Journal of Information Processing Vol.31 620–642 (Sep. 2023)

[DOI: 10.2197/ipsjjip.31.620]

Regular Paper

Understanding the Inconsistencies in the Permissions Mechanism of Web Browsers

Kazuki Nomoto[1,a) Takuya Watanabe[2,b) Eitaro Shioji[2,c) Mitsuaki Akiyama[2,d) Tatsuya Mori[1,3,4,e)

Received: December 13, 2022, Accepted: March 7, 2023

Abstract: Modern Web services provide advanced features by utilizing hardware resources on the user's device. Web browsers implement a user consent-based permission model to protect user privacy. In this study, we developed Permium, a web browser analysis framework that automatically analyzes the behavior of permission mechanisms implemented by various browsers. We systematically studied the behavior of permission mechanisms for 22 major browser implementations running on five different operating systems. We found fragmented implementations. Implementations between browsers running on different operating systems are not always identical. We determined that implementation inconsistencies could lead to privacy risks. We identified gaps between browser permission implementations and user perceptions from the user study corresponding to the analyses using Permium. Based on the implementation inconsistencies, we developed two proof-of-concept attacks and evaluated their feasibility. The first attack uses permission information to secretly track the user. The second attack aims to create a situation in which the user cannot correctly determine the origin of the permission request and the user mistakenly grants permission. Finally, we clarify the technical issues that must be standardized in privacy mechanisms and provide recommendations to OS/browser vendors to mitigate the threats identified in this study.

Keywords: Web, Browser, Permission, Privacy

1. Introduction

Modern Web services can provide a richer user experience, such as online conferencing and searching based on user location, if they are given permission to use hardware resources such as microphones and GPSs in a user's device. Permission mechanisms are widely deployed in web browsers to control access to these resources by websites and to protect user privacy. Users can control which hardware devices and browser functions they allow websites to use.

Major mobile operating systems (OSs), such as Android and iOS employ a permission mechanism as a means of exerting control over the access of applications to hardware and other resources [2], [3]. Users of mobile OSs can utilize the permission mechanism to fine-tune the available resources on a per-application basis. In the security research community, many studies have been conducted on permission mechanisms in mobile OSs, including large-scale measurements [4], [5], [6], user perception [7], [8], [9], [10], unauthorized privilege acquisition [11], [12], and the proposals of new permission mechanisms [13], [14], [15], [16]. While mobile OS permission mechanisms have been studied by many researchers, to the best of our knowledge, no studies have systematically analyzed the permission mechanisms of web browsers.

The behavior of the permission mechanism in Web browsers ("Web Permission") varies widely across OSs and browsers because it depends on the browser vendor's implementation. The implementations of Web Permission differ depending on the type of API and browser. Web Permission was developed for each API (MediaDevice API, Geolocation API, etc.) that utilizes the functions of individual devices such as cameras and GPSs [17]. For this reason, each API has its own specification for the permission mechanism, and these specifications are not standardized. Such implementation fragmentation leads to user confusion and gives opportunities for an attacker to exploit Web Permission. To avoid fragmentation, "Permissions API" was proposed to manage various permissions in a unified manner [18]. The Permissions API offers a new feature that allows websites to query the permission state without requesting permission. However, as of 2022, support for the Permissions API varies from browser to browser, and fragmented implementations are still widely used.

This study aims to conduct a large-scale study of web permission implementations to identify implementation inconsistencies across browsers and OSs, as well as vulnerabilities that lead to

[1] Waseda University, Shinjuku, Tokyo 169–8555, Japan
[2] NTT Social Informatics Laboratories, Musashino, Tokyo 180–8585, Japan
[3] National Institute of Information and Communications Technology, Koganei, Tokyo 184–8795, Japan
[4] RIKEN Center for Advanced Intelligence Project, Chuo, Tokyo 103–0027, Japan
a) nomotokazuki@nsl.cs.waseda.ac.jp
b) takuya.watanabe.yf@hco.ntt.co.jp
c) eitaro.shioji.es@hco.ntt.co.jp
d) akiyama@ieee.org
e) mori@nsl.cs.waseda.ac.jp

This paper is an extended version of the paper presented at NDSS'23 [1].

privacy risks. To achieve this, we developed Permium, a framework for the systematic analysis of the behavior of browser permission mechanisms. Permium is a framework that automatically analyzes a browser's web permission implementation according to predefined test scenarios and autopilots the browser as if operated by a human. Permium supports 22 different browser implementations running on 5 different OSs, including mobile and desktop. The share of these OSs and browsers covers 94% and 89% of desktop and mobile browser users, respectively [19]. This study targets Microphone, Camera, and Geolocation as sensitive permissions and Notification as the most frequently used permission. As a result of our extensive measurement study, we found 191 implementation inconsistencies in 22 different browsers that could pose a threat to user privacy. Those inconsistencies include cases where the selected web permission state was shared between normal and private browsing modes, where permission state was retained after clearing the web browser data, and where permission request dialogs for tabs running in the background overlaid the foreground tab, which can confuse users. We also perform user studies to identify gaps between user expectations and actual browser implementations.

In addition, we propose two proof-of-concept attacks that leverage the implementation inconsistencies of the web permission mechanism identified in our measurement study. We also evaluate the feasibility of the attacks. The first attack uses the permission information collected by exploiting the inconsistency to secretly track users. The attacker can distinguish users visiting the site using only the permission information, without using cookies or browser fingerprints. The other attack targets users that inadvertently granted permissions to malicious sites by creating a situation in which they cannot correctly determine the origin of the permission request. In a user study of 99 participants, we determined that this attack succeeds more than 55% of the time.

The contributions of this paper are summarized as follows.

- This is the first systematic and extensive study on the behavior of web permission implementations.
- We developed Permium, a framework for automatically analyzing web permission behavior, which uses a high-level abstraction to enable the operations of various browsers using common methods regardless of OS or browser type.
- As a result of our measurement study using Permium, we found 191 inconsistencies in the browser implementation that could pose privacy risks such as when a user grants or denies permission, the browser cannot properly reflect in some cases.
- We conducted a user study designed to understand users' expectations of browser permission implementations. We identified gaps between actual browser implementations and user perceptions.
- We proposed two proof-of-concept attacks, a permission-based user tracking attack and a permission-based phishing attack, which are realized by combining the web permission implementation inconsistencies revealed in the measurement study.
- We provided recommendations to stakeholders. We developed countermeasure techniques to mitigate privacy risks due to implementation inconsistencies.

2. Background

This section describes the two main mechanisms for protecting the privacy of web browser users: the permission mechanism and private browsing modes. Note that the permission mechanism is the main target of this study. Private browsing mode is a mechanism that enables users to explicitly protect their privacy while browsing.

2.1 Overview of the Permission Mechanism for Web Browsers

A website that aims to leverage various resources on a user device, including hardware devices such as a camera, microphone, GPS, etc., requests permission from the user to access the resources. In the following, we present a sequence of processes that range from the permission request from the website to the permission granted by the user.

1) A website requests permission from the user's browser to access hardware resources on the device using Web APIs such as the Geolocation API [20], which provides the user's location, and the MediaDevices API [21], which provides access to the connected microphones and cameras.

2) The web browser shows a permission request prompt on the user's screen.

3) The user grants or denies the request, or ignores the request by closing the prompt window.

4) The web browser allows the website to use the hardware resources corresponding to the permission when the permission state is granted.

In web browsers, the permission is managed per "origin" basis, whereas the permission in the mobile OS is managed per application basis [22]. In the context of the web, "origin" is defined by the 3-tuple: scheme, hostname, and port number, which are present in the URL that points to a web content [23]. The origin is a fundamental unit used to determine the identity of a website and is widely adopted in security protection mechanisms such as in the Same-Origin Policy [24].

Typically, the web browser maintains the permission state, either granted or denied, for each origin [*1]. When an origin website requests permission for the second time, the permission is automatically granted or denied based on the permission state selected by the user and kept by the browser from the previous time. Formally, the permission has the following three states: "prompt," "granted," and "denied." Here, "prompt" is a state in which the user has not selected a permission state. The default state of the permission state is "prompt." The "granted"/"denied" is the state when the user grants or denies the permission request, respectively. The web browser provides access to a hardware resource on the website only when the permission state for that resource is granted to the website (origin).

Permissions API. Historically, web permission requests have been provided by different APIs for each function. For example, the MediaDevices API is used to request permission to access the camera, and the Geolocation API is used to request permission to

[*1] As shown in Section 4.3, the rules for keeping the permission state are intrinsic to the browser implementation.

Table 1 Obtaining permission state using Permissions API.

Tab Status	Chrome					Firefox					Edge					Brave					Safari	
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Foreground	●	●	●	●	○	N	N	N	N	○	●	●	●	●	○	●	●	●	●	○	N†	○
Background	●	●	●	●	○	N	N	N	○	○	●	●	●	●	○	●	●	●	●	○	N†	○

● : All permission states can be obtained with Permissions API. N : Notification permission states can be obtained with Permissions API.
○ : Not all permission states are supported by Permissions API.
† : Not supported by Permissions API, but Notification permission state can be obtained by Notification API
W: Windows, L: Linux, M: macOS, A: Android, i: iOS

access the GPS [17]. The Permissions API is currently being standardized as a mechanism to unify these fragmented APIs [25]. Since 2022, most websites request permission using legacy APIs and not the Permissions API because no web browser currently allows websites to request permission using the Permissions API by default. We expect the Permissions API to be widely adopted for permission requests in the future.

Currently, the progress of implementing the Permissions API varies between web browsers [17]. Chrome, Edge, and Opera have enabled permission requests using the Permissions API as an experimental feature. Firefox and Safari have not adopted the permission request functionality using the Permissions API. The Permissions API provides a new feature that allows websites to query the permission state without requesting permission. Although this feature is expected to improve the user experience [26], the risk of using the feature as a browser fingerprinting technique is also being discussed [25].

We investigate whether the Permissions API for querying the permission status works in each case where the tab is in the foreground or background. For browsers that do not support the Permissions API, we use the Notification API to check if it is possible to obtain the permission state without notifying the user. The results are shown in **Table 1**. We found that it is possible to obtain the permission states without notifying the user by using the Permissions API or Notification API in all browsers except iOS.

2.2 Private Browsing Modes

Private browsing modes are mechanisms that enable users to perform private web browsing without using the personal browser environment in which cookies and account information are stored. These can therefore be expected to protect user privacy [27]. The major browsers support private browsing modes. Because technical specifications for private browsing modes have not been standardized, browser vendors have implemented their own features according to the principles of the market [28]. The name of this feature also varies between browsers; that is, it is called "Incognito mode" in Chrome and "InPrivate window" in Edge [29], [30]. In this paper, we use the term "private browsing modes," as used in the W3C [28].

3. Permium Framework

Permium is a unified framework that automatically analyzes browsers' web permission implementations according to predefined test scenarios and autopilots the browsers as if operated by a human, e.g., clicking a button and inputting text, which is required to open a URL. We designed the code to define test scenarios generically so these can be run even on various OSs and browsers.

3.1 Overview of the Framework

Figure 1 presents a workflow of the Permium framework.

①: The analyst creates a test scenario to analyze the behavior of permission implementation in a web browser and send it to *Commander,* which consists of two functionalities: *Wrapper* and *Manipulator*, which are described in the following.

②: Next, *Commander* uses *Wrapper* to convert the test scenario into specific operations for each OS/browser. *Manipulator* connects to each OS/browser device using remote control schemes, such as Remote Desktop Protocol (RDP) or Virtual Network Computing (VNC). Once the connection is established, *Manipulator* controls the web browser by performing the operations sent from *Wrapper*, e.g., accessing web pages for testing, clicking a button, and restarting the browser. The web server sends the permission state and access status to *Commander*. The access status is used to execute the next event, such as a permission request, permission state query, or page transition.

③: *Commander* logs information involving the permission state, dialog display status, etc.

④: Finally, the analyst analyzes the logs to clarify the permission handling for each browser in each test scenario.

The following section describes the technical components that compose the Permium framework.

3.2 Components of the Permium Framework

In the following, we describe the components of the Permium framework shown in Fig. 1.

Test Scenario. It is necessary that web browsers be operated in a consistent manner to assess the behavior of permission implementations under a variety of conditions, while considering the detailed conditions such as the order of user operations and use of private browsing modes. In the Permium framework, the browser operating procedure is defined as a coded test scenario. The code for test scenarios uses abstract methods that are independent of differences in OS and browsers. The abstract methods are provided by Wrapper, which will be described later. An example code for a test scenario is shown in Listing 1, which describes a series of operating procedures in which a web browser accesses a test page and grants the permission.

Wrapper. In general, the UI of each browser is different, and the way to operate the same operation differs greatly among browsers and operating systems. The purpose of the Wrapper is to provide a unified method that absorbs these differences. Analysts using the Permium framework write a common, browser-independent operation method in the test scenario code. The operation

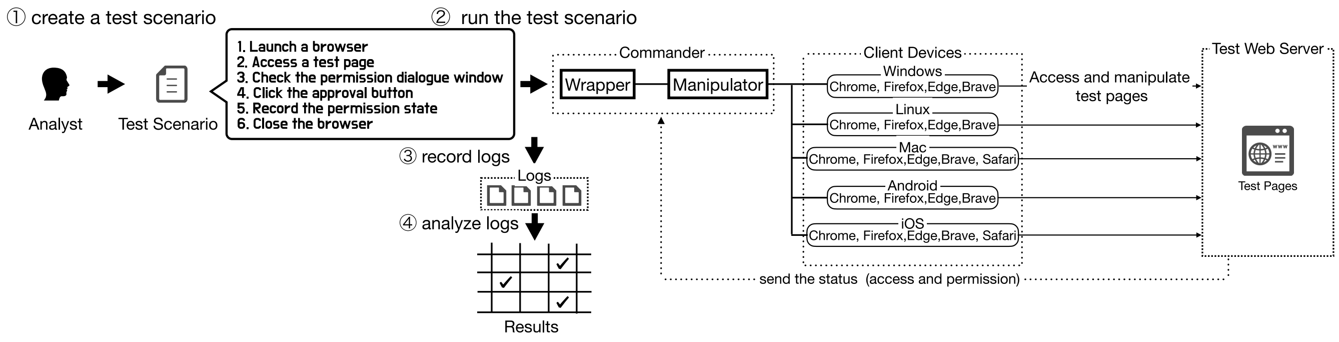

① create a test scenario
② run the test scenario
Analyst
Test Scenario
1. Launch a browser
2. Access a test page
3. Check the permission dialogue window
4. Click the approval button
5. Record the permission state
6. Close the browser
Commander
Wrapper
Manipulator
Client Devices
Windows
Chrome, Firefox,Edge,Brave
Linux
Chrome, Firefox,Edge,Brave
Mac
Chrome, Firefox,Edge,Brave, Safari
Android
Chrome, Firefox,Edge,Brave
iOS
Chrome, Firefox,Edge,Brave, Safari
Access and manipulate test pages
Test Web Server
Test Pages
③ record logs
Logs
④ analyze logs
Results
send the status (access and permission)

Fig. 1 Overview of the PERMIUM Framework.

```
url = "https://example.com"
mode = "normal"
platform.startBrowser(browserName)
browser.goToUrl(url,platformName,"normal")
browser.requestPermission(platformName, mode)
if browser.checkPermissionDialogue(platformName,
    mode):
    browser.clickAllow(platformName,mode)
browser.close(platformName, mode)
```

Liting 1 A Test Scenario

Table 2 Manipulator connection methods for each OS.

