificates in public mode as well.

b) Security exceptions: In desktop Firefox, if the user adds a security exception in private mode, it will not be visible in public mode; however, if users remove a security exception in private mode, the browser removes the exception in public mode. This is an example where the browser chooses security over privacy in the trade off. In mobile Safari, if a user adds a security exception in private mode, it is accessible in public mode.

	Chrome	Chrome (M)	Firefox	Firefox (M)	Safari	Safari (M)	Edge
history list	×	✓	✓	✓	✓	✓	✓
cookies	×	×	×	×	×	×	×
LocalStorage	×	×	×	×	…	…	×
SessionStorage	×	×	×	×	…	…	×
indexedDB	×	×	…	…	—	—	…
bookmarks	✓	✓	✓	✓	✓	✓	✓
password database	✓	✓	✓	✓	✓	✓	✓
form autocompletion	✓	✓	✓	✓	✓	✓	✓
add security exception	✓	✓	✓	✓	×	✓	×
remove security exception	✓	✓	✓	—	—	—	—
download list	✓	—	×	✓	—	—	×
download item	✓	✓	✓	✓	✓	✓	✓
suggest based on browsing history	✓	✓	—	✓	✓	✓	✓
browser's web cache	×	×	×	×	×	×	×
import certs	✓	✓	✓	✓	✓	✓	—
delete certs	✓	—	✓	—	—	—	—
remember choice for external protocol	✓	—	✓	✓	—	—	✓
per-site zoom level	✓	—	✓	—	—	—	✓
webkitRequestFileSystem	…	…	—	—	—	—	—

TABLE II: Which states in public mode are accessible in private mode?
✓: The state is accessible. ×: The state is inaccessible.
—: The state is unavailable in both public and private mode.
…: The state is unavailable in private mode.

	Chrome	Chrome (M)	Firefox	Firefox (M)	Safari	Safari (M)	Edge
history list	×	×	×	×	×	×	×
cookies	×	×	×	×	×	×	×
LocalStorage	×	×	×	×	…	…	×
SessionStorage	×	×	×	×	…	…	×
indexedDB	×	×	…	…	—	—	…
bookmarks	✓	✓	✓	✓	✓	✓	✓
password database	×	×	×	×	×	×	×
form autocompletion	×	×	×	×	×	×	×
add security exception	×	×	×	×	×	✓	×
remove security exception	×	×	✓	—	—	—	—
download list	×	—	×	×	—	—	×
download item	✓	✓	✓	✓	✓	✓	✓
suggest based on browsing history	×	×	—	×	×	×	×
browser's web cache	×	×	×	×	×	×	×
import certs	×	✓	✓	✓	✓	✓	—
delete certs	×	—	✓	—	—	—	—
remember choice for external protocol	✓	—	✓	×	—	—	✓
per-site zoom level	×	—	×	—	—	—	✓
webkitRequestFileSystem	…	…	—	—	—	—	—

TABLE III: Which states in private mode are left over after private session ends?

c) External protocol handler: In desktop Chrome, desktop Firefox, and desktop Edge, applications for handling external protocols remembered in private are available in public mode.

4) Information flow within private mode: The third threat is web attackers linking users' sessions in private mode. We consider two scenarios, sessions in the same tab and sessions in different tabs. With private mode on, browsers do not update history, passwords, or search terms at all. However, new cookies are stored in memory while private mode is on and are erased when users exit private mode. Browser's cookies and web cache are not written to persistent storage, ensuring that private mode data will be erased by default even if browser crashes in private mode.

Although websites visited in private mode in mobile Safari are also suggested in public mode, they are not suggested in private mode. Another interesting fact is that in mobile Firefox, if a user initiates a download task from private mode tab while having tabs of both public and private mode open, the download task is also shown in download list of public mode tabs. The task disappears when the user kill all tabs in private mode.

a) Different pages in the same tab: Table IV shows what data in private mode are available to a page loaded later into the same tab. If a user loads website A, then website B, and finally website A again, the third page (website A) can access data (e.g., cookies) stored by the first page (website A). This happens in all seven browsers. It allows a web attacker to link sessions in the same tab.

