Browser Permission Mechanisms Demystified

Kazuki Nomoto*, Takuya Watanabe†, Eitaro Shioji†, Mitsuaki Akiyama† and Tatsuya Mori*‡§
*Waseda University
Email: {nomotokazuki,mori}@nsl.cs.waseda.ac.jp
†NTT Social Informatics Laboratories
Email: {takuya.watanabe.yf, eitaro.shioji.es}@hco.ntt.co.jp, akiyama@ieee.org
‡National Institute of Information and Communications Technology
§RIKEN Center for Advanced Intelligence Project

***Abstract*—Modern Web services provide rich content by accessing resources on user devices, including hardware devices such as cameras, microphones, and GPSs. Web browser vendors have adopted permission mechanisms that achieve appropriate control over access to such resources to protect user privacy. The permission mechanism gives users the ability to grant or deny their browser access to resources for each website. Despite the importance of permission mechanisms in protecting user privacy, previous studies have not been conducted to systematically understand their behavior and implementation. In this study, we developed PERMIUM, a web browser analysis framework that automatically analyzes the behavior of permission mechanisms implemented by various browsers. Using the PERMIUM framework, we systematically studied the behavior of permission mechanisms for 22 major browser implementations running on five different operating systems, including mobile and desktop. We determined that the implementation and behavior of permission mechanisms are fragmented and inconsistent between operating systems, even for the same browser (i.e., Windows Chrome vs. iOS Chrome) and that the implementation inconsistencies can lead to privacy risks. Based on the behavior and implementation inconsistencies of the permission mechanism revealed by our measurement study, we developed two proof-of-concept attacks and evaluated their feasibility. The first attack uses the permission information collected by exploiting the inconsistencies 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 incorrectly grants permission to a malicious site. 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.**

I. INTRODUCTION

Modern Web services can provide dynamic and rich content, such as location-based recommendations and online conferencing services, by enabling the browser to access hardware resources, such as the cameras and GPSs on user devices. On the other hand, granting web browsers access to hardware resources without any restrictions may cause unintended security/privacy risks. The permission function is generally provided as an access control mechanism to protect user privacy. By configuring the permissions, users expect web browsers to appropriately control which resources they can access, thereby protecting user privacy.

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 [1], [2]. 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 [3], [4], [5], user perception [6], [7], [8], [9], unauthorized privilege acquisition [10], [11], and the proposals of new permission mechanisms [12], [13], [14], [15]. 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 erroneous implementation of the permission mechanism risks unintentional disclosure of user privacy information. Web Permission was developed for each API (MediaDevice API, Geolocation API, etc.) that utilizes the functions of individual devices, such as cameras and GPSs [16]. For this reason, each API has its specification for the permission mechanism. Owing to the background mentioned, the implementations of Web Permission differ depending on the type of API and browser. Such fragmentation of the implementation risks user confusion and exploitation of the permission mechanism by attackers. To avoid the fragmented implementations of the permission mechanism, "Permissions API" was proposed to manage various permissions in a unified manner across various platforms [17]. 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. (For more details on the Permissions API, see Appendix XI-D.)

This study aims to conduct a large-scale measurement study of web permission implementations to identify differences in behavior among implementations, as well as implementation inconsistencies and vulnerabilities that lead to 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 imple-

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23109
www.ndss-symposium.org

mentation 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 [18]. 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 combine 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, cross-platform browser analysis.
- 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 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 identified fixes for the inconsistencies present in the implementation of web permission mechanisms and provided recommendations for stakeholders.

II. 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.

A. 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 [19], which provides the user's location, and the MediaDevices API [20], 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 [21]. 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 [22]. 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 [23].

Typically, the Web browser maintains the permission state, i.e., 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 the previous time. Formally, the permission has the following three states: "prompt," "granted," and "denied." "prompt" is a state in which the user has not selected a permission state. The default state of the permission state is "prompt." "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).

B. 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; thus, these are expected to protect user privacy [24]. 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 [25]. The name of this feature also varies between browsers; that is, it is called "Incognito mode" in Chrome and "InPrivate window" in Edge [26], [27]. In this paper, we use the term "private browsing modes," as used in the W3C [25].

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

III. 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 as generic such that it can run on various OSs/browsers.

A. 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.

B. Components of the PERMIUM Framework

In the following, we describe the components of the PERMIUM framework shown in Figure 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 methods are divided into three types

```
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)
```

Listing 1. A Test Scenario

TABLE I. 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

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 I 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 XI-A 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.