OS	Connection methods
Windows	Remote Desktop Protocol
Linux	Virtual Network Computing
macOS	Virtual Network Computing
Android	Android Debug Bridge + scrcpy
iOS	Virtual Network Computing + screendump

methods are divided into three types of operations: basic operations, OS-related operations, and browser-related operations. Basic operations include mouse operation, key input for feeding URLs, communication with the server, and matching of operation elements. OS-related operations include starting a browser and opening and closing windows. Browser-related operations include opening new tabs and page transitions in each browser.

Manipulator. Manipulator receives the operating instructions specific to each OS/web browser generated by Wrapper based on the test scenarios. Manipulator then connects to each OS/web browser and operates them according to the received operation instructions. Manipulator connects to each device with the methods shown in **Table 2** to acquire information from the screen and operate the mouse and keyboard.

Client Devices. A client device corresponds to a combination of OS and browser, e.g., macOS Chrome, and is operated by Manipulator. The web browsers are installed on each OS of the client device, and a microphone and camera are connected to the device. In this study, all devices were physical machines. Note that virtual machines can also be used. Appendix A.1 details the OSs and web browsers used in this study.

Test Web Server. A test web server provides web pages that request the permission used in the test scenario. Specifically, when the server receives an HTTPS Request sent by a client device's browser, it responds with a page that contains JavaScript requesting the permission. We implement a test web server that provides behaviors used in test scenarios, such as requesting permission with arbitrary timing and creating arbitrary page transitions. The test web server also has the function of logging the response codes and the permission state of the web browser.

3.3 Technical Challenges of the PERMIUM Framework

Implementing the PERMIUM framework involves the following two technical challenges. The first is realizing the measurement of browser permission implementations across a wide range of browsers and OSs. Few measurement studies have been conducted that cover both desktop and mobile browsers. Popular automation frameworks such as Selenium, Puppeteer, and Playwright were designed to use APIs, such as WebDriver and DevTools protocols, provided by the web browser [31], [32], [33]. For iOS, existing browser measurement frameworks do not support autopilot for browsers other than Safari [34], [35]. We also note that existing frameworks cannot change settings that affect the permission state or operate permission request prompts. For example, it is impossible to click a button on the permission request prompt or change the settings of the Safari browser on iOS. Our PERMIUM implementation successfully resolves these issues. The second challenge is providing analysts with the abstracted operating methods that can absorb the browser UI differences. Our implementation enables us to launch 22 different web browser operations across desktop/mobile OSs using the abstracted operation methods. Using this approach, analysts can work with the 22 different browsers by simply writing a test scenario code.

The details of the PERMIUM framework (for example, how the framework clicks buttons on the web browser and how it obtains the permission state of the web browser) are described in Appendix A.1.

4. Measurement Study

In this section, we report the results of the study of the measurement of permission behaviors for the 22 Web browsers using the PERMIUM framework. We focus on the four primary permission types: Microphone, Camera, Geolocation, and Notification.

4.1 Overview

We measured the behaviors of permission implementations based on the following six test scenarios:

T_1: *Is the permission state set by a user (granted or denied) correctly reflected by the browser?*

T_2: *Is the permission state set by a user persistent?*

Table 3 Reflection of the permission state set by user (Scenario T_1).

Browsing Mode	Chrome					Firefox					Edge					Brave					Safari	
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal	○	○	○	○	N	○	○	○	○	N	○	○	○	○	N	○	○	○	○	N	○	N
Private	N	N	N	N	N	†	†	†	○	N	N	N	N	N	N	N	N	N	N	N	○	N

○ : Permission state set by a user is correctly reflected for all permission resources (Camera, Microphone, Geolocation, and Notification).
N : Notification permission is unsupported. Other permissions are correctly supported.
†: Notification permission is supported, but the state "denied" is not correctly reflected. Other permissions are correctly supported.
W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table 4 Persistence of the permission state (Scenario T_2).

Browsing mode	Permission state	Chrome					Firefox					Edge					Brave					Safari	
		W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal	Granted	●	●	●	●	G	N	N	N	○	G	●	●	●	●	G	○	○	○	○	G	N	G
	Denied	●	●	●	●	G	N	N	N	○	G	●	●	●	●	G	○	○	○	○	G	N	G
Private	Granted	○	○	○	○	G	○	○	N	○	G	○	○	○	○	G	○	○	○	○	G	N	○
	Denied	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	N	○

● : Permission state persists for all supported resources. ○ : Permission state does not persist for all supported resources.
N : Notification permission state persists. G : Geolocation permission state persists. W: Windows, L: Linux, M: macOS, A: Android, i: iOS

T_3: *Is the permission state isolated between the browsing modes?*
T_4: *Does clearing browser data and settings erase the permission state?*
T_5: *How is the permission state set when the prompt is ignored?*
T_6: *Does a permission request from a tab running in the background pop up in front?*

Scenarios T_1–T_4 aim to study the basic functionality of permission mechanisms, whereas scenarios T_5 and T_6 aim to study the behaviors of permission implementations under relatively complex conditions. In addition to the measurement results of the six test scenarios, we derived several implications from the findings. Section 8.1 discusses solutions against privacy risks determined in our study.

4.2 Is the Permission State Set by a User (Granted or Denied) Correctly Reflected by the Browser? (T_1)

In test scenario T_1, we investigate how browsers set the permission state when a user grants or denies permission requests.

Measurement Results. **Table 3** summarizes the measurement results. First, in normal browsing modes, the permission state set by the user was correctly reflected by all browsers, except iOS, which does not support Notification permissions [36]. In private browsing modes, most OSs and web browsers did not support the Notification permission. The exceptions were Safari on macOS, where all permission states were correctly reflected, and Firefox on Desktop OS, where the denied Notification permission was not correctly reflected by the browser.

Implications. The differences in the permission handling depending on the OS, web browser, and browsing modes can contribute to establish browser fingerprinting. The difference in functionality between normal and private browsing modes is not itself a problem however a mechanism is necessary for hiding these differences from the website and preventing them from being used for browser fingerprinting. Furthermore, in Firefox private browsing modes, the permission states set by the user should be reflected properly.

4.3 Is the Permission State Set by a User Persistent? (T_2)

In test scenario T_2, we investigate whether the permission states set by the user persists after the web browser is closed. To this end, a client device first grants or denies a permission prompt. Then, we restart the web browser. Finally, we investigate the permission state on the browser when it receives a permission request from the same origin website.

Measurement Results. **Table 4** summarizes the measurement results. In normal browsing modes, the permission state persisted in many cases. In private browsing modes, we might expect that a granted permission state would not be persistent or namely, the state would revert to "prompt" when the private browsing mode is closed. However, we found several exceptions. In general, the conditions under which permission states persisted and the permission types that persisted vary depending on the browsers.

In normal browsing modes of Chrome and Edge, all four analyzed permissions persisted, except for iOS. The behavior of the iOS web browser was different from other browsers; with the exception of iOS Safari in private browsing modes, only the Geolocation permission persisted when a user grants the permission at least twice.

Implications. As observed, the conditions and environments under which permission states persist are not obvious or transparent to users. The unclear persistence of the permission state can pose an unintended privacy risk to users. For example, some users may grant permission to provide their location on the website at the office or cafe but do not want to provide their location to the website at home. Table 4 suggests that many environments and conditions do not meet these user expectations.

If a user sets permission for a website and that permission state persists, it means that the website may be able to determine if the user has previously visited that website. This could potentially raise privacy concerns for some users. We have identified cases in which the user's access history to a particular website can be inferred, even in private browsing mode, if the user has previously denied the permission due to scripts installed on the website.

Note that the permission state persisting in private browsing modes violates the official documents published by web browser

Table 5 Sharing of permission state between different browsing modes (Scenario T_3)

Order	Permission state	Tab Status	Chrome					Firefox					Edge					Brave					Safari	
			W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal → Private	Granted	Open	○	○	○	○	G	○	○	○	N	G	○	○	○	○	G	○	○	○	○	G	N	○
	Granted	Closed	○	○	○	○	G	○	○	○	N	G	○	○	○	○	G	○	○	○	○	G	N	○
Normal → Private	Denied	Open	●	●	●	●	G	○	○	○	N	G	●	●	●	●	G	●	●	●	●	G	N	○
	Denied	Closed	●	●	●	●	G	○	○	○	N	G	●	●	●	●	G	○	○	○	○	G	N	○
Private → Normal	Granted	Open	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	N	○
	Granted	Closed	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	N	○
Private → Normal	Denied	Open	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	N	○
	Denied	Closed	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	○	○	○	○	G	N	○

● : Permission state of all resources is shared. ○ : Permission state of all resources is not shared.
N : Notification permission state is shared. G : Geolocation permission state is shared. W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table 6 Permission state after deleting browser data (Scenario T_4)

Mode	Chrome					Firefox					Edge					Brave					Safari	
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal	○	○	○	○	G	NMC_1	NMC_1	NMC_1	○	G	○	○	○	○	G	○	○	○	○	G	○	○
Private	●	●	●	●	G	NMC_2	NMC_2	NMC_2	○	G	●	●	●	●	G	●	●	●	●	G	○	○

○ : Permission states for all resources are erased. ● : Permission states for all resources are retained.
G : Permission state (granted/denied) for Geolocation is retained.
NMC_1 : Permission states (granted/denied) for Notification and permission states (granted) for Microphone and Camera are retained.
NMC_2 : Permission states (granted) for Notification, Microphone, and Camera are retained.
W: Windows, L: Linux, M: macOS, A: Android, i: iOS

vendors [29], [30], [37], [38], [39]. Inconsistencies between the specifications described in the documents and actual behaviors create user confusion and unintended risks.

4.4 Is the Permission State Isolated Between the Browsing Modes? (T_3)

In test scenario T_3, we investigate whether the permission state set by a user is isolated or shared between the normal and private browsing modes. We check the permission state in one browsing mode after the permission state was set in another browsing mode. The analysis procedure is as follows. First, in normal browsing mode, a client device accesses website A, which requests permission. The client device grants or denies the request. Next, the browser of the client device is switched to private browsing mode. The client device accesses website A again, and we analyze the permission state. Similarly, we investigate changes in the permission state when the permission state is first set in private browsing mode and then switched to normal browsing mode. We investigate two cases: when a tab of website A is closed (Closed) after being accessed in the first browsing mode, and when the tab is not closed (Open).

Measurement Results. **Table 5** summarizes the measurement results. To our surprise, the permission state was not always isolated between browsing modes in many browsers. In general, permission states set in private browsing modes were rarely shared with normal browsing modes, but the iOS web browser (WebKit) shares the Geolocation permission state, and Safari on macOS shares the Notification permission state. In Chrome, Edge, and Brave, the denied permission state set in normal browsing modes was reflected in private browsing modes. In Brave, permission was not shared with private browsing modes when the tab was closed in normal browsing modes. Chrome and Edge shared denied permissions regardless of the Open or Close state of the tab.

Implications. As mentioned in Section 2.2, because private browsing modes lack a standardized specification, behavior varies between browser vendor implementations. This situation creates the risk of confusing the users. For example, consider a case in which the Geolocation permission was granted in normal browsing mode and then unintentionally reflected in the private browsing mode. The user may expect that their private information was not sent to the website when using the private browsing mode. However, in reality, the website can acquire the user's location information. Unintentional location leakage can pose the threat of linking accounts used in each mode according to the consistency of location and IP address. In particular, sharing the state of granted permission between browsing modes can lead to a high risk of mismatching user expectations with web browser behavior.

4.5 Does Clearing Browser Data and Settings Erase the Permission State? (T_4)

In test scenario T_4, we investigate whether the permission state is erased when a user clears stored data such as history, settings, and cookies, from their browser. The analysis procedure is as follows. First, the web browser accesses the website requesting permission and grants or denies the permission request. To clear browser data, the Clear Data mechanism installed in each browser or more specifically the feature for deleting data and settings, is used. The web browser accesses the same website, and we check whether the permission state was affected.

Measurement Results. **Table 6** summarizes the measurement results. In normal browsing modes, at least one permission state was retained by all browsers except Safari, after clearing the web browser data. Interestingly, in private browsing modes, the permission state was not erased in most browsers, except Safari. Although inconclusive, we assume that the measurement result reflects the web browser storing its configuration data in a different location (such as memory) in private browsing modes rather than in normal browsing modes; these are therefore unaffected by the clear-data method, which erases data stored in the storage space.

Table 7 Results of automatically setting permission state to denied by ignoring the prompt multiple times (Scenario T_5).

Mode	Tab Status	Chrome					Firefox					Edge					Brave					Safari	
		W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal	Foreground	●	●	●	●	–	○	○	○	○	–	●	●	●	●	–	●	●	●	●	–	●	–
	Background	MCN	MCN	MCN	N	○	○	○	○	○	◐	MCN	MCN	MCN	N	○	MCN	MCN	MCN	N	◐	N†	○
Private	Foreground	●	●	●	●	–	○	○	○	○	–	●	●	●	●	–	●	●	●	●	–	●	–
	Background	MC	MC	MC	○	○	○	○	○	○	◐	MC	MC	MC	○	○	MC	MC	MC	○	◐	N†	○

● : Permission state is automatically set to denied for all resources. ○ : Permission state is not automatically set to denied for all resources.
M/C/N : Microphone/Camera/Notification permission state is automatically set to denied.
◐ : No results (Permission request dialog overlay display occurs and dialog cannot be ignored.). – : Analysis inapplicable.
† : Permission request dialog is overlaid, and state is automatically set to denied. W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table 8 Display of a permission request dialog from a background tab (Scenario T_6).

Chrome					Firefox					Edge					Brave					Safari	
W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
○	○	○	○	●§	○	○	○	○	●†	○	○	○	○	○	○	○	○	○	●†	●‡	○

○ : The dialog is not displayed on top of the foreground tab. ● : The dialog is displayed on top of the foreground tab.
§ For all permissions, both "granted" and "denied" are reflected in the permission state (special case).
† For Microphone and Camera permissions, both "granted" and "denied" are reflected in the permission state. For Geolocation permission, only "denied" is reflected in the permission state. ‡ Notification permission will have "denied" reflected in the permission state.
W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Implications. As demonstrated, deleting web browser settings does not necessarily remove the permission state. This behavior may run opposite to user expectations and risks the unintended leakage of privacy data. This implementation in Firefox does not comply with the official documentation [40], [41]. As with private browsing modes, the permission state behavior after clearing web browser data greatly differs between browsers as well as OSs, implying that correctly understanding this behavior is extremely difficult for users.