b) Pages in different tabs: Table V shows what data in private mode are available to a page loaded into a different tab. All browsers except desktop and mobile Safari allow pages from the same website in different tabs to share cookies. Therefore, only Safari prevents a web attacker from linking sessions in different tabs using cookies. Edge has an inconsistent policy regarding the sharing of *LocalStorage*, *Session-*

	Chrome	Chrome (M)	Firefox	Firefox (M)	Safari	Safari (M)	Edge
history list	×	×	×	×	×	×	×
cookies	✓	✓	✓	✓	✓	✓	✓
LocalStorage	✓	✓	✓	✓	…	…	✓
SessionStorage	✓	✓	✓	✓	…	…	✓
indexedDB	✓	✓	…	…	—	—	…
bookmarks	✓	✓	✓	✓	✓	✓	✓
password database	×	×	×	×	×	×	×
form autocompletion	×	×	×	×	×	×	×
add security exception	✓	✓	✓	✓	✓	✓	✓
remove security exception	✓	✓	✓	—	—	—	—
download list	✓	—	✓	✓	—	—	✓
download item	✓	✓	✓	✓	✓	✓	✓
suggest based on browsing history	×	×	—	×	×	×	×
browser's web cache	✓	✓	✓	✓	✓	✓	✓
import certs	×	✓	✓	✓	✓	✓	—
delete certs	×	—	✓	—	—	—	—
remember choice for external protocol	✓	—	✓	×	—	—	✓
per-site zoom level	✓	—	✓	—	—	—	✓
webkitRequestFileSystem	…	…	—	—	—	—	—

TABLE IV: Which states created by a page in private mode are accessible to future pages in the same tab?

Storage, and cookies between different tabs. A website cannot share *LocalStorage* and *SessionStorage* between different tabs but can share cookies.

C. Difference between Desktop and Mobile Browsers

To illustrate mobile browsers' new features that are different from desktop browsers, we compared mobile Chrome on Android to desktop Chrome on Windows, mobile Firefox on Android to desktop Firefox on Windows and mobile Safari on iOS to desktop Safari on MacOS.

1) Screen: First, compared to desktop devices, a mobile device's screen is smaller but has higher resolution. Rendering the same image, the area needed for mobile devices is smaller than desktop devices. Second, most smart mobile device are equipped with touch screens. Comparing to precise mouse clicking in desktop browsers, most mobile device users touch an approximate location on the touch screen with fingers. During input, the virtual keyboard occupies almost half of the screen. Copying, cutting, and pasting are more difficult too.

- *History list in Chrome.* Mobile Chrome displays history list of public mode when requested in private mode. This new feature aims to simplify user input. Information flowing from public to private mode does not compromise user privacy.
- *Remove security exception in Firefox.* In desktop Firefox, to visit an unsafe website, a user must first add an exception. Desktop Firefox shows a warning icon at the head of address bar to notify the user about security risks. When the user leaves this page, she can click the icon to remove this security exception. However, in both public mode and private mode of mobile Firefox, no equivalent icon shows up. Therefore, the user is not able to remove the security exception in mobile Firefox, which means that she will not be notified again when she visits this unsafe website in the future. Meanwhile, Chrome supports this functionality in both its desktop and mobile version. We suggest that mobile Firefox should build this functionality back in.
- *Search suggestions based on browsing history in Firefox.* Mobile Firefox shows suggestions based on the browsing history of public mode in private mode. This feature saves users from entering the entire query. Information flows from public mode to private mode. User privacy remains preserved.

2) Operating system: Mobile operating systems bring about new features to mobile browsers too.