C. 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 [28], [29], [30]. For iOS, existing browser measurement frameworks do not support autopilot for browsers other than Safari [31],

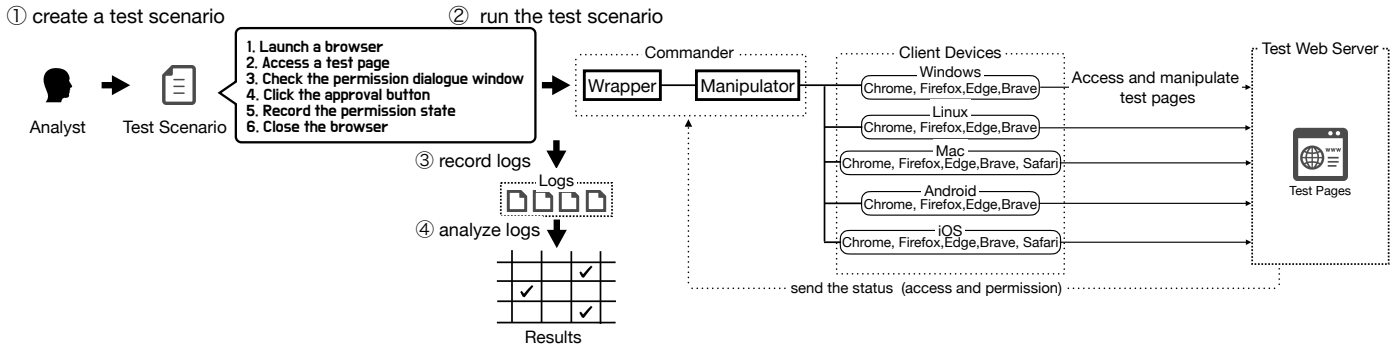

Fig. 1. Overview of the PERMIUM Framework.

[32]. 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 XI-A.

IV. 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.

A. 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?*
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 VIII-A discusses solutions against privacy risks determined in our study.

B. 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 II 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 [33]. 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 a problem itself; a mechanism is necessary to hide these differences from the website and prevent them from being used for browser fingerprinting. Furthermore, in Firefox private browsing modes, the permission states set by the user should be reflected properly.

C. 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 III summarizes the measurement results. In normal browsing modes, the permission state persisted in many cases. In private browsing modes, we may have expected that the state of granted permissions would not be persistent, i.e., the state set would revert to "prompt" when 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,

TABLE II. 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 (Microphone, Camera, Geolocation, and Notification)
N : Notification permission is unsupported. Other permissoins 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 III. 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

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 III suggests that many environments and conditions do not meet these user expectations.

Furthermore, the permission state set by a user for a website persisting implies that the website can determine if the user has visited the website before. That is, users risk website administrators inferring whether a user has accessed the site from the permission state. 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 vendors [26], [27], [34], [35], [36]. Inconsistencies between the specifications described in the documents and actual behaviors create user confusion and unintended risks.

D. 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 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 IV 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 to 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 to 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 II-B, because private browsing modes lack a standardized specification, behavior varies between browser vendor implementations. This situation creates a risk to confuse 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 mismatching user expectations and web browser behavior.

E. 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, that is, 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.

TABLE IV. 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 V. 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

Measurement Results. Table V 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 than in normal browsing modes; thus, they are unaffected by the clear-data method, which erases data stored in the storage space.

Implications. As demonstrated, deleting web browser settings does not necessarily remove the permission state. This behavior may oppose user expectations and risks the unintended leakage of privacy data. This implementation in Firefox does not follow the official documentation [37], [38]. 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.

F. 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.

[2]The number of reloads for which the permission state is automatically set to "denied" depends on the web browser and OS. Table XV 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 VI-A

Measurement Results. Table VI 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 VI-A.

G. 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 VII 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 the web browser screen where the active foreground page was displayed. In the special case, all iOS

TABLE VI. 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 VII. 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.
† 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

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 overlaid 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. 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 VI-A, 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 VII-B 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 VI-A. Furthermore, the overlaid display of prompts originated 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 XIV in Appendix XI-B 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.

V. 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 IV and the users' perceptions. The user study consisted of six surveys, $\mathbf{U_1}$–$\mathbf{U_6}$, which correspond to the six test scenarios T_1–T_6 defined in Section IV. In the following, we highlight only the key results of the survey due to space limitation. The details of the demographics are shown in Table XIX in Appendix XI-H. The detailed results of the following survey can be found in our website [39].