4.6 How Is the Permission State Set When the Prompt Is Ignored? (T_5)

In test scenario T_5, we investigate how the permission state is set when the permission request prompt is ignored. Our preliminary experiments revealed that if none of the permission states are selected at the permission request prompt and the reload is repeated multiple times, the permission state is automatically set to "denied" [*2]. The analysis procedure is as follows. First, a client device browser accesses website A, which requests a permission; the browser accesses the site in either a background or foreground tab [*3]. The client device ignores the request and reloads the page multiple times. Finally, we analyze the state of the permissions.

Measurement Results. Table 7 summarizes the measurement results. In many web browsers, the permission was automatically set to "denied" by ignoring the prompt multiple times. The type of permission resource whose state is automatically denied differs depending on whether the reloading tab is in the foreground or background. In general, more cases in which the permission was automatically denied when the tab was in the foreground were observed. At least one permission was automatically denied in all browsers except Firefox.

Implications. Multiple reloads can cause the permission state to automatically be set to denied. This implies that attackers can control the permission state, which can be used to track users. Furthermore, such operations can be performed covertly in a background tab. We present a novel user tracking attack that uses this property in Section 6.1.

4.7 Does a Permission Request from a Tab Running in the Background Pop Up in Front? (T_6)

In test scenario T_6, we first investigate whether a permission prompt pops up when permission is requested on a tab running in the background. We call this analysis the base case. In the base case, all operations such as permission requests, page reloads, and returning to the previous page are performed in a background tab. In addition, as a special case, we investigate whether a permission prompt pops up for a permission request sent from a background tab running in a private browsing mode. We assume that the browser was restarted. We target browsers for iOS in which the private browsing mode tabs are retained even after the web browser is restarted. In browsers for iOS, tabs running in private browsing mode are invisible, although the tabs can render web content in the background.

Measurement Results. Table 8 presents the measurement results. In many browsers, permission prompts were not invoked from the background tabs. In the base case, permission prompts were invoked only in Firefox and Brave on iOS and Safari on macOS. In a web browser that displays a prompt, the prompt overlaid on the web browser screen where the active foreground page was displayed. In the special case, all iOS web browsers, except Chrome, did not display the prompt as an overlay. The behavior in Chrome for iOS was as follows. The permissions dialog requested by the tab placed in the front row of the private browsing mode was overlaid on the active tab working in the normal browsing mode.

Implications. We found a browser implementation that does not display a permission request prompt when a website loaded in a background tab requests permissions. This suggests that noticing that the tab is performing such behavior is difficult for a user.

[*2] The number of reloads for which the permission state is automatically set to "denied" depends on the web browser and OS. Table A·4 summarizes the results.

[*3] The reason for analyzing the case in which the browser accesses the website in a background tab is to evaluate the feasibility of the advanced attack described in Section 6.1.

The combination of these behaviors and the behavior shown in test scenario T_5, i.e., automatically setting the permission state to denied, allows the website to secretly manipulate the permission state of the web browser. In Section 6.1, we propose a user-tracking attack that takes advantage of these features.

Cases in which the prompts are overlaid also pose an inherent threat. As Section 7.2 demonstrates, distinguishing between a prompt prompted by a tab running in the background and a prompt prompted by a tab running in the foreground is often difficult for users. Users may misidentify the website (origin) that requests permission and make an erroneous decision to grant/deny permission. We call this attack "permission-based phishing attack" and provide details in Section 6.1. Furthermore, the overlaid display of prompts originating from the background private browsing mode tab (special case) risks being used for attacks that are more difficult for the target to detect.

In total, we found 191 implementation inconsistencies that could lead to user privacy risks. For reference, Table A·3 in Appendix A.2 presents the number of implementation inconsistencies for each browser. We note that one or more implementation inconsistencies are found in all 22 browsers analyzed in this paper.

5. Gap Between the User Perception and Browser Permission Implementations

We conduct a user study designed to understand users' expectations and perceptions of web browser permissions. To this end, we conducted an online survey that included 298 participants. After applying a consistency check, we identified a total of 232 valid responses. Through the online survey, we attempt to comprehend the intrinsic gap that exists between the web browser implementation inconsistencies identified in Section 4 and the users' perceptions. The user study consisted of six surveys, **U_1–U_6**, which correspond to the six test scenarios T_1–T_6 defined in Section 4. We conducted the online surveys using Lancers [42], hence our participants are Japanese. The demographics of the surveys are shown in **Table 9**. The questionnaire used for the online survey is available on the official website of this study [43].

Here is an example of a task that participants in the experiment engage in. In the survey **U_1**, we ask participants to assume the following scenario.

You are browsing a website using a web browser. The website requests camera permissions, and the permission request prompt shown in ***Fig. 2*** *appears on your web browser. Suppose that you have denied or granted the camera permission request.*

The participants answer the following questions about the permission granted to the website for each case: (1) whether a website can record camera footage and (2) whether a website can send the recorded camera footage to outside parties. The participants choose the answer from "Yes," "No," or "Don't know."

U_1: User Expectations on the Permission Mechanisms. Survey **U_1** aims to investigate users' expectation on the permissions they give to websites when they grant or deny these permissions. Specifically, we investigate users' expectations of whether a website can (1) record camera footage and (2) send the recorded camera footage to outside parties when users deny

Table 9 Demographic data from the online survey (N = 232).

Age	18–29	24
	30–39	59
	40–49	83
	50–59	41
	60 or over	19
	I don't want to answer.	6
Level of education	Graduate degree	11
	Bachelor's degree	132
	Assoc. degree/Tech. degree	16
	High school	65
	I don't want to answer.	6
	Other	2
Self-identified gender	Female	98
	Male	132
	Others/I don't want to answer.	2
Job status	unemployed	64
	self-employed	43
	employed	113
	I don't want to answer.	12
IT professionals	Yes	23
	No	203
	I don't want to answer.	6

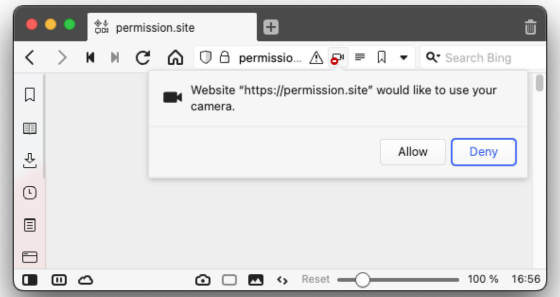

Website "https://permission.site" would like to use your camera.

Fig. 2 Screenshot of a web browser requesting camera permission (Japanese prompts were used in the survey. The prompts are shown translated into English).

or grant camera permissions. **Table 10** shows that the percentage of users who were able to correctly understand the relationship between camera permission status and whether or not the video can be recorded is 93% for denying permissions and 79% for granting permissions, respectively. The table also shows that the percentage of users who were able to correctly understand the relationship between camera permission status and the ability to send the recorded images to the outside parties was 88% for the denial of permissions and 60% for the grant of permissions. This means that 40% of the users did not have a correct understanding of the permission mechanisms.

U_2: User Expectations Regarding the Persistence of the Permission State. Survey **U_2** aims to investigate how users expect whether the permission state is persistent after setting the permission state to granted or denied. In our experiments, we asked users to assume a situation in which, after allowing or denying permissions on a website, they accessed the same website and received a second request for permissions.

Table 11 shows the results. Here, all participant groups were defined as Group A (N = 232), and Group B (N = 121) was defined as the group of participants who knew about and had used the private browsing mode. In the normal browsing mode, when the permission state is denied/granted, 65%/38% of users expect

Table 10 Results of **U1**: User expectations on the camera permission status.

User expectations (recording images)	Denied		Granted	
The website can record camera footage.	3	(1.3%)	183	(78.9%)
The website cannot record camera footage.	216	(93.1%)	10	(4.3%)
Other	13	(5.6%)	39	(16.8%)
User expectations (sending iamges)	**Denied**		**Granted**	
The website can send camera footage to an external source.	4	(1.7%)	140	(60.3%)
The website cannot send camera footage to an external source.	203	(87.5%)	19	(8.2%)
Other	25	(10.8%)	73	(31.5%)

Table 11 Results of **U2**: User expectations regarding persistence of permission status.

Browsing Modes	Normal				Private							
Group	A				A				B			
Permission status	Denied		Granted		Denied		Granted		Denied		Granted	
Persistent	49	(21.1%)	107	(46.1%)	44	(19.0%)	32	(13.8%)	18	(14.9%)	12	(9.9%)
Not persistent	150	(64.7%)	88	(37.9%)	146	(62.9%)	141	(60.8%)	92	(76.0%)	91	(75.2%)
I don't know	33	(14.2%)	35	(15.1%)	41	(17.7%)	7	(3.0%)	11	(9.1%)	14	(11.6%)
Other	0	(0%)	2	(0.9%)	1	(0.4%)	52	(22.4%)	0	(0%)	4	(3.3%)

Group A : All participants, Group B : Participants familiar with Private Browsing Modes

Table 12 Results of **U3**: User expectations regarding isolation of the permission status across browsing modes.

Browsing Modes	From normal to private								From private to normal							
Group	A				B				A				B			
Permission status	Denied		Granted		Denied		Granted		Denied		Granted		Denied		Granted	
Shared	48	(20.7%)	38	(16.4%)	21	(17.4%)	15	(12.4%)	34	(14.7%)	46	(19.8%)	15	(12.4%)	19	(15.7%)
Not shared	152	(65.5%)	147	(63.4%)	87	(71.9%)	88	(72.7%)	171	(73.7%)	145	(62.5%)	97	(80.2%)	85	(70.2%)
I don't know	32	(13.8%)	38	(16.4%)	13	(10.7%)	13	(10.7%)	27	(11.6%)	38	(16.4%)	9	(7.4%)	15	(12.4%)
Other	0	(0%)	9	(3.9%)	0	(0%)	5	(4.1%)	0	(0%)	3	(1.3%)	0	(0%)	2	(1.7%)

Group A : All participants, Group B : Participants familiar with private browsing modes

Table 13 Results of **U4**: User expectations regarding the browser data deletion mechanism.

Browsing Modes	Normal								Private							
Group	A				B				A				C			
Permission status	Denied		Granted		Denied		Granted		Denied		Granted		Denied		Granted	
Erased	174	(75.0%)	161	(69.4%)	143	(80.8%)	135	(76.3%)	159	(68.5%)	161	(69.4%)	88	(83.0%)	84	(79.2%)
Not erased	35	(15.1%)	45	(19.4%)	23	(13.0%)	28	(15.8%)	37	(15.9%)	29	(12.5%)	11	(10.4%)	10	(9.4%)
I don't know	22	(9.5%)	25	(10.8%)	11	(6.2%)	13	(7.3%)	35	(15.1%)	37	(15.9%)	7	(6.6%)	10	(9.4%)
Other	1	(0.4%)	1	(0.4%)	0	(0%)	1	(0.6%)	1	(0.4%)	5	(2.2%)	0	(0%)	2	(1.9%)

Group A : All participants, Group B : Participants familiar with private browsing modes

that the permission is not persistent, respectively. This result suggests that, in the normal browsing mode, users expect the permissions they grant to have persistence rather than the permissions they deny. In the private browsing mode, approximately 60% of users in Group A and 75% of users in Group B expect permissions to be non-persistent, regardless of the permission status.

The results shown in Section 4 revealed that, in the private browsing mode, all the browsers investigated persist in at least one or more permission states. These observations indicate that user expectations and the current web browser implementations differ.

U_3: User Expectations Regarding Isolation of the Permission State Across Browsing Modes. Survey **U_3** aims to identify users' expectations regarding the isolation of permission states across browsing modes. In our experiments, we asked users to assume a situation in which they grant or deny a camera permission request on one website and then access the same website in a different browsing mode and again receive a camera permission request.

Table 12 shows the results. As in the **U_2** experiment, we divided the users into two groups, A and B. Specifically, 63–73% of all users (Group A) expect that the permission state is not inherited across browsing modes. In Group B, where users are more familiar with the private browsing mode, the trend is more significant, with 72–80% of users expecting the permission state not to be shared across browsing modes, especially for denied permissions. The results presented in Section 4 revealed that, for all the web browsers investigated, the permission state is shared across browsing modes for one or more of the permission types. This result also suggests a gap between user expectations and browser implementation.

U_4: User Expectations for Browsing Data Deletion Mechanisms. Survey **U_4** aims to identify users' expectations of the browsing data deletion mechanism. In this experiment, we asked users to assume the following scenario. The user applies the data deletion feature provided by the browser after denying or granting a camera permission request for a certain website. The user then accesses the same website again, and camera permission is requested.

The results are shown in **Table 13**. Here, Group A consists of all participants ($N = 232$), Group B consists of participants who know and have used the data deletion mechanism ($N = 177$), and Group C consists of participants who know and have used both the private browsing mode and the data deletion mechanism ($N = 106$). Among all users (Group A), approximately 70% or more expect that the data deletion mechanism will erase the permission state, regardless of the browsing mode or permission state. Approximately, 80% of Group C users expect the data

Table 14 Results of **U5**: User expectations regarding browser behavior when permission requests are ignored.

Expected permission status	$m = 1$		$m = 2$		$m = 3$	
The website can use/get the camera footage	12	(5.2%)	9	(3.9%)	9	(3.9%)
The website cannot use/get the camera footage	177	(76.3%)	176	(75.9%)	180	(77.6%)
I don't know	43	(18.5%)	47	(20.3%)	43	(18.5%)

Table 15 Results of **U5**: User opinions of browser behavior when permission requests are ignored.

Question	Options	Count (%)	
As a user, what do you think about the repetitive display of permission request prompts?	Appropriate	61	(26.3%)
	Somewhat appropriate	98	(42.2%)
	Not really appropriate	39	(16.8%)
	Not appropriate	24	(10.3%)
	I don't know	10	(4.3%)
Do you think the mechanism † is implemented?	Yes	92	(39.7%)
	No	140	(60.3%)
The permission status should automatically be denied when permission is repeatedly requested.	Strongly agree.	54	(23.3%)
	Somewhat agree	82	(35.3%)
	Somewhat disagree	61	(26.3%)
	Strongly disagree.	28	(12.1%)
	I don't know	7	(3.0%)

† : Mechanism that prevents web browsers from repeatedly displaying permission request prompts.

Table 16 Results of **U6**: User expectations on the overlaid prompt display.