- *Remember choice of external protocol in Chrome.* When a user opens an external protocol in private mode of desktop Chrome and chooses to remember how to deal with this external protocol, desktop Chrome applies her choice to public mode too. Information flows from private mode to public mode, leaving traces for local attackers. Mobile Chrome forces users to open external protocols with default applications and hence avoids this problem. Mobile Firefox takes a step further, allowing users to choose whatever application they like to open external protocols in private mode and restores users' choice after they leave private mode. Mobile Chrome and mobile Firefox's new feature protects users' privacy.
- *Delete certificates in Chrome and Firefox.* The procedure for users to delete installed certificates is more cumbersome in mobile Chrome and mobile Firefox than in their desktop counterparts. Users have to open the *Setting* to delete certificates. Therefore, it is more troublesome for users to clean suspicious certificates on mobile devices.
- *Download list in Firefox.* Of all desktop browsers that we tested, none allows users to check the download list of public mode from private mode. On mobile devices, however, Firefox shows download list of public mode to users in private mode. The design improves usability. Desktop operating systems have mature support for multi-window UI so download list can be shown in a separate window when the user is in private mode. But most mobile devices only support a single window at front stage. In this case, information flows from public mode to private mode. User privacy is not harmed.

3) Inconsistency:

	Chrome	Chrome (M)	Firefox	Firefox (M)	Safari	Safari (M)	Edge
history list	×	×	×	×	×	×	×
cookies	✓	✓	✓	✓	×	×	✓
LocalStorage	✓	✓	✓	✓	…	…	×
SessionStorage	×	×	×	×	…	…	×
indexedDB	✓	✓	…	…	—	—	…
bookmarks	✓	✓	✓	✓	✓	✓	✓
password database	×	×	×	×	×	×	×
form autocompletion	×	×	×	×	×	×	×
add security exception	✓	✓	✓	✓	✓	✓	✓
remove security exception	✓	✓	✓	—	—	—	—
download list	✓	—	✓	✓	—	—	✓
download item	✓	✓	✓	✓	✓	×	✓
suggest based on browsing history	×	×	—	×	×	×	×
browser's web cache	✓	✓	✓	✓	×	×	×
import certs	×	✓	✓	✓	✓	✓	—
delete certs	×	—	✓	—	—	—	—
remember choice for external protocol	✓	—	✓	×	—	—	✓
per-site zoom level	✓	—	✓	—	—	—	✓
webkitRequestFileSystem	…	…	—	—	—	—	—

TABLE V: Which states created by a page in a tab in private mode are accessible to other tabs in private mode?

- *Import certificates in Chrome.* In desktop Chrome, certificates imported in private mode are not available in public mode. Desktop Chrome will delete certificates imported in private mode after the user leaves private mode. With experience with desktop Chrome in mind, when users try to import suspicious certificates for some websites in mobile Chrome, they may use private mode and naturally assume that mobile Chrome would delete imported certificates automatically when they leave private mode. But in fact, mobile Chrome does not delete certificates imported in private mode. Therefore, suspicious certificates remain in users' mobile Chrome. Such inconsistency is security-critical.
- *Add security exception in Safari.* In desktop Safari, users are notified when they try to visit unsafe sites. They may browse an unsafe site once they choose to continue on the user interface. This safety choice is not persistent as users get notified again when they visit the same unsafe site after a browser restart. Mobile Safari deploys a different design. Once a user chooses to continue to visit an unsafe website, she will not get any further notification for this site anymore unless she restores her device to factory settings. Users are more likely to make wrong judgements about whether a website is safe or not.

III. Discussion: Tradeoff Between Security and Privacy in Private Mode

section II shows that many popular browsers leak information between public and private modes. Often this problem is due to implementation weaknesses. However, if the browser prevents such leaks, security can be harmed. Aggarwal's work did not discuss these issues.

One example is HSTS [14]. HSTS is a strong security mechanism, but it can also be used when tracking users. Therefore, the browser faces a dilemma. If the saved HSTS domains in public mode are visible in private mode, a malicious web page could link the user's sessions in public mode to those in private mode. On the other hand, if, to protect user's privacy, the saved HSTS domains in public mode are invisible in private mode, then when the user visits these domains in private mode, the security of the connections won't be protected. Chrome prefers security over privacy, while Firefox does it the other way around.

Another example is security exception. Adding a security exception *decreases* security, and removing a security exception *increases* security. Chrome (both desktop and mobile) and desktop Firefox differ in how they choose the tradeoff between security and privacy. In Chrome, when the user adds or deletes a security exception in private mode, it does not affect public mode, i.e., the browser does not add or delete the security exception in public mode. This shows that Chrome chooses privacy over security. By contrast, in desktop Firefox, when the user adds a security exception in private mode, the browser does not add the exception in public mode. However, when the user deletes a security exception in private mode, the browser also deletes it in public mode. This shows that Firefox chooses security over privacy when protecting privacy would decrease security.