Permission Behaviors in the Browsing Modes ($\mathbf{U_2}, \mathbf{U_3}$). Among the users who were familiar with the private browsing mode, 70–80% expected that the permission state is not persistent in the private browsing mode and the permission state will not be inherited between browsing modes. However, as we have clarified in Section IV, all the web browsers examined in this study persist some permission states in the private browsing mode, and some of the permission states are inherited across the normal and the private browsing modes.

Data Deletion Mechanism ($\mathbf{U_4}$). More than 70 % of the users expected that the data deletion mechanism would clear the permission state. However, as we have clarified in Section IV, in Chrome, Firefox, Edge, and Brave, the permission state is retained even after the data deletion mechanism is applied.

Behavior When Permission Requests Are Ignored ($\mathbf{U_5}$). 60% of users were unaware that some browsers will automatically set the permission status denied when a website requests the permission several times and the requests are ignored. As we have clarified in Section IV, such a feature is implemented in all web browsers except Firefox.

VI. ATTACK CONCEPT

This section describes the concepts of two attacks based on the 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 at-

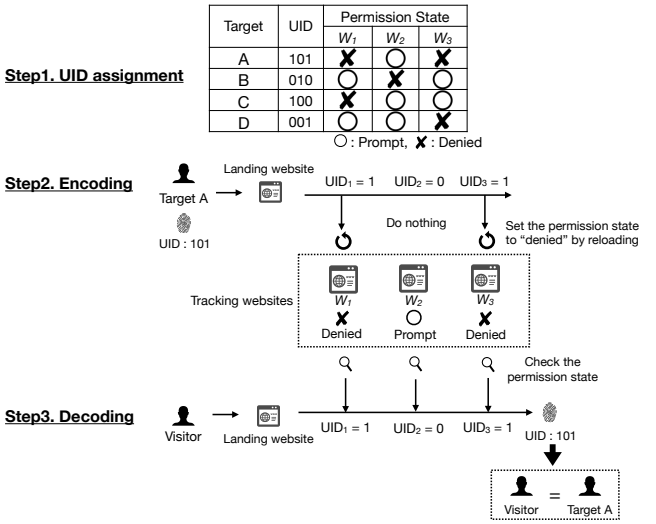

Fig. 2. Overview of permission-based user tracking attack.

tacker induces an erroneous decision to grant/deny permission by displaying a fraudulent overlay of permission prompts.

A. Permission-based User Tracking Attack

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 IV. 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 [40], [41], 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 IV-F. 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 between browsing modes, as shown in Section IV-D, which implies that an attacker can track a target across normal and private browsing modes.

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 2 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 require 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 makes the permission state for W_n to be "denied" by repeatedly reloading the tracking website. Malicious code lets 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 $2\,\mathrm{ms}$ to $160\,\mathrm{ms}$ required for a successful attack, detailed in Appendix XI-F.

Step 3: Decoding. When a user revisits the landing website, their browser loads malicious code, making the browser 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 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.

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 the 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

[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.

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 [42]. As 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, permision-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.

B. Permission-based Phishing Attack

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 IV. 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 overlayed 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.

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 are 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 [43], [44].

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 [45], [46], which allow them to monitor and collect information such as video, audio, and location of the target in real time.

VII. Feasibility of the Attack

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

A. Permission-based User Tracking Attack

1) Attack Targets: Attackable browsers vary depending on the attack conditions. Table VIII 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 IX 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 secretly obtain the permission state of the browser without displaying a permission prompt. These APIs are available in all browsers except iOS. Appendix XI-D presents the details of our survey. We investigated the availability of packing multiple permission requests for Microphone and Camera permissions and found it in all browsers on desktop operating systems except Safari. The details of our survey are presented in Appendix XI-E.

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 2GB 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 3 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

TABLE VIII. 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 IX. 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

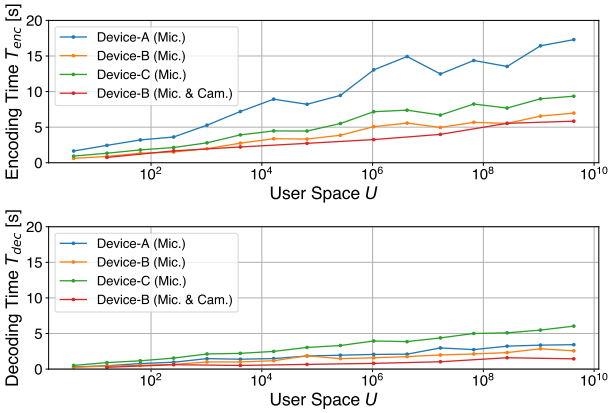

Fig. 3. U vs. required time. Top: encoding and Bottom: decoding.