Permission status	Denied				Granted			
Website	A		B		A		B	
The website can use/get the camera footage.	10	(4.3%)	10	(4.3%)	53	(22.8%)	190	(81.9%)
The website cannot use/get the camera footage.	188	(81.0%)	200	(86.2%)	129	(55.6%)	20	(8.6%)
I don't know	34	(14.7%)	22	(9.5%)	50	(21.6%)	22	(9.5%)

† : The implementation is the display of permission request prompts on the different websites.

Table 17 Results of **U6**: User opinions on the implementation of the overlaid dialog display.

Question	Options	Count (%)	
As a user, what do you think about the implementation? †	Appropriate	28	(12.1%)
	Somewhat appropriate	27	(11.6%)
	Not really appropriate	43	(18.5%)
	Not appropriate	126	(54.3%)
	I don't know	8	(3.4%)

† : The implementation is the display of permission request prompts on the different websites.

deletion mechanism to erase their permission state in the private browsing mode.

As shown in Section 4, all web browsers, except for Safari, retain their permission state even after the data deletion mechanism is applied. Thus, a gap between user expectation and implementation with respect to the data deletion mechanism of web browsers for permission states is noticeable.

U_5: User Expectations of Browser Behavior when Permission Requests Are Ignored. The objective of survey **U_5** is to understand the user's expectations of the permission state when the user ignores permission request prompts multiple times. In this experiment, we asked participants to assume the following scenario. A user accesses a website that requires camera permission. The user ignores the permission request prompt several times.

As shown in Section 4, all browsers, except for Firefox, have introduced a mechanism to automatically deny permission and not display the permission request prompt if the permission request is ignored several times. **Table 14** shows the results of examining the relationship between the number of times a permission request is ignored m and the browser's behavior regarding the permission status expected by the user. Regardless of the number of times a request was ignored, about 25% of users gave answers that were either different from the actual browser behavior or indicated they were unsure. **Table 15** examines users' expectations of browser behavior when permissions are repeatedly requested. Approximately, 68% of users believe that an implementation that displays a permission request prompt each time is appropriate, even if the request is ignored multiple times. Another 38% of users believe that permissions should not be denied automatically. These results clearly indicate a gap between user expectations and current browser implementations.

U_6: User Expectations for the Overlaid Prompt Display. The purpose of survey **U_6** is to determine users' expectations of browser behavior when a permission request prompt pops up from a different website than the one displayed on the screen. In this study, we asked participants to assume the following scenario. While the user is browsing website A displayed on the screen, website B, which is open in a background tab, displays a camera permission request prompt overlaid on top of website A. The user selects to deny or grant the displayed permission request. In the above scenario, we surveyed what permission status the user expects to be given to each website.

Table 16 shows the results from asking the user what permissions were granted to websites A and B when the user chose to deny or grant the camera permission request sent by website B, which was hidden from the screen. If the user grants the permission request sent by website B, then the correct specification is that no permissions are granted to website A. If the user grants the permission request sent by website B, then the correct specification is that permissions are granted to website B. However, 44% of the users either did not know or said that website A could use the camera. This result suggests that it is not easy for users to make the right decision when the permission request prompt is overlaid on the screen.

Table 17 asks users' opinions about the implementation of the overlay of the permission request prompt. Here, 19% of the users answered that it was rather inappropriate, and 54% answered that it was not appropriate. As indicated in Section 6.2, implementations in which permission request prompts are overlaid can be exploited by attackers.

6. Attack Concept

This section describes the concepts of two attacks based on inconsistencies in permission implementations of the web browsers we found in the measurement study: *permission-based user tracking attack* and *permission-based phishing attack*. In a permission-based user tracking attack, the attacker uses the permission state maintained in the web browser to track the user. In a permission-based phishing attack, the attacker induces an erroneous decision to grant/deny permission by displaying a fraudulent overlay of permission prompts.

6.1 Permission-based User Tracking Attack

6.1.1 Threat Model

In a permission-based user-tracking attack, an attacker tracks users who visit a landing website by checking the permission state set for the tracking websites and stored in the target's browser. This attack uses implementation inconsistencies T_3 and T_5 found in Section 4. To collect/check the permission state, the attacker installs malicious code on the landing website or installs the code into a web advertisement. The attacker sets up their tracking websites and lets the target's browser set the permission states for each tracking website, following a predetermined pattern specific to each user, the user ID (UID). By checking the permission states for the tracking websites when the user revisits the landing website, an attacker can reconstruct the UID.

This attack has advantages over tracking methods that simply encode information in URLs, such as the persistence and the ability to track users across the different browsing modes. The advantages of this attack are as follows: First, even in situations where third-party cookies are deprecated [44], [45], this attack allows an attacker to track users from a cross-origin. Second, this attack does not require user consent because the permission state is automatically set as denied by ignoring the permission prompts, as shown in Section 4.6. The attack is stealth because the automatic permission state can be executed in a background tab. Third, after the target leaves the website, this attack can still track the target as long as the permission state is maintained. Finally, the permission state is shared among the browsing modes as shown in Section 4.4, which implies that an attacker can track a target across normal and private browsing modes.

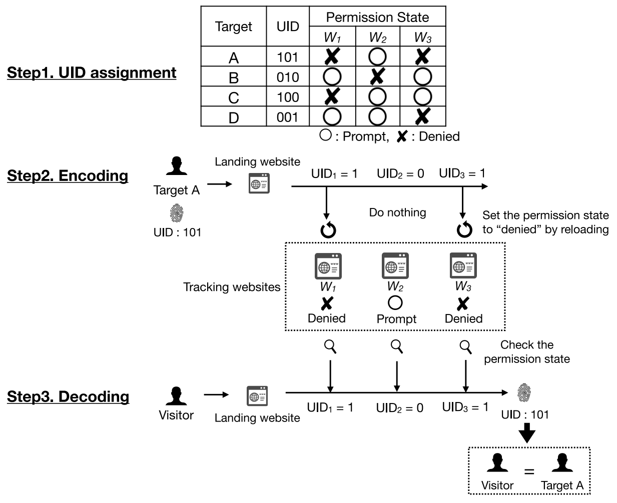


Fig. 3 Overview of permission-based user tracking attack.

6.1.2 Attack Procedure

The procedure for this attack comprises the following three steps: **Step 1**: UID assignment, **Step 2**: encoding, and **Step 3**: decoding. **Figure 3** presents an overview of the three steps.

Step 1: UID Assignment. First, an attacker assigns a unique UID to each target. The length of the UID is l bit, the maximum number of people to be tracked, U is $U = 2^l$. When $l = 32$, approximately 4.3 billion users can be uniquely identified in theory. Second, the attacker prepares l-tracking websites with different origins. Here, the n-th tracking website W_n $(1 \leq n \leq l)$ is mapped to the n-th bit of the ID. Finally, a JavaScript code is added to each tracking website to automatically set the denied permission state (**Step 2**) or obtain the permission state (**Step 3**).

Step 2: Encoding. Using malicious JavaScript code loaded by the target's browser, an attacker manipulates the permission states of the tracking websites, following the ID generated in **Step 1**. Any type of permission can be used. Regardless of the browser, the resources used universally are cameras or microphones; hence, the attacker is likely to need these permissions. When the n-th digit of the UID is 0, the attacker does nothing, leaving the permission state for W_n to be "prompt." When the n-th digit of the UID is 1, the attacker causes the permission state for W_n to be "denied" by repeatedly reloading the tracking website. Malicious code causes the target web browser to repeat the aforementioned process for all W_n $(1 \leq n \leq l)$.

The target browser must wait to render permission prompts in the foreground or background at each tracking website. Otherwise, the number of times that the prompt is ignored will not be incremented, and the permission state for the tracking website W_n cannot be denied. We experimentally derived the waiting time as 2 ms to 160 ms required for a successful attack, detailed in Appendix A.4.

Step 3: Decoding. When a user revisits the landing website, their browser loads the malicious code causing the browser to access the tracking websites W_n $(1 \leq n \leq l)$ and checks the permission state on each tracking website [*4]. Following the data created in

[*4] An attacker can prepare a special bit to determine whether it is a first visit or a return visit. Malicious code running on a landing website can check that special bit to decide whether or not it should perform the encoding or decoding operation next. For brevity, we omit a detailed description.

Step 2, the attacker can decode the binary sequence corresponding to the permission states and obtain the ID of the user; hence, tracking the user is completed.

6.1.3 Tweaks

In the following section, we present tweaks to increase the feasibility of the attack.

Packing Multiple Permissions. By leveraging the fact that the landing website can simultaneously request or obtain multiple permission states, the attack can occur sooner than expected. Web browsers for desktop OS display only one permission prompt when a tracking website requires Microphone and Camera permissions simultaneously. The combinations of the two permission states are represented as a 2-bit sequence, `00, 01, 10, 11`, where 0/1 represents permission granted/denied. This approach enables an attacker to encode/decode 2 bits of information per request. This method can reduce the required time to complete the encoding/decoding by about half.

Background Attack. An attacker can improve the secrecy of the attack by executing this attack in a background tab. Many web browsers provide a feature of opening different pages simultaneously in multiple tabs and this feature is widely used. When multiple tabs exist, the user can only see the tab in the foreground and cannot know the behavior of the tabs in the background. By taking advantage of this characteristic, an attacker can perform the encoding/decoding process unbeknownst to the user.

Iframe Attack. In Safari on macOS, the decoding process can be efficiently performed using an iframe element. Most browsers implement "Permission Delegation" to manage permissions in iframes. Permission delegation is a mechanism that allows a child page embedded in an iframe to request permission from the domain of the parent page when it attempts to request permission and then delegates the results to the child page [46]. Since Safari on macOS has not implemented permission delegation, it can request/obtain permission in multiple domains depending on the domains of the child pages by transitioning only the child pages embedded in the iframe window. With Safari on macOS, the decoding process can be performed only with page transitions of the child pages in the iframe window without causing the transition of the parent page.

In summary, permission-based user tracking attack aims to track users by encoding and decoding UIDs into permission state. In addition to ① normal attack, attackers can improve the efficiency and secrecy of this attack by: ② packing multiple permissions, ③ background attack, and ④ iframe attack.

6.2 Permission-based Phishing Attack

6.2.1 Threat Model

An attacker compels the target to mistakenly grant access to a resource by presenting a fake permission request. This attack uses implementation inconsistencies T_6 found in Section 4. First, the attacker prepares a website that requests permission. This site is hereafter called the "attack site." When users access the attack site, they are prompted to click a link that opens Website A in a new tab. When the user clicks the link, a tab displaying Website A is opened in the foreground, whereas the tab on the attack site runs in the background. The script runs on the attack site in the background tab and requests permission. Consequently, a dialog is overlaid on the screen of Website A, displayed in the foreground. Once the user is tricked into granting permission to the attack site, the attacker accesses the privacy-sensitive resources of the target such as camera footage, microphone audio, and location information.

6.2.2 Attack Procedure

The procedure of this attack has the following steps: **Step 1**: Preparation of the attack site, **Step 2**: Displaying prompt, and **Step 3**: Abuse of granted permission.

Step 1: Preparation of the Attack Site. The attacker prepares an attack site that the target accesses. The domain name of the attack site is displayed in the permission prompt as the source domain for the permission request. The attacker adopts a domain name that looks authentic; thus, the target users may believe that a permission request is sent from a trustworthy source such as a browser or a trusted site. The attack site uses HTML links and JavaScript to allow the target to open a widely trusted popular website in a new tab. Such trusted websites include search engines, map services, and online conferencing services, which many users use daily and for which granting permission requests is natural.

Step 2: Displaying Prompt. The attack site displays a permission prompt as an overlay on the foreground tab, displaying the trusted website. The tab of the attack site is moved to the background, and the attack script can detect this change by monitoring the on click event of the link or the visibilitychange event of DOM [47], [48].

Step 3: Abuse of Granted Permission. After the user is tricked and grants permission to the attack site, the attacker can abuse the privacy-sensitive resources of the target such as camera footage, microphone audio, and location information. The attacker can use WebSocket and WebRTC [49], [50], which allow them to monitor and collect information such as video, audio, and location of the target in real time.

7. Feasibility of the Attack

In this section, we evaluate the feasibility of the two attacks described in Section 6.

7.1 Permission-based User Tracking Attack

7.1.1 Attack Targets

Attackable browsers vary depending on the attack conditions. **Table 18** summarizes the attackable targets for each attack condition. First, it is clear that all browsers except iOS browsers and Firefox, can be targeted by ① normal attacks. Second, the results show that ② packing multiple permissions and ③ background attack are also feasible for a wide range of browsers. ④ iframe attack is feasible in Safari on macOS. **Table 19** summarized the effectiveness of tracking across browsing modes. Tracking users from normal browsing modes to private browsing modes is feasible in browsers similar to normal attack. Tracking users from private browsing modes to normal browsing modes is only possible in Safari on macOS.

To succeed in this attack, the target browser must support the Permissions API or Notifications API to enable the attacker to

Table 18 The target of the permission-based user tracking attack.

	Chrome					Firefox					Edge					Brave					Safari	
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
① Normal attack	●	●	●	●	○	○	○	○	○	○	●	●	●	●	○	●	●	●	●	○	●	○
② Packing multiple permissions	●	●	●	○	○	○	○	○	○	○	●	●	●	○	○	●	●	●	○	○	○	○
③ Background attack	●	●	●	●	○	○	○	○	○	○	●	●	●	●	○	●	●	●	●	○	◐	○
④ Iframe attack	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○

● : Attackable, ○ : Not attackable, ◐ : Attackable for Encoding, W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table 19 The target of the permission-based user tracking attack across browsing modes.

Order	Chrome					Firefox					Edge					Brave					Safari	
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i
Normal → Private	●	●	●	●	○	○	○	○	○	○	●	●	●	●	○	●	●	●	●	○	●	○
Private → Normal	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○

● : Attackable, ○ : Not attackable, W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table 20 Support for the packing multiple permissions (Microphone and Camera)*.
○ : Supported ● : Not supported

	chrome	firefox	edge	brave	safari
Windows	○	○	○	○	n/a
Linux	○	○	○	○	n/a
macOS	○	○	○	○	○
Android	●	●	●	●	n/a
iOS	●	●	●	●	●

secretly obtain the permission state of the browser without displaying a permission prompt. These APIs are available in all browsers except iOS. Section 2.1 presents the details of our survey.

We investigated browsers that support packing multiple permissions for Microphone and Camera permissions. **Table 20** shows the results. All browsers capable of attacking permission-based tracking attacks, with the exception of the Android browsers, support packing Microphone and Camera permissions.

7.1.2 Evaluation of the required time

The time required for the attack was evaluated. We measured the time T_{enc} and T_{dec}, being the time required to complete the encoding and decoding processes, respectively.

Measurement Setup. The following section describes the measurement setup. The server is a virtual environment with Ubuntu 20.04 LTS installed with 2 GB memory and 2 vCPUs. The web interface was implemented in Python and the Flask framework. The three client devices, Device A, Device B, and Device C, are a Windows PC (desktop), macOS (laptop), and Android smartphone. In Device A, the microphone and camera were connected via USB. In Device-B and Device-C, the microphone and camera were built into the device.