IV. Browser Fingerprinting

Even if private mode perfectly isolates all user information, web attackers may still link private and public sessions of the same user by fingerprinting the browser. Two common fingerprint attributes are installed fonts and plugins.

A. Background

1) Fingerprint: The attacker wishes to find out what fonts are installed in a browser, but he cannot query this information directly. Instead, he measures the sizes (widths and heights) of a string rendered in different fonts. Specifically, he compiles a list of common fonts and chooses a string whose sizes are different when rendered in different fonts. Then, his attack web page renders the string in each font and measures its size. If a font is installed, the measured size would match the expected size. Otherwise, if the font is not installed, the string would be rendered in the default font of the browser and sizes will be different. In this way, attackers are able to infer what fonts are installed with high accuracy. For plugins, the attack web page can query the installed plugins directly.

Font	Rendered text
Verdana	hello|★☆♀◆◇
MS Reference Sans Serif	hello|★☆♀◆◇
STSong	永字八法|hello2017
STFangsong	永字八法|hello2017

TABLE VI: Different fonts render the same character in the same size

2) PriVaricator's defense: PriVaricator is a tool to defeat fingerprinting. PriVaricator modifies the browser so that every web page sees a different browser configuration [12]. Specifically, for every web page, the browser perturbs the fonts sizes and the set of installed plugins. For each font, it randomly perturbs the reported `offsetHeight` and `offsetWidth` by $\pm 5\%$, an amount that they determined to strike a good balance between privacy and usability. When reporting plugins, it randomly omits 30% plugins.

B. Limitation

1) Limitations on fingerprinting based on installed fonts: The set of installed fonts is a popular feature to fingerprint browsers. In practice, a web page cannot query the font list directly if Flash is disabled. In this case, the web page has to infer the font list. A common inference mechanism is described in IV.A.1. However, we found this approach ineffective as the perturbation introduced by PriVaricator is not neglectable in the inference procedure. In fact, this inference method remains ineffective even without the presence of PriVaricator, because many fonts share the same size by design. For example, a user installs font A and font B. Both fonts render a string in the same size but the user's default font is font A. In this case, the side channel inference described above will consider font B as not installed. Table VI shows a pair of fonts that render certain characters in the same size.

2) Limitations of mobile fingerprint: Laperdrix et al. [10] demonstrated the effectiveness of Android device fingerprinting with 81% of unique mobile fingerprints in their dataset without leveraging plugin and font information. This result, however, is too positive for Safari, the most popular browser on iOS devices. As oppose to Android devices, a very limited number of iOS device models take up huge marker share. Here we use Safari on iPhone 7 as an example to demonstrate available attributes when fingerprinting iOS devices. We enumerate all available Safari fingerprinting attributes and their possible values in Table VII.

We now show that the tests introduced by Laperdrix is not effective on iOS. Firstly, all Safaris on iPhone 7 share the same value in attributes *Accept*, *Content encoding*, *List of plugins*, *Platform*, *Screen resolution*, *WebGL vendor*, *WebGL render* and *List of fonts*, so tests based on these attributes do not work anymore. Also, Safari on iPhone 7 does not support plugins or Flash, so the web attacker can not make use of information about plugins, fonts, or cross-validate attributes obtained from Flash APIs either. Tests based on canvas element collect information about three attributes: font, device and OS, hardware and OS. In common cases, users cannot change the default font in their iPhone if their iPhone is not jailbroken. Some may argue that applications like AnyFont support changing system font without jailbreaking. However, these apps only work with Word, Excel, PowerPoint and other apps [16]. Laperdrix's method also mentions that emoji is related to the version of iOS, but the "Smiling face with open mouth" emoji, which they used in their test remain the same throughout several versions of iOS. Besides, devices share the same device model, device and hardware are not useful here too. To sum up, the tests Laperdrix chose based on canvas are not effective anymore when fingerprinting Safaris on iPhone 7. These problems make cross-browser fingerprinting harder on iOS devices too.