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. 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 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.

Appendix XI-G presents several other methods that can further improve the attack efficiency.

B. 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.

1) Attack Targets: Table X summarizes the attackable targets for each attack condition. The attack applies to iOS browsers such as Firefox and Brave.

TABLE X. 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.

TABLE XI. 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

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 [47], 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 XX in Appendix XI-H. The questionnaire used for the online survey is available on our website [39].

Our user study consisted of the following three experiments: $\mathbf{E_1}$, $\mathbf{E_2}$, and $\mathbf{E_3}$.

$\mathbf{E_1}$: 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 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. In the "Fake" scenario, the foreground tab of the browser shows the results of a search for Italian restaurants

using Google. There is an overlay of a permission request dialog invoked by the attack site running in the background tab on this tab. Table XI (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.

$\mathbf{E_2}$: Reason to Grant or Deny a Permission Request. This experiment aims to answer the following question: *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 XI (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.

$\mathbf{E_3}$: 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 XI (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 55% of entire users are potential attack targets. Note, we can directly compute the conditional joint probability as 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.,
$P(C_1|C_0)P(C_2|C_0) = 68/76 \times 60/76 \approx P(C_1, C_2|C_0)$.

VIII. Discussion

A. Toward the Fundamental Solutions

Sections IV, VI, and VII 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 [48] 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 [49], [50] and sharing of best practices among browser vendors are needed to solve this problem. In fact, our user study, presented in Section V, revealed the mismatch between the user expectation and browser implementations. The standardization process will be the 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 VI and VII, 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 [51]. However, they are incomplete, as 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 [42] in the working draft of the Permissions Policy [52]. 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 permission state improves the transparency of permission mechanisms for users. R_4 has been implemented in many browsers [53], [51], [54], [55]. 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 stored permission state when exiting private browsing mode.
- F_3: Do not overlay a permission prompt screen sent from a 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 [56]. Apple's policy is that all iOS browsers must be developed using WebKit [57]. 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 IV, 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 confounds 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.

B. Ethical Considerations

User Study. We followed the policy set forth by our organization's IRB and confirmed that our user study falls under the exemption of IRB review. Our user study did not collect sensitive information about participants. In the experiment shown in Appendix XI-G2, 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 actions within two weeks of receiving our report. Furthermore, we had extensive discussion with JPCERT/CC [58], 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 IV-D), 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 IV-F). The Brave team has already merged the revised code. A fixed version of Brave will be published [59], [60]. They are also considering a UX/UI update regarding the implementation where the private browsing mode permission state is not cleared when the permission-clearing mechanism is used (Section IV-E), so that users are aware of this implementation correctly [61]. 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 [39].

C. 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 [62]. 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 supply options for users to select whether to remember 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 [63]. 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 VI-A2; 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 Appendix XI-G3.

IX. 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 [64]. 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 [65]. The DNS cache 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 [66]. 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 [67]. The analysis found that browser features such as PDF rendering libraries and pre-rendering functionality leak third-party cookies. Luo et al. developed Hindsight as a framework for identifying UI vulnerabilities in mobile browsers [68]. 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 [69]. 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 [10]. 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 [11]. 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 [3]. 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 [7]. 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 [8].

X. 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. The PERMIUM framework is expected to be useful in the standardization process and in sharing best practices among browser vendors, as it allows the systematic investigation of various browser implementations of permission mechanisms that exhibit complex behavior.

ACKNOWLEDGMENT

The authors would like to thank the members of JPCERT/CC [58] for their cooperation in our responsible disclosure.

REFERENCES

[1] Google. Permissions on Android — Android Developers. https://developer.android.com/guide/topics/permissions/overview, December 2021.

[2] Apple. Accessing User Data - App Architecture - iOS - Human Interface Guidelines - Apple Developer. https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/accessing-user-data/, October 2021.

[3] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this app safe? a large scale study on application permissions and risk signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, page 311–320, New York, NY, USA, 2012. Association for Computing Machinery.

[4] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, page 1354–1365, New York, NY, USA, 2014. Association for Computing Machinery.

[5] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 627–638, New York, NY, USA, 2011. Association for Computing Machinery.