Measurement. Figure 4 presents the relationship between U and T_{enc} and T_{dec} when a normal attack is performed on devices A, B, and C and when the packing multiple permissions technique is performed on device B. We performed the experiments three times for each combination of parameters and calculated the average value of the measured time. We used Microphone permission during a normal attack.

When the number of target users, U, was approximately 4.3 billion and when the ID length assigned to each user l was 32 bits, the T_{enc} of devices A, B, and C were 17.3 s, 7.0 s, and 9.4 s, respectively. The T_{dec} were 3.4 s, 2.6 s, and 6.0 s, respectively. We deem that Device-A took a long time encoding because the

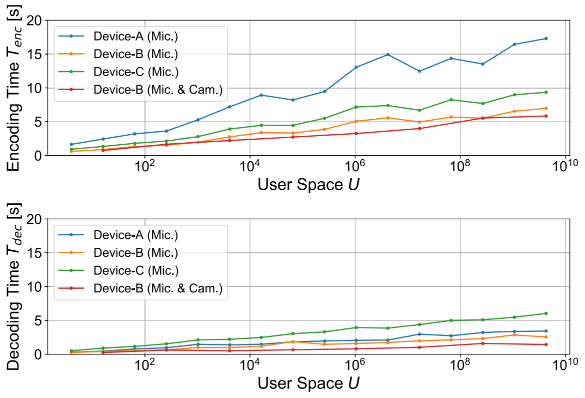


Fig. 4 U vs. required time. Top: encoding and Bottom: decoding

microphone was connected as a USB external device, which required more time for the browser to iterate through the permission request process. Furthermore, the result implies that packing multiple permissions (Microphone and Camera) increases the attack efficiency.

7.1.3 Further Evaluation of the Permission-based User Tracking Attack

Optimizing the Time for Attacks. We propose a method to dynamically change the length of UIDs and evaluate the time required for the attack. The proposed method generates and assigns UIDs of dynamic length, sorted according to the user's access order. More specifically, the N-th person is assigned a UID of N, represented as a binary number; i.e., $N = 1, 01, 10, 11, 001, , \ldots$. We introduce a flag that represents the end of the identifier; i.e., when the state of a particular permission is denied, we use that information to represent the end of the identifier. This approach named "sorted index" allows us to optimize the time required for the attack.

First, we surveyed the operating systems and browsers that support the sorted index. As a result, we found that Chrome, Edge, and Brave, excluding the iOS version, support the sorted index. Next, we measured the time T_{enc} and T_{dec} for the case of random ID assignment and the case of ID assignment with sorted index approach, using the device B. Here, we fixed the user space $U = 2^{32}$ and measured the time taken for the attack when three random users visited from the first to the Nth user. **Figure 5** present the results. We can see that the use of sorted index optimizes the time required for the attack regardless of the

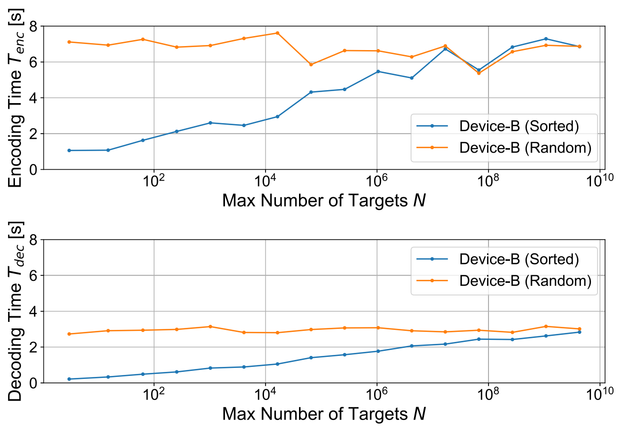


Fig. 5 N vs. required time. Top: encoding and Bottom: decoding

Table 21 Impact of browser restarts on permission state persistence.

	Chrome	Firefox	Edge	Brave	Safari
Windows	●	–	●	●	n/a
Linux	●	–	●	●	n/a
macOS	●	–	●	●	N
Android	●	–	●	●	n/a
iOS	–	–	–	–	–

● : All permission states persist after the browser restarts.
N : Notification permission state persists after the browser restarts.
– : Permission state will not be automatically set to denied by ignoring the prompt.

user space to be attacked.

Evaluation of User Perception. We evaluated user perceptions in the permission-based user tracking attack. Twenty participants were recruited to conduct the study. All the participants were university students majoring in science and engineering. The participants were divided into two groups, A and B, each consisting of 10 participants. Participants in Groups A and B browsed the attack website, where the user tracking attack ran in the foreground/background, respectively, using a desktop (macOS) Chrome browser that we provided. We asked all participants if they had noticed anything suspicious after browsing. If they reported seeing anything suspicious, we asked them why and what they thought should be done about it.

The results showed that 80% of the participants in Group A noticed suspicious behavior related to this attack. Moreover, 40% of the participants in Group A were able to correctly indicate how to stop the attack while it was happening. None of the participants in Group A were able to demonstrate how to eliminate the threat of this attack such as erasing the permission state stored in the browser or deleting the browser data. None of the participants in Group B noticed any suspicious behavior related to this attack. These results suggest that users are unable to detect user tracking attacks using background tabs.

Note that, according to the results of the user study presented in Section 5, 60% of users do not know the feature that lets web browsers automatically set the permission status to denied when a permission is requested multiple times by the same website.

Evaluating the Persistence of the Attack. In the following, we evaluate the persistence of the attack. First, we investigated whether a permission state set to denied by a user tracking attack persisted when the browser was restarted. **Table 21** shows the results. In Chrome, Edge, and Brave, the permission state persisted even after a browser restart. Next, we examined the permission state more than 7 days after the attack. For Chromium-based browsers — Chrome, Edge, and Brave (except for the iOS version, which is WebKit), the permission state set to "denied" by reloading returned to the state "prompt" after 7 days. Then, when the prompt was reloaded once again, the permission state was set to "denied." This behavior is consistent with the description in the literature [51]. Thus, 7 days after the attack the encoded permission state can be restored by making a single permission request to all tracking websites. By performing these additional operations, user tracking can be achieved for longer than 7 days. Safari maintained the permission state even after 7 days had elapsed.

Table 22 Attackability of the browsers with the permission-based phishing attack.

	Chrome	Firefox	Edge	Brave	Safari
macOS	○	○	○	○	○
iOS	○	MC	○	MC	○

MC : Microphone+Camera permissions are vulnerable.
○ : All permissions are not vulnerable.

7.2 Permission-based Phishing Attack

The following section evaluates the feasibility of the permission-based phishing attack where an attacker aims to mislead the user's judgment.

7.2.1 Attack Targets

Table 22 summarizes the attackable targets for each attack condition. The attack applies to iOS browsers such as Firefox and Brave.

7.2.2 User Study

We conducted a user study to assess the threat of a permission-based phishing attack. We aimed to investigate how users understand permission mechanisms and how they interact with the displayed permission prompts. Therefore, we adopted an online survey approach. We recruited participants for our survey using Lancers [42], a well-known crowdsourcing platform in Japan, hence our participants are Japanese. Our participants spanned a wide range of ages (18 years and older) and had a variety of educational backgrounds. Many participants were in their 30s or 40s. The details of the demographics are shown in **Table 23**. The questionnaire used for the online survey is available on our website [43].

Our user study consisted of the following three experiments: **E$_1$**, **E$_2$**, and **E$_3$**.

E$_1$: Understanding the Source of the Permission Request. This experiment aims to answer the following question: *When a user sees an overlaid permission request prompt, what does the user think the source of the permission request is?* To answer this question, we set two scenarios "Normal" and "Fake." Participants saw the smartphone browser screens for the two scenarios and were asked to determine which sites requested permission for both scenarios. In the "Normal" scenario, the foreground tab of the browser shows the restaurant's website, with an overlay of a prompt permission dialog requested by that website (**Fig. 6**). In the "Fake" scenario, the foreground tab of the browser shows the results of a search for Italian restaurants using Google. There is

Table 23 Demographic data from the online user study (N = 99).

Age	18–29	17
	30–39	38
	40–49	31
	50–59	11
	60 or over	1
	I don't want to answer.	1
Level of education	Graduate degree	2
	Bachelor's degree	62
	Assoc. degree/Tech. degree	13
	High school	21
	I don't want to answer.	1
Self-identified gender	Female	50
	Male	48
	Others/I don't want to answer.	1
Job status	unemployed	24
	self-employed	6
	employed	64
	I don't want to answer.	5
IT professionals	Yes	15
	No	84

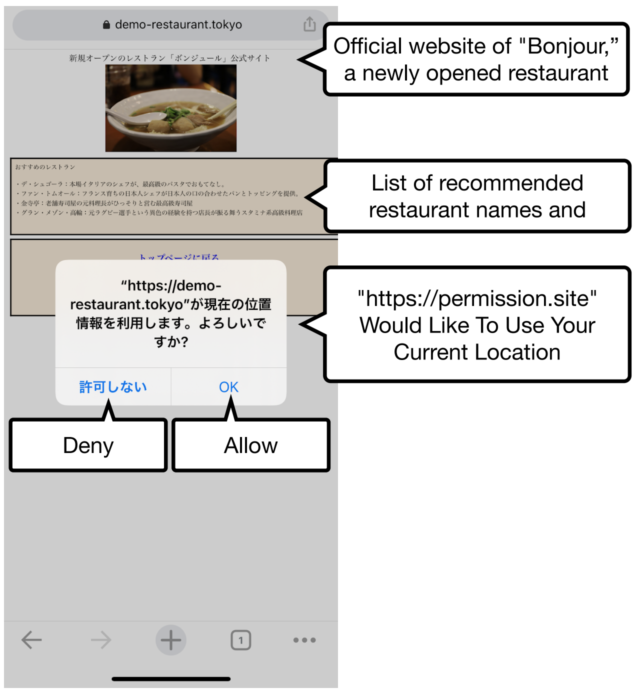

demo-restaurant.tokyo
Official website of "Bonjour," a newly opened restaurant
List of recommended restaurant names and
"https://demo-restaurant.tokyo"が現在の位置情報を利用します。よろしいですか?
許可しない
OK
"https://permission.site" Would Like To Use Your Current Location
Deny
Allow

Fig. 6 A screenshot of a "Normal" situation (The English translations are shown in speech bubbles).

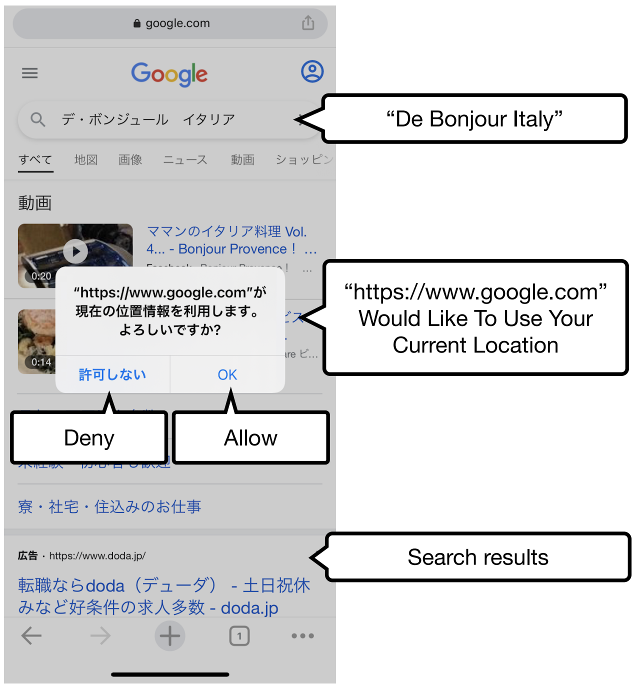

google.com
Google
デ・ボンジュール イタリア
"De Bonjour Italy"
すべて 地図 画像 ニュース 動画 ショッピン
動画
ママンのイタリア料理 Vol. 4... - Bonjour Provence ! ...
"https://www.google.com"が現在の位置情報を利用します。よろしいですか?
許可しない
OK
"https://www.google.com" Would Like To Use Your Current Location
Deny
Allow
寮・社宅・住込みのお仕事
広告・https://www.doda.jp/
転職ならdoda（デューダ） - 土日祝休みなど好条件の求人多数 - doda.jp
Search results

Fig. 7 A screenshot of a "Fake" situation (The English translations are shown in speech bubbles).

Table 24 Results of the Experiment (E_1, E_2: $N = 99$, Experiment E_3: $N = 60$).

Experiment	Participants' answers	Counts
E_1 (Normal)	Restaurant website (**correct answer**)	67
	Google website	31
	Other	1
E_1 (Fake)	Restaurant website	10
	Google website	89
	Other (**correct answer**)	0
E_2	Website from which permission is requested	60
	Necessity of permission	61
	Permission type	40
	Timing of permission prompt	11
	No basis for judgment as I always grant it.	4
	No basis for judgment as I always deny it.	19
E_3	Features provided by the website	37
	Daily use or not	39
	Existence of a description of the purpose	22
	Appearance of the website	3
	Expertise of the website	9
	Authoritativeness of the website	22
	Trustworthiness of the website	40

an overlay of a permission request dialog invoked by the attack site running in a background tab (**Fig. 7**).

Table 24 (top) shows the results. Surprisingly, even for the "Normal" scenario, 32% of the participants could not correctly identify the actual source of the permission request. Furthermore, for the "Fake" scenario, none of the participants could identify the correct source. These results suggest that it is not easy for users to understand where the permission request dialog originates correctly and that our proposed attack makes understanding even more difficult.

E_2: Reason to Grant or Deny a Permission Request. This experiment aims to answer the following question: *On what basis does the user choose to grant or deny a permission request?* We presented the participant with options that provided the basis for granting or denying permission. The participants selected one or more of these options as their basis. We also provided participants with two options: always grants or always denies. Table 24 (middle) shows the results. Most participants answered that they relied on information about the website or the context (necessity of permission) to decide whether to grant permission. We performed experiment $\mathbf{E_3}$ for those who responded that they relied on the features of the website. Approximately 20% of users reject permission requests regardless of the content or context of the website, whereas 4% of users always accept permissions.

E_3: Information about the Website to Decide Whether to Grant or Deny a Permission Request. This experiment aims to answer the following question: *When a user receives a permission request, what information about the website does the user use in deciding whether to grant or deny the request?* Participants in experiment (E_3) were limited to those who responded "Website from which permission is requested" in experiment (E_2). We presented participants with information options that can provide a basis for decisions related to the website. Participants selected one or more of the options presented. Table 24 (bottom) shows the results. The following were reported by many participants as the basis for their judgments about websites that requested permissions: "trustworthiness of the website" (67%), "daily use or not" (65%), and "features provided by the website" (62%).