Attribute	Fixed value
Accept	"`text/html,application/xhtml+xml, application/xml;q=0.9,*/*;q=0.8`"
Content encoding	"`gzip,deflate`"
List of plugins	N/A
Platform	"`iPhone`"
Screen resolution	$375 \times 667 \times 32$
WebGL vendor	"`Apple Inc.`"
WebGL render	"`Apple A10 GPU`"
List of fonts	Fixed

(a) Attributes with fixed values

Attribute	Number of variants
User agent	4
Cookies enabled	2
Use of local storage	2
Use of session storage	2
Do Not Track	2
Use of AdBlock	2
Content language	41
Timezone	24

(b) Attributes with variable values

TABLE VII: Fingerprinting attributes in Safari in iPhone 7

With the previous discussion, only seven attributes can be used as fingerprinting features on iOS devices. What makes fingerprinting even harder is that these attributes have low information entropy. *User agent* contains detailed version information of iOS, but Apple only released four official versions for iOS 10 by March 28th [18]. *Cookies enabled*, *Use of local storage*, *Use of session storage*, *Do Not Track* and *Use of AdBlock* only have two possible values. *Timezone* and *Content language* play a more important role as iPhone 7 supports 24 timezones and 41 languages. Therefore, even in the best case, a web attacker has only $4 \times 2 \times 2 \times 2 \times 24 \times 41 = 31\,488$ unique fingerprints for Safari on iPhone 7, but current number of iPhone 7 devices is more than 70 million.

To make the situation even harder, some of those attributes are closely related. Firstly, values of *Cookies enabled*, *Use of local storage*, *Use of session storage*, *Do Not Track* and *Use of AdBlock* are related as they are are determined by the user's privacy awareness. Secondly, mobile Safari treats cookies and local storage as the same thing. The values of *Use of local storage* and *Use of session storage* are same as

the value of *Cookies enabled* in public mode. Safari blocks *Use of local storage* and *Use of session storage* in private mode, no matter what the value of *Cookies enabled* is. Thirdly, both *Timezone* and *Content language* are related to the user's geographic information. People in the same time zone have a high probability to speak the same language. Due to all these reasons, a web attacker gets even fewer effective fingerprints than expected. Laperdrix's AmIUnique received good results mainly because they used a small database with $357\,692 \times 4.27\% = 15\,273$ iOS devices.

C. Attack

In this section, we focus on how to obtain correct fingerprinting features in the presence of a defender. To infer the real configuration of the browser, an attacker has to sample configuration for several times in various scenarios. The more samples an attacker gets, the more precise his inference would be. A defender(e.g. PriVaricator) perturbs browser configuration(e.g. font size, installed plugins) according to a probability distribution in private mode. Then the goal of a web attacker is to estimate the parameters of the distribution to reveal the true configuration. In this section, we assume a uniform distribution, which is used by PriVaricator, but the same methodology applies to any distribution. To defeat PriVaricator's defense, our attack web page measures the font sizes and installed plugins many times to estimate their true values.

The browser may defend against fingerprinting in private mode in three models with different granularities. In the first model, the defender creates and caches a random value once a user starts private mode and keep all these random values unchanged till the user quits private mode. In another word, browser provides a temporary identity for the user in the form of a fake fingerprint. In the second model, the defender creates and caches the random values separately for each session. This means that when a user's browser is in private mode, each session owns a different fingerprint. In the third model, the defender creates a fresh random values for each request in all sessions. In this case, even the same session may have various fingerprints. In the first model, a web attacker may use this temporary identity like a cookie, tracking users efficiently in a short time. In the second and the third model, we will show that a web attacker is capable of inferring the correct fingerprint of a browser given enough samples.

In the first model, a browser keeps the temporary identity unchanged throughout the whole private mode instance. Although web attackers can not link a user's sessions from different private mode instances, they may link a user's sessions in the same instance with this temporary identity. This identity thus serves as a temporary fingerprint. As we mentioned in II-B1, the attacker can detect whether the browser is in private mode via JavaScript APIs. He then knows the collected identity is not a fingerprint and will not store the temporary identity into fingerprinting database.