[6] Lena Reinfelder, Andrea Schankin, Sophie Russ, and Zinaida Benenson. An inquiry into perception and usage of smartphone permission models. In Steven Furnell, Haralambos Mouratidis, and Günther Pernul, editors, *Trust, Privacy and Security in Digital Business*, pages 9–22, Cham, 2018. Springer International Publishing.

[7] Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. Exploring decision making with Android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 195–210, Santa Clara, CA, July 2017. USENIX Association.

[8] Weicheng Cao, Chunqiu Xia, Sai Teja Peddinti, David Lie, Nina Taft, and Lisa M. Austin. A large scale study of user behavior, expectations and engagement with android permissions. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 803–820. USENIX Association, August 2021.

[9] Takuya Watanabe, Mitsuaki Akiyama, Tetsuya Sakai, and Tatsuya Mori. Understanding the inconsistencies between text descriptions and the use of privacy-sensitive resources of mobile apps. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 241–255, Ottawa, July 2015. USENIX Association.

[10] Güliz Seray Tuncay, Jingyu Qian, and Carl A. Gunter. See no evil: Phishing for permissions with false transparency. In *USENIX Security Symposium*, 2020.

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

[12] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun (Aerin) Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 27–41, Denver, CO, June 2016. USENIX Association.

[13] Yuan Zhang, Min Yang, Guofei Gu, and Hao Chen. Rethinking permission enforcement mechanism on mobile systems. *IEEE Transactions on Information Forensics and Security*, 11(10):2227–2240, 2016.

[14] Yiting Qu, Suguo Du, Shaofeng Li, Yan Meng, Le Zhang, and Haojin Zhu. Automatic permission optimization framework for privacy enhancement of mobile applications. *IEEE Internet of Things Journal*, 8(9):7394–7406, 2021.

[15] Xing Gao, Dachuan Liu, Haining Wang, and Kun Sun. Pmdroid: Permission supervision for android advertising. In *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*, pages 120–129, 2015.

[16] Mozilla. Permissions API - Web API — MDN. https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API, September 2021.

[17] W3C. Requesting Permissions. https://wicg.github.io/permissions-request/, May 2022.

[18] StatCounter. Browser Market Share Worldwide — Statcounter Global Stats. https://gs.statcounter.com/browser-market-share/, March 2022.

[19] W3C. Geolocation API. https://www.w3.org/TR/geolocation/, November 2021.

[20] W3C. Media Capture and Streams. https://www.w3.org/TR/mediacapture-streams/#dom-mediadevices-getusermedia, December 2021.

[21] WHATWG. Html Standard. https://html.spec.whatwg.org/multipage/origin.html, May 2022.

[22] Mozilla. Origin - MDN Web Docs Glossary: Definitions of Web-related terms — MDN. https://developer.mozilla.org/en-US/docs/Glossary/Origin, October 2021.

[23] Mozilla. Same-origin policy - Web security — MDN. https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy, April 2022.

[24] W3C. Private Mode Browsing. https://w3ctag.github.io/private-mode/, November 2018.

[25] W3C. W3C TAG Observations on Private Browsing Modes. https://www.w3.org/2001/tag/doc/private-browsing-modes/, July 2019.

[26] Google. How private browsing works in Chrome. https://support.google.com/chrome/answer/7440301, April 2022.

[27] Microsoft. Browse InPrivate in Microsoft Edge. https://support.microsoft.com/en-us/microsoft-edge/browse-inprivate-in-microsoft-edge-cd2c9a48-0bc4-b98e-\5e46-ac40c84e27e2, April 2022.

[28] Software Freedom Conservancy. Selenium overview — Selenium. https://www.selenium.dev/documentation/overview/, December 2021.

[29] Google Developers. Puppeteer — Tools for Web Developers — Google Developers. https://developers.google.com/web/tools/puppeteer, February 2021.

[30] Microsoft. Fast and reliable end-to-end testing for modern web apps — Playwright. https://playwright.dev/, May 2022.

[31] Apple. Webdriver is Coming to Safari in iOS 13 — Webkit. https://webkit.org/blog/9395/webdriver-is-coming-to-safari-in-ios-13/, July 2019.

[32] Software Freedom Conservancy. Install browser drivers — Selenium. https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/, August 2022.

[33] Mozilla. Notification - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/Notification, April 2022.

[34] Mozilla. Incognito browser: What it really means. https://www.mozilla.org/en-US/firefox/browsers/incognito-browser/, April 2022.

[35] Brave Helo Center. What is a Private Window? https://support.brave.com/hc/en-us/articles/360017840332-What-is-a-Private-Window-, April 2022.