Implications of the User Study to the Attack Success.

In a permission-based phishing attack, potential attack targets should first meet the following condition:

C_0: *The user makes permission decisions without always granting or denying permission.*

The experiment $\mathbf{E_2}$ revealed that the probability that a user meets the condition is $P(C_0) = 76/99 = 0.77$. Subsequently, under condition C_0, the potential attack targets should meet the following two conditions:

C_1: *The user misidentifies the source website of the permission request owing to an overlay of permission prompts.*

C_2: *The user uses the basis of the website features when making permission decisions.*

Experiments $\mathbf{E_1}$ and $\mathbf{E_2}$ revealed that the conditional joint probability is $P(C_1, C_2|C_0) = 54/76 = 0.71$. In summary, 77% users meet condition C_0 and of these, 71% of users meet the two conditions, implying that 55% are potential attack targets. Note, we can directly compute the conditional joint probability since the same group of users participated in the three experiments. Under condition C_0, the two conditions C_1 and C_2 were not correlated; i.e.,

$$P(C_1|C_0)P(C_2|C_0) = 68/76 \times 60/76 \approx P(C_1, C_2|C_0).$$

8. Discussion

8.1 Toward the Fundamental Solutions

Sections 4, 6, and 7 indicate the specifications and implementations of permission mechanisms are fragmented, and it is difficult for users to correctly understand their behavior. We showed several implementation inconsistencies in web browsers and that attacks that exploit these inconsistencies are a real threat. The root cause of these inconsistencies in the permission mechanism is the lack of standardization and sharing of best practices. Currently, only a working draft [25] exists to standardize the permission mechanism. Although the draft has a substantial discussion for the Permissions API, there is a lack of discussion about the browser's handling of permissions, e.g, persistence and automatic setting of "denied" state.

Specifications and implementations are not consistent across browsers because they are determined independently by each browser vendor. We suggest that standardization at W3C and WHATWG [52], [53] and sharing of best practices among browser vendors are needed to solve this problem. In fact, our user study, presented in Section 5, revealed the mismatch between the user expectation and browser implementations. The standardization process will be a promising first step toward achieving consistent and user-friendly browser behavior implementation. We present several generic technical approaches to address the implementation inconsistencies in this study and mitigate threats. Comments on issues specific to each browser were also presented.

Desirable Features and Settings to Be Introduced in the Permission Mechanism. As shown in Sections 6 and 7, there is a risk that privacy information, such as browsing history, can be inferred from the permission state that could be leaked from the web permission mechanism. An attacker can covertly track users by browsing footprints on multiple websites. A primary factor that makes these attacks a practical and serious threat is that some implementations share the permission state between normal and private browsing modes.

Based on the above observations, we summarize the requirements that implementations of permission mechanisms must meet to protect users' privacy as follows:

- R_1: Do not make the permission state permanent; create an option to clear it periodically.
- R_2: Do not automatically set the permission state denied when the prompt state is reloaded multiple times.
- R_3: Restrict permission requests sent from pages in the iframe.
- R_4: Make permission state visible and configurable by users.

R_1 expires permission state after a set period and allows the user to reset it when the site is visited again. Users can check for incorrectly set permission states by providing this option. Firefox, Brave, and Safari implemented this approach [54]. However, they are incomplete since some permissions do not support the ability to set an expiration date and the expiration options provided are limited.

R_2 is a proposal to avoid an attacker remotely forcing the user's permission state to deny. Currently, Firefox and iOS browsers (WebKit) have adopted this approach. We believe that other browsers need to support this as well.

R_3 is a feature provided as a Permission Delegation [46] in the working draft of the Permissions Policy [55]. While the Permission Delegation is implemented only in Chromium and Firefox, it should be standardized and spread to all browsers.

R_4 provides a list of permissions that the users set for each site. Visualization of the permission state improves the transparency of permission mechanisms for users. R_4 has been implemented in many browsers [54], [56], [57], [58]. However, because these functions are in the deep hierarchy of the settings screen, it is difficult for users to recognize their existence unless they understand web permissions and operate their browsers to check or change their settings. The standardization of functions that provide permission settings has not progressed. Alternatively, mobile operating systems already provide a usable interface for controlling permission settings for each app, allowing users to review app permission settings regularly. A similar user interface should be provided for web permission.

We expect these requirements to be shared as best practices by browser vendors and established as technical standards.

Fixes for Implementation Inconsistencies We present the proposed solutions for implementation inconsistencies in the permission mechanism.

- F_1: Do not share the permission state between the normal and private browsing modes.
- F_2: Explicitly clear the stored permission state when exiting private browsing mode.
- F_3: Do not overlay a permission prompt screen sent from a

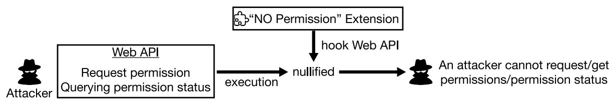

"NO Permission" Extension
hook Web API
Web API
Request permission
Querying permission status
Attacker
execution
nullified
An attacker cannot request/get permissions/permission status

Fig. 8 Mechanism of the "NO Permission" Extension.

background tab. When a background tab requests permission, a pop-up is held until the requesting tab becomes visible.

Note that each browser vendor can fix all the aforementioned inconsistencies.

Finally, we comment on Apple WebKit. WebKit is a browser engine developed by Apple [59]. Apple's policy is that all iOS browsers must be developed using WebKit [60]. This policy has the advantage that vulnerabilities caused by the browser engine can be fixed early, without waiting for individual app updates, because WebKit is set in conjunction with the OS updates. However, as shown in Section 4, there are several cases in which all iOS browsers have common inconsistencies in the permission mechanism. For example, even with Brave, a browser designed with privacy in mind, the only iOS version implemented inconsistencies such as persistent permissions and sharing permission states between normal browsing mode and private browsing mode. This inconsistency defeats the expectations of Brave users, who value privacy. Because WebKit's impact is significant, we hope that countermeasures would be introduced, such as the aforementioned proposed modifications, being applied or aligned with the policy that reaches a consensus in the browser vendor community.

8.2 Browser Extensions to Protect Users from Abuse of Permissions

In this section, we describe the browser extension "NO Permission," an extension that protects the user from inappropriate permission implementations.

Purpose of the Extension. "NO Permission" is designed to protect users from abuse of features that require permission and from user tracking based on the permission states described in Section 6.1. The results in Section 4.5 show that sometimes the function to clear data stored in the browser does not properly clear the permission states. Furthermore, it is not sufficient to simply deny permission requests from websites in order to protect users from the Permission-based user tracking described in Section 6.1. Thus, "NO Permission" protects user privacy by intercepting permission requests and permission status inquiries by websites without relying on the data clearing feature of the browser.

Mechanism of the Extension. "NO Permission" disables permission requests and the obtaining of permission status by a website. **Figure 8** shows the mechanism of "NO Permission". "NO Permission" is provided as a web browser extension and injects JavaScript codes into websites that users visit. JavaScript Hooker [61] hooks the Web APIs related to the permissions shown in **Table 25** at runtime, and prevents the execution of the permission request and obtaining permission status functions. A user can disable extensions in order to utilize websites with features that require permissions.

Evaluation of the Extension. We evaluated the effectiveness of the extension as a countermeasure against the permission-based user tracking attack. We found that "NO Permission" is able to protect users from permission-based user tracking attacks. We measured the feasibility of a permission-based user tracking attack, ① normal attack, in both normal Chrome and "NO Permission" installed and enabled Chrome. **Table 26** shows the measurement results. This result shows that in Chrome with "NO Permission" installed and enabled, tracking is not achieved when an attacker uses any of the following permission types: Microphone, Camera, Geolocation, and Notification. Note that we used Chrome installed on Mac for our measurements.

Discussion of trade-offs. There are two trade-offs in "NO Permission": usability and fingerprinting. Users need to manually disable extensions when they want to use browser functions that require permissions. This may reduce usability.

It is pointed out that the installation status of browser extensions contributes to browser fingerprinting [63]. "NO Permission" blocks the actions of Web APIs related to permissions, making it easy for an attacker operating sites to detect the use of the extension. Therefore, an attacker can potentially leverage the installation status of the "NO Permission" as one element of browser fingerprinting.

Note that "NO Permission" is open source and available to everyone [43].

8.3 Ethical Considerations

User Study. We followed the policy set forth by our organization's IRB and confirmed that our user study is in compliance with IRB review. Our user study did not collect sensitive information about participants. In the experiment shown in Section 7.1.3, the participants used the devices we provided, and the data obtained in the experiment were anonymized and statistically processed so that individuals could not be identified. Furthermore, fake domain names used in the experiment were not publicly registered. This domain name is accessible only in our experimental environment and has no negative impact on third parties. We paid participants an amount above the minimum wage in the region in which the experiments were conducted.

Responsible Disclosure. We contacted and disclosed information to the five browser vendors, Google, Mozilla, Microsoft, Brave Software, and Apple on August 14, 2022. We have asked each browser vendor to let us know their course of action within two weeks of receiving our report. Furthermore, we had held extensive discussion with JPCERT/CC [64], which is a vulnerability coordinator in Japan, prior to the disclosure.

The five browser vendors are currently reviewing the issue and implementing fixes based on our report. Brave has fixed the implementation where the permission state is inherited from normal browsing mode to private browsing mode (Section 4.4), and on Android, they have fixed the implementation where the Notification permission is automatically set to "denied" when it is requested multiple times from the background tab (Section 4.6). The Brave team has already merged the revised code. A fixed version of Brave will be published [65], [66]. They are also considering a UX/UI update regarding the implementation where the private browsing mode permission state is not cleared when

Table 25 List of Web APIs supported by "NO Permission" (We selected the target Web APIs based on the permission.site project [62]).

Permission Type	Web API
Notification	Notification.requestPermission
Geolocation	navigator.geolocation.getCurrentPosition navigator.geolocation.watchPosition
Microphone, Camera, Screenshare	navigator.mediaDevices.getUserMedia navigator.mediaDevices.getDisplayMedia
MIDI	navigator.requestMIDIAccess
Bluetooth	navigator.bluetooth.requestDevice
USB	navigator.usb.requestDevice
Serial	navigator.serial.requestPort
Human Interface Device (HID)	navigator.hid.requestDevice
Encrypted Media Extensions (EME)	navigator.requestMediaKeySystemAccess
IdleDetector	IdleDetector.requestPermission
Storage	navigator.storage.persist
Quota Management	navigator.webkitPersistentStorage.requestQuota
Protocol Handler	navigator.registerProtocolHandler
WebAuthn	navigator.credentials.create
Near Field Communication (NFC)	reader.scan
Extended reality	navigator.xr.requestSession
Clipboard	navigator.clipboard.readText

Table 26 Attack feasibility targeting normal Chrome and Chrome with "NO Permission" enabled.

	Permission type			
	Microphone	Camera	Geolocation	Notification
Normal Chrome	●	●	●	●
Chrome with "NO Permission" enabled	○	○	○	○

● : Attackable, ○ : Not attackable

Table 27 OS and Browser versions.

Platform	OS version	Brave version
Windows	Windows10 21H2	1.45.118
Linux	20.04.5 LTS	1.45.118
Mac	macOS 13.0	1.45.118
Android	12	1.45.120
iOS	iOS 16.0 (Not JailBreak)	1.44.1†

† The version of Brave on iOS is not linked to versions on other platforms.

Table 28 Sharing of permission state between different browsing modes (after fixed).

Order	Permission state	Tab Status	Brave				
			W	L	M	A	i
Normal → Private	Denied	Open	○	○	○	○	G

● : Permission state of all resources is shared. ○ : Permission state of all resources is not shared.
G : Geolocation permission state is shared. W: Windows, L: Linux, M: macOS, A: Android, i: iOS

the permission-clearing mechanism is used (Section 4.5), so that users are correctly aware of this implementation [67]. Microsoft has informed us that they are working on implementing a fix. Google is treating our report as a high priority and is currently discussing it internally. Mozilla has separated our report into issues for each platform. The corresponding teams in charge are reviewing each issue. Apple is currently reviewing our report.

The details of responsible disclosure process are available on our website [43].

Evaluation of Fixed Vulnerabilities. In this section, we re-evaluate the vulnerabilities that were fixed as a result of our responsible disclosure. We verified that the vulnerabilities for which browser vendors reported fixes were correctly fixed. We received reports that on platforms other than iOS, the implementation of a takeover of denied permission states across browsing modes has been fixed in the Brave browser, as shown in Section 4.4. The release note for Brave browser v1.45.113 [68] shows that a fixed version of the Brave browser is publicly available. We manually measured the implementation of the modified version of the Brave browser for the test scenario T_3 in Section 4.4. The measurement results are shown in **Table 28**. These results demonstrate that in the fixed Brave browser, the denied permission state is not shared across browsing modes for all permission types. Table 5 shows that the permission state across browsing modes is inherited when Order is from normal browsing mode to private browsing mode and Tab Status is Open. The measurements in this section targeted this condition. **Table 27** shows the Brave and OS versions to be measured.

8.4 Limitations

Coverage of Permission Types. This study selected Microphone, Camera, and Geolocation permissions because they are used to control resources directly associated with user privacy. Additionally, the Notification was selected as permission, accounting for 74% of all permission requests in the real world [69]. However, some minor Web APIs that require permission (such as the clipboard API and idle detection API) have not been studied. With the Permium framework, it is straightforward to study newly implemented features of the web browser or other web APIs. We leave the analysis of other permission types for a future study.

Persistence of Permission States. The duration for which our proposed permission-based user tracking attack is effective depends on the persistence of the permission state stored by the browser. We confirmed that some web browsers provide users with the option to choose whether to remember the permission state in permission prompts. Our measurement study adopted only the default settings, although the behavior may change depending on the options.

Note that in chromium-based browsers, if the permission state "denied" is stored by repeatedly ignoring a prompt, the state is maintained for one week [70]. Our experiments found that after a week had passed, ignoring the prompt just once would cause the stored permissions to remain denied again. The attacker can still use the difference in the number of required ignorings to conduct the permission-based user tracking attack. The attacker first uses a special bit to determine whether the user is one of the following: A, a user who has visited within a week (permission already denied); B, a user who has visited before a week (permission denied after a single prompt ignore); C, a user who has never visited the site (permission is not denied after ignoring the prompt once). If the visitor is A, the attacker can be tracked by applying the same decoding step as in Section 6.1.2; for B, the attacker can be tracked by repeating the same procedure after ignoring permissions once for all tracked sites. For C, the attacker must only apply the encoding step. Therefore, this attack can continue to have tracking effects, even if over a week has passed since the user's previous visit. We discuss the implementation of permission state persistence in each browser in Section 7.1.3.