In the second model, a web attacker gets the same random values when sampling from one session but different random values when sampling across sessions. As we mentioned in II-B3, *Cookies*, *LocalStorage* and *IndexedDB* are shared in different sessions across tabs even if browsers are in private mode. Then, if these sessions use the same third party analysis, such as Google or Facebook, the analysis provider can still link random values in different sessions by comparing *Cookies*, *LocalStorage*, and *IndexedDB*. Therefore, if the users establish enough sessions, the attacker may get enough samples and infer the correct fingerprint. In this threat model, with enough sessions, the attacker can infer the true fingerprint and link the user's sessions in public mode and private mode together by this true fingerprint.

In the third model, a web attacker may constantly query for random values in one session. This indicates that for each session, the attacker can obtain enough samples in only a few seconds. In this threat model, the attacker has a great chance to link a user's sessions in public mode and private mode.

1) Estimate installed fonts: In our design, to estimate installed fonts, we choose a string that has different sizes when rendered in different fonts, just like the fingerprint attack we discussed in IV-A1. Then, our attack web page renders the string in different fonts, measures its sizes, and repeats this procedure for many times. For each font, our web page estimates the size of the string based on the measured sizes. If the estimated size is different from the size in default font, we consider the font to be installed; otherwise, the font is not installed.

A web attacker can infer the distribution by observing enough samples and choose the best hypothesis distribution. We assume that PriVaricator perturbs the font sizes according to uniform distribution and rounds the size to the nearest integer. For this distribution, the easiest method is estimating the true size with the average of all measured sizes. We find that a more accurate estimation is $\mathrm{round}((max + min)/2)$, where max and min are the maximum and minimum of measured sizes, respectively. Let $P_{(r,k)}$ be the probability that our estimation is the true size, r be the range of perturbed font sizes, k be the number of measured sizes. Here we give a lower bound of $P_{(r,k)}$ by calculating the probability of both the maximum and minimum values of the distribution are sampled.

$$P_{(r,k)} = \frac{r^k - 2 * (r-1)^k + (r-2)^k}{r^k} \tag{1}$$

2) Estimate installed plugins: To estimate the installed plugins, since PriVaricator randomly omits 30% of installed plugins, we query the installed plugins many times and take the union of the reported plugins. Let $P_{(m,h,k)}$ be the probability that this set contains all the installed plugins, where h is the probability that the browser omits a plugin in its report, m is the total number of plugins, and k is the number of times the attack page queries the browser for installed plugins.

$$P_{(m,h,k)} = \left[1 - h^k\right]^m \tag{2}$$

D. Evaluation

As we could not access the source code of PriVricator, we simulated how it perturbed font sizes and the set of plugins in Chrome 52.0 on Windows 8.1.

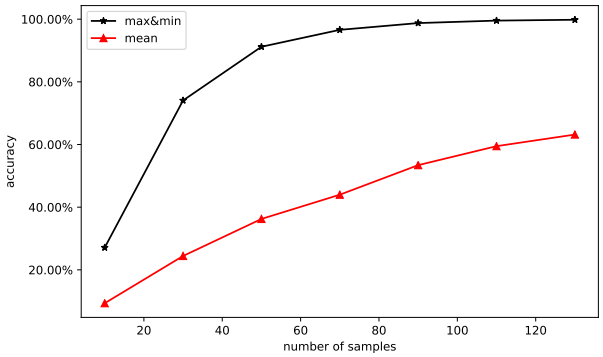


Fig. 2: Accuracy of font size estimation. *max&min* shows the result of estimating the true size by rounding the average of maximum and minimum measured size. *mean* shows the result of estimating the true size by taking the average of all measured sizes.

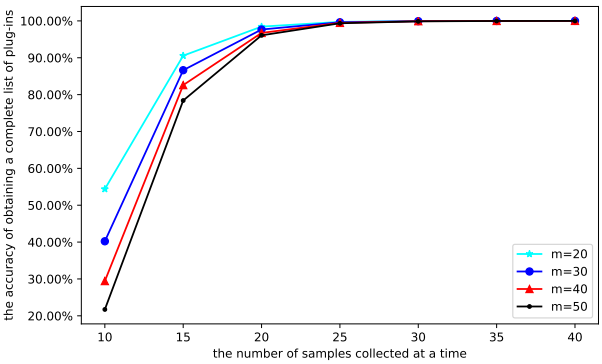


Fig. 3: When the browser randomly hides 70% plugins, how much percentage of all the plugins can our attack recover?

1) Fonts: We use the discrete uniform distribution as an example. The range of random value is from $\text{round}(0.95 \times true\_value)$ to $\text{round}(1.05 \times true\_value)$. We choose $\text{round}(0.5 \times (random\_max + random\_min))$ as our estimation of the true value.

We chose 78 Latin fonts, 20 Chinese fonts, and 1 non-existent font. The browser would substitute unavailable fonts with its default font. We chose the test string `milj_^`, because it has distinct width and height for each of those fonts.

We created a test page that rendered the test string in a specified font and measured its size (width and height) repeatedly . The measured sizes were the true sizes perturbed uniformly at random by $\pm 5\%$ followed by rounding, as implemented by PriVricator. For each font, we rendered the string k times and estimated the sizes of the font with the method described in IV-C1.

Figure 2 shows the precision of our estimation with regard to k, the number of times that we measured the size of the test string. Our method may not be the best for a uniform distribution, but estimating the true size by $\text{round}((max + min)/2)$ is better than estimating the true size by taking the average of measured sizes. With the first method, when $k = 90$, the precision is 98.75%, when $k = 130$, the precision is 99.81%. With the second method, when $k = 90$, the precision is 53.41%, when $k = 130$, the precision is 63.17%.

2) Plugins: Figure 3 shows the precision of our plugin estimation attack. Our result is based on 100 thousand test runs for each scenario. With only 25 measures, we were able to get the complete list of 20 plugins with a precision of 99.73% and get the complete list of 50 plugins with a precision of 99.35%. When we measured 40 times, we were able to get the complete list of 50 plugins with more than 99.99% precision. The result suggests that our method is highly effective when used to infer installed plugins.

V. Related Work

A. How to Detect Private Mode

Since Firefox 20, the *privatebrowsingmode* API is deprecated and will likely to be removed in Firefox 21 [15]. Plugin developers can only use *NPNVprivateModeBool* before the end of 2016 [17]. After that, Mozilla will stop the support for most NPAPI plugins in Firefox [3]. However, there are still side-channel methods to detect whether the browser is in private mode for Chrome, Firefox, Safari, and Edge [6]. These browsers block some JavaScript API in private mode, so attackers can take advantage of this feature to detect whether user's browser is in private mode.

B. Browser's Fingerprint

In 2010, Eckersley et al. launched the Panopticlick website to collect device-specific information via a script which runs in the browser [8]. They showed that the list of fonts and the list of plugins were the most distinguishable attributes. Some follow-up efforts [1, 9, 13, 11] showed that tracking companies started using fingerprint and developers provided open source fingerprinting libraries. The work of Laperdrix et al. [10] performed the first large-scale study to examine the validity of browser fingerprinting in current web environment on both desktop and mobile platforms. The work of Nikiforakis et al. reported the design, implementation and deployment of FPDetective, a framework for the detection and analysis of web-based fingerprints.

C. Cross-browser Fingerprint

Cross-browser fingerprinting is a browser fingerprinting technique that tracks users across different browsers on the same machine. AmIUnique [10] described WebGL as "too brittle and unreliable". However, this accusation is not valid as it selected random WebGL tasks and left variables such as canvas size and anti-aliasing unrestricted. Cao et al. [19] proposed a cross-browser fingerprinting technique based on features from OS and hardware, e.g., graphics card, CPU, audio stack, and installed writing scripts. Their result is than AmIUnique for single-browser fingerprinting, and better than

Boda et al. [5] for cross-browser fingerprinting. Das et al. [7] developed a highly accurate fingerprinting mechanism that combined multiple motion sensors and made use of inaudible audio stimulation to improve detection. Manufacture imperfection gave each sensor unique characteristics in the signal they produced. Such characteristics can serve as fingerprinting features and in turn be used to track users across visits. A shortage of their work is the test set only contains 30 smartphones.