We note that the user study in Section 5 does not adequately take into account the expertise and knowledge of the users. Future research should classify users not only based on their familiarity with private browsing mode but also consider their purpose and frequency of use, as well as their privacy awareness. It should also understand developers' intentions and website administrators' expectations regarding the browser permission mechanisms.

9. Related Work

Web Tracking. Several methods have been proposed for tracking users visiting websites. Solomos et al. proposed and evaluated a novel tracking technique for tracking users without cookies by leveraging favicons [71]. This tracking method is effective because it is persistent and allows websites to write and read IDs to track users in only 2 seconds. Klein and Pinkas proposed and evaluated a novel user-tracking method that leverages the DNS caching mechanism and assigns unique DNS records to users [72]. The DNS cache is used for tracking and is shared among various browsers on the same device. This method allows an attacker to track users across multiple browsers and browsing modes. Koop et al. conducted the first large-scale study on user tracking using redirects [73]. This study crawled websites in the Alexa top 50k and revealed 100 redirect domains. The results show that 11.6% of websites in the Alexa Top 50k have at least one link that leads to the redirect domain. In this study, we propose a novel web-tracking technique that leverages permission mechanisms.

Cross-Browser Analysis. Several existing studies have revealed threats by analyzing and comparing the behavior of several different web browsers. Franken et al. developed a framework to evaluate policy implementation for third-party requests, and analyzed it for seven browsers and 46 extensions [74]. The analysis found that browser features such as PDF rendering libraries and prerendering functionality leak third-party cookies. Luo et al. developed Hindsight as a framework for identifying UI vulnerabilities in mobile browsers [75]. This study analyzed 27 attack building blocks (ABBs) that attackers could use in their attacks, covered 128 browser families and 2,324 individual browser versions. The results show that 2,292 (98.6%) of the 2,324 browser versions were vulnerable to at least one ABBs. Wu et al. performed a comparative analysis of the implementation of private browsing modes on desktop and mobile versions of browsers [76]. The study found inconsistencies between different browsers and the desktop and mobile versions of the same browser. This study also evaluated a browser fingerprinting attack and showed that an attacker could link user sessions in private browsing modes. Using the automated analysis framework we developed, our study investigated permission mechanisms across multiple operating systems and web browsers.

Mobile Permission. While our study is the first systematic analysis of permission mechanisms for web browsers, many studies have investigated permission mechanisms in mobile operating systems, typified by Android smartphones. Tuncay et al. proposed "false transparency attacks" in the runtime permission model introduced for Android. This attack allows an attacker to obtain illegitimate permissions from a target by layering a transparent malicious app on top of the other apps [11]. Marforio et al. proposed the "application collusion attack," which allows apps that have not obtained permissions to indirectly perform operations that require permissions through collusion between apps [12]. Chia et al. evaluated the characteristics of apps that require many permissions, the information users use to make permission decisions, and their effectiveness through an extensive survey of Facebook apps, Chrome extensions, and Android apps [4]. Bonné et al. conducted a user study with 157 participants to identify user decision-making when presented with permission prompts in an Android runtime permission model [8]. The study found that user decision-making depends on the app's functionality, whether the app really needs permission or whether the user needs the functionality associated with the permission. Cao et al. conducted a large survey of 1,719 participants from ten countries and regions to determine users' behaviors, expectations, and engagement with permissions in the Android runtime permission model [9].

10. Conclusion

We developed Permium, a web browser analysis framework that automatically analyzes the browser implementations of the permission mechanism. Using the Permium framework, we conducted a large-scale measurement study of permission mechanism implementations in 22 browsers running on five different operating systems to understand their behaviors and inconsistencies. We found that browser implementations are highly fragmented. Even within the same browser, implementing a permission mechanism differs from one OS to another. We also found 191 implementation inconsistencies in the permission mechanism implementation that could threaten user privacy. We also propose two practical attacks that exploit the identified implementation inconsistencies in the permission mechanism. One attack is a user tracking attack that uses the permission state set by the attacker in the victim's browser for multiple websites. In the other attack, the attacker obtains permission by causing the user to misidentify the permission requester. We analyzed the threats of these attacks in detail by performing an automated analysis using our framework and several user studies. We identified the viability conditions under these attacks and show that they are highly feasible. We then summarized the requirements for permission mechanisms to be standardized by the web browser community and the inconsistencies in browser implementations that browser vendors should fix. We developed and validated a browser extension that mitigates the discovered threats. The Permium framework is expected to be useful in the standardization process and in sharing best practices among browser vendors, since it allows the systematic investigation of various browser implementations of permission mechanisms that exhibit complex behavior.

Acknowledgments The authors would like to thank the members of JPCERT/CC [64] for their cooperation in our responsible disclosure.

References

[1] Nomoto, K., Watanabe, T., Shioji, E., Akiyama, M. and Mori, T.: Browser Permission Mechanisms Demystified (to appear), *Network and Distributed System Security Symposium* (2023).

[2] Google: Permissions on Android — Android Developers (2021), available from ⟨https://developer.android.com/guide/topics/permissions/overview⟩.

[3] Apple: Accessing User Data - App Architecture - iOS - Human Interface Guidelines - Apple Developer (2021), available from ⟨https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data/⟩.

[4] Chia, P.H., Yamamoto, Y. and Asokan, N.: Is This App Safe? A Large Scale Study on Application Permissions and Risk Signals, *Proc. 21st International Conference on World Wide Web*, *WWW '12*, pp.311–320, Association for Computing Machinery (online), DOI: 10.1145/2187836.2187879 (2012).

[5] Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T. and Chen, Z.: AutoCog: Measuring the Description-to-Permission Fidelity in Android Applications, *Proc. 2014 ACM SIGSAC Conference on Computer and Communications Security*, *CCS '14*, pp.1354–1365, Association for Computing Machinery (online), DOI: 10.1145/2660267.2660287 (2014).

[6] Felt, A.P., Chin, E., Hanna, S., Song, D. and Wagner, D.: Android Permissions Demystified, *Proc. 18th ACM Conference on Computer and Communications Security*, *CCS '11*, pp.627–638, Association for Computing Machinery (online), DOI: 10.1145/2046707.2046779 (2011).

[7] Reinfelder, L., Schankin, A., Russ, S. and Benenson, Z.: An Inquiry into Perception and Usage of Smartphone Permission Models, *Trust, Privacy and Security in Digital Business*, Furnell, S., Mouratidis, H. and Pernul, G. (Eds.), pp.9–22, Springer International Publishing (2018).

[8] Bonné, B., Peddinti, S.T., Bilogrevic, I. and Taft, N.: Exploring decision making with Android's runtime permission dialogs using in-context surveys, *13th Symposium on Usable Privacy and Security* (*SOUPS 2017*), pp.195–210, USENIX Association (2017) (online), available from ⟨https://www.usenix.org/conference/soups2017/technical-sessions/presentation/bonne⟩.

[9] Cao, W., Xia, C., Peddinti, S.T., Lie, D., Taft, N. and Austin, L.M.: A Large Scale Study of User Behavior, Expectations and Engagement with Android Permissions, *30th USENIX Security Symposium* (*USENIX Security 21*), pp.803–820, USENIX Association (online), available from ⟨https://www.usenix.org/conference/usenixsecurity21/presentation/cao-weicheng⟩ (2021).

[10] Watanabe, T., Akiyama, M., Sakai, T. and Mori, T.: Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps, *11th Symposium On Usable Privacy and Security* (*SOUPS 2015*), pp.241–255, USENIX Association (2015) (online), available from ⟨https://www.usenix.org/conference/soups2015/proceedings/presentation/watanabe⟩.

[11] Tuncay, G.S., Qian, J. and Gunter, C.A.: See No Evil: Phishing for Permissions with False Transparency, *USENIX Security Symposium* (2020).

[12] Marforio, C., Francillon, A. and Capkun, S.: Application Collusion Attack on the Permission-Based Security Model and Its Implications for Modern Smartphone Systems, *Department of Computer Science*, ETH Zurich (2010).

[13] Liu, B., Andersen, M.S., Schaub, F., Almuhimedi, H., Zhang, S.A., Sadeh, N., Agarwal, Y. and Acquisti, A.: Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions, *12th Symposium on Usable Privacy and Security* (*SOUPS 2016*), pp.27–41, USENIX Association (2016) (online), available from ⟨https://www.usenix.org/conference/soups2016/technical-sessions/presentation/liu⟩.

[14] Zhang, Y., Yang, M., Gu, G. and Chen, H.: Rethinking Permission Enforcement Mechanism on Mobile Systems, *IEEE Trans. Information Forensics and Security*, Vol.11, No.10, pp.2227–2240 (online), DOI: 10.1109/TIFS.2016.2581304 (2016).

[15] Qu, Y., Du, S., Li, S., Meng, Y., Zhang, L. and Zhu, H.: Automatic Permission Optimization Framework for Privacy Enhancement of Mobile Applications, *IEEE Internet of Things Journal*, Vol.8, No.9, pp.7394–7406 (online), DOI: 10.1109/JIOT.2020.3039472 (2021).

[16] Gao, X., Liu, D., Wang, H. and Sun, K.: PmDroid: Permission Supervision for Android Advertising, *2015 IEEE 34th Symposium on Reliable Distributed Systems* (*SRDS*), pp.120–129 (online), DOI: 10.1109/SRDS.2015.41 (2015).

[17] Mozilla: Permissions API - Web API — MDN (2021), available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API⟩.

[18] W3C: Requesting Permissions (2022), available from ⟨https://wicg.github.io/permissions-request/⟩.

[19] StatCounter: Browser Market Share Worldwide — Statcounter Global Stats (2022), available from ⟨https://gs.statcounter.com/browser-market-share/⟩.

[20] W3C: Geolocation API (2021), available from ⟨https://www.w3.org/TR/geolocation/⟩.

[21] W3C: Media Capture and Streams (2021), available from ⟨https://www.w3.org/TR/mediacapture-streams/#dom-mediadevices-getusermedia⟩.

[22] WHATWG: HTML Standard (2022), available from ⟨https://html.spec.whatwg.org/multipage/origin.html⟩.

[23] Mozilla: Origin - MDN Web Docs Glossary: Definitions of Web-related terms — MDN (2021), available from ⟨https://developer.mozilla.org/en-US/docs/Glossary/Origin⟩.

[24] Mozilla: Same-origin policy - Web security — MDN (2022), available from ⟨https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy⟩.

[25] W3C: Permissions (2021), available from ⟨https://www.w3.org/TR/permissions/⟩.

[26] Gaunt, M.: Permissions API for the Web (2019), available from ⟨https://developer.chrome.com/blog/permissions-api-for-the-web/⟩.

[27] W3C: Private Mode Browsing, available from ⟨https://w3ctag.github.io/private-mode/⟩ (2018).

[28] W3C: W3C TAG Observations on Private Browsing Modes (2019), available from ⟨https://www.w3.org/2001/tag/doc/private-browsing-modes/⟩.

[29] Google: How private browsing works in Chrome (2022), available from ⟨https://support.google.com/chrome/answer/7440301⟩.

[30] Microsoft: Browse InPrivate in Microsoft Edge, available from ⟨https://support.microsoft.com/en-us/microsoft-edge/browse-inprivate-in-microsoft-edge-cd2c9a48-0bc4-b98e-5e46-ac40c84e27e2⟩ (2022).

[31] Software Freedom Conservancy: Selenium overview — Selenium (2021), available from ⟨https://www.selenium.dev/documentation/overview/⟩.

[32] Google Developers: Puppeteer — Tools for Web Developers — Google Developers (2021), available from ⟨https://developers.google.com/web/tools/puppeteer⟩.

[33] Microsoft: Fast and reliable end-to-end testing for modern web apps — Playwright (2022), available from ⟨https://playwright.dev/⟩.

[34] Apple: WebDriver is Coming to Safari in iOS 13 — WebKit (2019), available from ⟨https://webkit.org/blog/9395/webdriver-is-coming-to-safari-in-ios-13/⟩.

[35] Software Freedom Conservancy: Install browser drivers — Selenium (2022), available from ⟨https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/⟩.

[36] Mozilla: Notification - Web APIs — MDN (2022), available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/Notification⟩.

[37] Mozilla: Incognito browser: What it really means (2022), available from ⟨https://www.mozilla.org/en-US/firefox/browsers/incognito-browser/⟩.

[38] Brave Help Center: What is a Private Window? (2022), available from ⟨https://support.brave.com/hc/en-us/articles/360017840332-What-is-a-Private-Window-⟩.

[39] Apple: Use Private Browsing in Safari on Mac (2022), available from ⟨https://support.apple.com/guide/safari/browse-privately-ibrw1069/mac⟩.

[40] Mozilla: Manage local site storage settings — Firefox Help (2022), available from ⟨https://support.mozilla.org/en-US/kb/storage⟩.
[41] Mozilla: Delete browsing, search and download history on Firefox — Firefox Help (2022), available from ⟨https://support.mozilla.org/en-US/kb/delete-browsing-search-download-history-firefox⟩.
[42] Lancers, inc.: Lancers (2022), available from ⟨https://www.lancers.jp/⟩.
[43] Nomoto, K.: PERMIUM (Password: ne4jpp75z4xfxk, which will be publicly available later.) (2022), available from ⟨https://permium.seclab.jp/⟩.
[44] Apple: Full Third-Party Cookie Blocking and More (2020), available from ⟨https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/⟩.
[45] Google: Chromium Blog: Building a more private web: A path towards making third party cookies obsolete (2020), available from ⟨https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html⟩.
[46] Google: Permission Delegation - Chrome Platform Status (2021), available from ⟨https://chromestatus.com/feature/5670617353289728⟩.
[47] Mozilla: GlobalEventHandlers.onclick - Web APIs — MDN (2022), available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick⟩.
[48] Mozilla: Document: visibilitychange event - Web APIs — MDN (2022), available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/Document/visibilitychange_event⟩.
[49] Mozilla: The WebSocket API (WebSockets) - Web APIs — MDN (2022), available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API⟩.
[50] Google: WebRTC (2019), available from ⟨https://webrtc.org/⟩.
[51] Google: Temporarily stop permission requests after 3 dismissals - Chrome Platform Status (2022), available from ⟨https://chromestatus.com/feature/6443143280984064⟩.
[52] World Wide Web Consortium (W3C) (2022), available from ⟨https://www.w3.org/⟩.
[53] WHATWG: Web Hypertext Application Technology Working Group (WHATWG) (2022), available from ⟨https://whatwg.org/⟩.
[54] Mozilla: How to manage your camera and microphone permissions with Firefox — Firefox Help (2022), available from ⟨https://support.mozilla.org/en-US/kb/how-manage-your-camera-and-microphone-permissions⟩.
[55] W3C: Permissions Policy (2020), available from ⟨https://www.w3.org/TR/permissions-policy-1/⟩.
[56] Google: Change site permissions - Computer - Google Chrome Help (2022), available from ⟨https://support.google.com/chrome/answer/114662⟩.
[57] Brave: How do I change site permissions? -Brave Help Center (2022), available from ⟨https://support.brave.com/hc/en-us/articles/360018205431-How-do-I-change-site-permissions-⟩.
[58] Apple: Change Websites preferences in Safari on Mac - Apple Support (CA) (2022), available from ⟨https://support.apple.com/en-ca/guide/safari/ibrwe2159f50/mac⟩.
[59] Apple: WebKit (2022), available from ⟨https://webkit.org/⟩.
[60] Apple: App Store Review Guidelines - Apple Developer (2021), available from ⟨https://developer.apple.com/app-store/review/guidelines/⟩.
[61] cowboy: cowboy/javascript-hooker (2014), available from ⟨https://github.com/cowboy/javascript-hooker⟩.
[62] chromium: chromium/permission.site (2021), available from ⟨https://github.com/chromium/permission.site⟩.
[63] z0ccc: z0ccc/extension-fingerprints (2022), available from ⟨https://github.com/z0ccc/extension-fingerprints⟩.
[64] JPCERT Coordination Center (2022), available from ⟨https://www.jpcert.or.jp/english/⟩.
[65] Brave: Don't inherit permissions in private windows · Issue #24720 · brave/brave-browser (2022), available from ⟨https://github.com/brave/brave-browser/issues/24720⟩.
[66] Brave: Don't inherit privacy-sensitive content settings in incognito. by goodov · Pull Request #14765 · brave/brave-core (2022), available from ⟨https://github.com/brave/brave-core/pull/14765⟩.
[67] Brave: Inform users that they need to close private windows to clear data in them · Issue #25046 · brave/brave-browser (2022), available from ⟨https://github.com/brave/brave-browser/issues/25046⟩.
[68] Brave: Release v1.45.113 (Chromium 107.0.5304.62) (2022), available from ⟨https://github.com/brave/brave-browser/releases/tag/v1.45.113⟩.
[69] Bilogrevic, I., Engedy, B., Porter, J., Taft, N., Hasanbega, K., Paseltiner, A., Lee, H.K., Jung, E., Watkins, M., McLachlan, P. and James, J.: "Shhh…be quiet!" Reducing the Unwanted Interruptions of Notification Permission Prompts on Chrome, *30th USENIX Security Symposium* (*USENIX Security 21*), pp.769–784, USENIX Association (2021) (online), available from ⟨https://www.usenix.org/conference/usenixsecurity21/presentation/bilogrevic⟩.
[70] Status in Chromium (2022), available from ⟨https://chromestatus.com/feature/6443143280984064⟩.
[71] Solomos, K., Kristoff, J., Kanich, C. and Polakis, J.: Tales of favicons and caches: Persistent tracking in modern browsers, *Network and Distributed System Security Symposium* (2021).
[72] Klein, A. and Pinkas, B.: DNS Cache-Based User Tracking, *Network and Distributed System Security Symposium* (2019).
[73] Koop, M., Tews, E. and Katzenbeisser, S.: In-Depth Evaluation of Redirect Tracking and Link Usage, *Proc. Privacy Enhancing Technologies*, Vol.2020, pp.394–413 (2020).
[74] Franken, G., Goethem, T.V. and Joosen, W.: Who Left Open the Cookie Jar? A Comprehensive Evaluation of Third-Party Cookie Policies, *27th USENIX Security Symposium* (*USENIX Security 18*), pp.151–168, USENIX Association (2018) (online), available from ⟨https://www.usenix.org/conference/usenixsecurity18/presentation/franken⟩.
[75] Luo, M., Starov, O., Honarmand, N. and Nikiforakis, N.: Hindsight: Understanding the Evolution of UI Vulnerabilities in Mobile Browsers, *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security*, *CCS '17*, pp.149–162, Association for Computing Machinery (online), DOI: 10.1145/3133956.3133987 (2017).
[76] Yuanyi, W., Meng, D. and Chen, H.: Evaluating private modes in desktop and mobile browsers and their resistance to fingerprinting, pp.1–9 (online), DOI: 10.1109/CNS.2017.8228636 (2017).
[77] julioverne: julioverne/screendump (2021), available from ⟨https://github.com/julioverne/screendump⟩.
[78] Pallets: Welcome to Flask; Flask Documentation (2.1.x) (2022), available from ⟨https://flask.palletsprojects.com/en/2.1.x/⟩.
[79] Peng, B., Fan, H., Wang, W., Dong, J., Li, Y., Lyu, S., Li, Q., Sun, Z., Chen, H., Chen, B., Hu, Y., Luo, S., Huang, J., Yao, Y., Liu, B., Ling, H., Zhang, G., Xu, Z., Miao, C., Lu, C., He, S., Wu, X. and Zhuang, W.: DFGC 2021: A DeepFake Game Competition (2021).
[80] Li, Y., Yang, X., Sun, P., Qi, H. and Lyu, S.: Celeb-DF: A Large-scale Challenging Dataset for DeepFake Forensics, *IEEE Conference on Computer Vision and Patten Recognition* (*CVPR*) (2020).

Appendix

A.1 Details of the Permium Framework

We describe the details of the Permium framework.

Setup of the Permium Framework. We used three desktop PCs and two smartphones as client devices. As desktop OSs, Windows, Linux, and macOS were installed on the three PCs. As mobile OSs, Android and iOS were installed on the two smartphones. **Table A·1** summarizes the details of each client device and OS version. To establish remote controlling on iOS, screendump [77] was installed on a jailbroken iOS. For browsers, we used Chrome, Firefox, Edge, Brave, and Safari as major browsers. In summary, the number of OS/browser combinations is 22, because Safari is only available on macOS and iOS. **Table A·2** lists the web browser versions. The entire framework was implemented in Python and the Flask framework [78].

Mechanism for Determining the Permission Status. PERMIUM uses a JavaScript API to get the permission status of the web browsers. PERMIUM attempts to access each resource protected by permissions through the JavaScript API. At this time, PERMIUM determines whether the permission status is granted, denied, or unset by observing the property

Table A·1 Client device details and OS versions

Type	Device Name	OS Version
Desktop	Acer Veriton X490	Windows 10 21H1
Desktop	Home-built computer	Ubuntu 20.04 LTS
Desktop	MacBook Pro 15-inch Mid 2015	macOS 12.0.1
Mobile	LG G8X ThinQ	Android 10
Mobile	iPhone X	iOS 14.3 (JailBreak)

value or callback function. This checking process is available on all operating systems and browsers and does not affect the evaluation results.

Mechanism for Operating a Web Browser. PERMIUM operates browsers from hard-coded images of button icons that each web browser displays. We manually extracted information about the button-icon images and their corresponding operations (e.g., allow, deny, close a dialogue) from all the browsers and registered the information in the framework. With this information, the framework can automatically analyze the browser behaviors. To interact with a browser, the framework identifies a button-icon image corresponding to a certain operation by matching the registered images and clicking it.

Availability of PERMIUM Framework. We share artifacts with researchers based on their application on our website [43]. Applications will be accepted from researchers who wish to use the artifact, which will shared through a review process. It should be noted that the artifact-sharing model is widely adopted in the research community. For example, Celeb-DF [79], [80], a widely used benchmark dataset for Deepfake.

A.2 Breakdown of the Identified Implementation Inconsistencies

We counted the implementation inconsistencies identified in Section 4 for each of the 22 browsers. **Table A·3** presents the breakdown of the implementation inconsistencies identified for each browser. We can see that each browser has, on average, 8.5 implementation inconsistencies.

Table A·2 Browser versions used in this study

Platform	Browser	Version
Windows	Chrome	96.0.4664
	Firefox	94.0.2
	Edge	95.0.1020
	Brave	1.31.91
Linux	Chrome	94.0.4606
	Firefox	94.0
	Edge	95.0.1020
	Brave	1.30.87
macOS	Chrome	96.0.4664
	Firefox	94.0.1
	Edge	95.0.1020
	Brave	1.31.91
	Safari	15.1
Android	Chrome	95.0.4638
	Firefox	94.1.2
	Edge	95.0.1020
	Brave	1.31.90
iOS	Chrome	95.0.4638
	Firefox	39.0
	Edge	93.0.961
	Brave	1.32
	Safari	604.1

A.3 Conditions for Automatically Setting the Permission State to "Denied"

The number of times the prompt needs to be ignored to automatically set the permission state to "denied" is shown in **Table A·4**. It is clear that the number of times required differs between desktop browsers and mobile browsers. In desktop browsers, most browsers automatically set permissions with 4 prompt-ignores, while in mobile browsers, most browsers automatically set permissions with 3 prompt-ignores. In Safari, Microphone, Camera, and Geolocation permissions were automatically denied after three ignores, while Notification permission was automatically denied after one ignore.

A.4 Wait Time T_{wait} Required for the Permission-based User Tracking Attack

In the encoding process of the permission-based user tracking attack, the browser needs to wait for a moment when a permission prompt is required. If the browser reloads the tracking website before the permission prompt is rendered in the foreground or background, the number of times that the prompt is ignored will not be incremented, and the permission state for the tracking website cannot be denied. The waiting time is defined as T_{wait} seconds. This section evaluates T_{wait} for devices A, B, and C during a normal attack, and T_{wait} for device B during a packing multi permissions. In the normal attack, Microphone permission was used, and in the packing multi permissions, Microphone and Camera permissions were used.

We measured the number of attack success rates when $T_{wait} = 20, 40, \ldots, 100$. Each attack was carried out 10 times. If all the attack success rates were 1, we further measure the attack success rate when $T_{wait} = 0, 2, \ldots, 20$. If all attack success rates are less than 1, then further measure the attack success rate when $T_{wait} = 120, 140, \ldots, 200$.

Figure A·1 presents the relationship between T_{wait} and the attack success rate. The T_{wait} for deice A, B, and C during normal attacks were 160 ms, 2 ms, and 40 ms, respectively. In the packing multi permissions case, T_{wait} was 2 ms. These results show that T_{wait} is small, less than 0.2 seconds, in all cases. On the other hand, the difference in time by device suggests that T_{wait} varies depending on the operating system and browser of the attack target and the connection method of the camera and microphone. Also, it can be seen that packing multiple permissions does not

Table A·3 Breakdown of the identified implementation inconsistencies. For T_1, cases that do not support permission settings are excluded.

	Chrome					Firefox					Edge					Brave					Safari		SUM
	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	W	L	M	A	i	M	i	
T_1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
T_2	2	2	2	2	4	2	2	3	0	4	2	2	2	2	4	0	0	0	0	4	4	2	45
T_3	2	2	2	2	8	0	0	0	4	8	2	2	2	2	8	1	1	1	1	8	8	0	64
T_4	1	1	1	1	2	2	2	2	0	2	1	1	1	1	2	1	1	1	1	2	0	0	26
T_5	4	4	4	3	0	0	0	0	0	0	4	4	4	3	0	4	4	4	3	0	4	0	49
T_6	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	4
SUM	9	9	9	8	15	5	5	6	4	15	9	9	9	8	14	6	6	6	5	15	17	2	191

W: Windows, L: Linux, M: macOS, A: Android, i: iOS

Table A·4 Number of times the prompt needs to be ignored to automatically set the permission state to "denied".

	Chrome	Firefox	Edge	Brave	Safari
Windows	4	–	4	4	n/a
Linux	4	–	4	4	n/a
macOS	4	–	4	4	†
Android	3	–	3	3	n/a
iOS	–	–	–	–	–

4: Permission is automatically set to denied after ignoring the prompt 4 times.
3: Permission is automatically set to denied after ignoring the prompt 3 times.
† : Microphone, Camera, and Geolocation are automatically set to denied after ignoring the prompt four times, and for Notification, permission is automatically set to denied after ignoring the prompt once.
– : Permission state will not be automatically set to denied by ignoring the prompt.

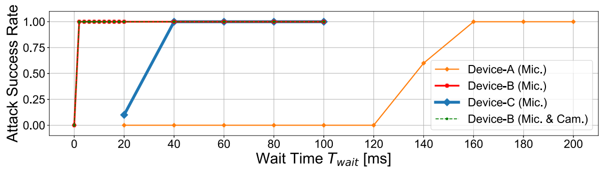
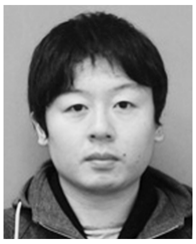


Fig. A·1 Relationship between T_{wait} and attack success rate.

have an effect on processing time, even though two permissions are requested at the same time.

Kazuki Nomoto was born in 1999. He received his B.E. degree in engineering from department of communications and computer engineering at Waseda University in 2021. He is currently a master's student in the department of computer science and communications engineering of Waseda University. His research interests are web security and autonomous vehicle security. He is a member of the IEEE.

Takuya Watanabe received B.E. and M.E. degrees in computer science and engineering, and a Ph.D. in engineering from Waseda University in 2014, 2016, and 2020, respectively. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2016, he has been engaged in research of system security and privacy from an attacker's perspective, especially web and mobile. He is now with the Cyber Security Project of NTT Social Informatics Laboratories.



Eitaro Shioji received his B.E. degree in Computer Science and M.E. degree in Communications and Integrated Systems from Tokyo Institute of Technology in 2008 and 2010, respectively. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2010, he has been engaged in research and development on cyber security. His research interests include systems and software security.

Mitsuaki Akiyama received his M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology in 2007 and 2013. Since joining Nippon Telegraph and Telephone Corporation (NTT) in 2007, he has been engaged in research and development on cybersecurity. He is currently a Senior Distinguished Researcher at NTT Social Informatics Laboratories. He received Cybersecurity Encouragement Award of the Minister for Internal Affairs and Communications in 2020 and IPSJ/IEEE Computer Society Young Computer Researcher Award in 2022. His research interests include cybersecurity measurement, offensive security, and usable security and privacy. He is a senior member of IPSJ and a member of IEEE and IEICE.

Tatsuya Mori is currently a professor at Waseda University, Tokyo, Japan. He received B.E. and M.E. degrees in applied physics, and Ph.D. degree in information science from the Waseda University, in 1997, 1999 and 2005, respectively. He joined NTT lab in 1999. Since then, he has been engaged in the research of measurement and analysis of networks and cyber security. From Mar 2007 to Mar 2008, he was a visiting researcher at the University of Wisconsin-Madison. From May 2018, he is a guest researcher at Cybersecurity Research Institute, National Institute of Information and Communications Technology (NICT). From April 2019, he is a visiting researcher at Center for Advanced Intelligence Project, RIKEN (RIKEN AIP). He received Telecom System Technology Award from TAF in 2010 and Best Paper Awards from IEICE transactions, IEEE/ACM COMSNETS, ATIS, NDSS, and EuroUSEC in 2009, 2010, 2017, 2020, and 2021, respectively. Dr. Mori is a member of ACM, IEEE, IEICE, and IPSJ.