D. Deceive Fingerprinters

In a recent work, Besson et al. [4] showed how to enforce knowledge threshold security using a more flexible mechanism based on randomization and how to synthesize a randomization mechanism that defines the distribution of configurations for each user. In 2015, Nikiforakis et al. [12] argued that the real problem with web tracking is not $uniqueness$ but $linkability$, the ability to connect the same fingerprint across sessions. They used randomization to break linkability between different sessions. To attack their methodology, we use statistical methods to estimate the true values of a browser's configurations. Our experiments showed that our attack was easy and effective.

VI. Conclusion

We performed the first study comparing private browsing modes between popular desktop and mobile browsers and found many inconsistencies. Some inconsistencies are due to implementation flaws, and we showed how they compromise user privacy that should be protected by private mode. However, some inconsistencies are due to the tradeoff between security and privacy. But even if a browser implements private mode correctly, a web attacker may still link a user's sessions by fingerprinting the browser. We proposed an attack on a previous defense against browser fingerprinting and demonstrated that our attack was easy and highly effective.

VII. References

[1] Gunes Acar et al. "The web never forgets: Persistent tracking mechanisms in the wild". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 674–689.

[2] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. "An Analysis of Private Browsing Modes in Modern Browsers." In: *USENIX Security Symposium*. 2010, pp. 79–94.

[3] Smedberg Benjamin. *NPAPI Plugins in Firefox*. https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox/.

[4] Frdric Besson, Nataliia Bielova, and Thomas Jensen. "Browser randomisation against fingerprinting: A quantitative information flow approach". In: *Nordic Conference on Secure IT Systems*. Springer. 2014, pp. 181–196.

[5] Kroly Boda, dm Mt Fldes, Gbor Gyrgy Gulys, and Sndor Imre. "User tracking on the web via cross-browser fingerprinting". In: *Nordic Conference on Secure IT Systems*. Springer. 2011, pp. 31–46.

[6] cou929. *Detect private browsing mode (InPrivate Browsing or Incognito)*. https://gist.github.com/cou929/7973956.

[7] Anupam Das, Nikita Borisov, and Matthew Caesar. "Tracking mobile web users through motion sensors: Attacks and defenses". In: *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS)*. 2016.

[8] Peter Eckersley. "How unique is your web browser?" In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2010, pp. 1–18.

[9] Yosifon Elad. *Modern & flexible browser fingerprinting library,* https://github.com/Valve/fingerprintjs2.

[10] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. "Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints". In: *37th IEEE Symposium on Security and Privacy (S&P 2016)*. 2016.

[11] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. "Fingerprinting information in JavaScript implementations". In: *Proceedings of W2SP* 2 (2011), pp. 180–193.

[12] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. "Privaricator: Deceiving fingerprinters with little white lies". In: *Proceedings of the 24th International Conference on World Wide Web*. ACM. 2015, pp. 820–830.

[13] Nick Nikiforakis et al. "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting". In: *Security and privacy (SP), 2013 IEEE symposium on*. IEEE. 2013, pp. 541–555.

[14] Greenhalgh Sam. *HSTS Super Cookies*. http://www.radicalresearch.co.uk/lab/hstssupercookies.

[15] teoli, ignisvulpis, Sheppy, Ehsan, and Dao. *Supporting private browsing mode*. https://developer.mozilla.org/en-US/docs/Archive/Mozilla/Supporting_private_browsing_mode.

[16] Carlos Vega. *How to install fonts on iPad or iPhone with AnyFont*. http://www.digitaltrends.com/mobile/how-to-install-fonts-on-ipad-or-iphone/.

[17] Sheppy wbamberg. *Supporting private browsing in plugins*. https://developer.mozilla.org/en-US/Add-ons/Plugins/Supporting_private_browsing_in_plugins.

[18] Wikipedia. *How to install fonts on iPad or iPhone with AnyFont*. https://en.wikipedia.org/wiki/IOS_10.

[19] Cao Yinzhi, Li Song, and Wijmans Erik. "(Cross-)Browser Fingerprinting via OS and Hardware Level Features". In: *Proc. of NDSS17* (2017).