[36] Apple. Use Private Browsing in Safari on Mac. https://support.apple.com/guide/safari/browse-privately-ibrw1069/mac, April 2022.

[37] Mozilla. Manage local site storage settings — Firefox Help. https://support.mozilla.org/en-US/kb/storage, September 2022.

[38] Mozilla. Delete browsing, search and download history on Firefox — Firefox Help. https://support.mozilla.org/en-US/kb/delete-browsing-search-download-history-firefox, September 2022.

[39] Kazuki Nomoto. Permium. https://permium.seclab.jp/, October 2022.

[40] Apple. Full Third-Party Cookie Blocking and More. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more/, March 2020.

[41] Google. Chromium Blog: Building a more private web: A path towards making third party cookies obsolete. https://blog.chromium.org/2020/01/building-more-private-web-path-towards.html, January 2020.

[42] Google. Permission Delegation - Chrome Platform Status. https://chromestatus.com/feature/5670617353289728, December 2021.

[43] Mozilla. GlobalEventHandlers.onclick - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers/onclick, January 2022.

[44] Mozilla. Document: visibilitychange event - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/Document/visibilitychange_event, March 2022.

[45] Mozilla. The WebSocket API (WebSockets) - Web APIs — MDN. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, May 2022.

[46] Google. WebRTC. https://webrtc.org/, May 2019.

[47] Lancers,inc. Lancers. https://www.lancers.jp/, October 2022.

[48] W3C. Permissions. https://www.w3.org/TR/permissions/, December 2021.

[49] World Wide Web Consortium (W3C). World Wide Web Consortium (W3C). https://www.w3.org/, May 2022.

[50] WHATWG. Web Hypertext Application Technology Working Group (WHATWG). https://whatwg.org/, May 2022.

[51] Mozilla. How to manage your camera and microphone permissions with Firefox — Firefox Help. https://support.mozilla.org/en-US/kb/how-manage-your-camera-and-microphone-permissions, May 2022.

[52] W3C. Permissions Policy. https://www.w3.org/TR/permissions-policy-1/, July 2020.

[53] Google. Change site permissions - Computer - Google Chrome Help. https://support.google.com/chrome/answer/114662, May 2022.

[54] Brave. How do I change site permissions? - Brave Help Center. https://support.brave.com/hc/en-us/articles/360018205431-How-do-I-change-site-permissions-, May 2022.

[55] Apple. Change Websites preferences in Safari on Mac - Apple Support (CA). https://support.apple.com/en-ca/guide/safari/ibrwe2159f50/mac, May 2022.

[56] Apple. WebKit. https://webkit.org/, April 2022.

[57] Apple. App Store Review Guidelines - Apple Developer. https://developer.apple.com/app-store/review/guidelines/, October 2021.

[58] JPCERT Coordination Center. JPCERT Coordination Center. https://www.jpcert.or.jp/english/, October 2022.

[59] Brave. Don't inherit permissions in private windows · Issue #24720 · brave/brave-browser. https://github.com/brave/brave-browser/issues/24720, August 2022.

[60] Brave. Don't inherit privacy-sensitive content settings in incognito. by goodov · Pull request #14765 · brave/brave-core. https://github.com/brave/brave-core/pull/14765, August 2022.

[61] Brave. Inform users that they need to close private windows to clear data in them · Issue #25046 · brave/brave-browser. https://github.com/brave/brave-browser/issues/25046, August 2022.

[62] Igor Bilogrevic, Balazs Engedy, Judson L. Porter III, Nina Taft, Kamila Hasanbega, Andrew Paseltiner, Hwi Kyoung Lee, Edward Jung, Meggyn Watkins, PJ McLachlan, and Jason James. "shhh...be quiet!" reducing the unwanted interruptions of notification permission prompts on chrome. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 769–784. USENIX Association, August 2021.

[63] Chromium Platform Status. Status in Chromium. https://chromestatus.com/feature/6443143280984064, May 2022.

[64] Konstantinos Solomos, John Kristoff, Chris Kanich, and Jason Polakis. Tales of favicons and caches: Persistent tracking in modern browsers. In *Network and Distributed System Security Symposium*, 2021.

TABLE XII. 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)

[65] Amit Klein and Benny Pinkas. Dns cache-based user tracking. In *Network and Distributed System Security Symposium*, 2019.

[66] Martin Koop, Erik Tews, and Stefan Katzenbeisser. In-depth evaluation of redirect tracking and link usage. *Proceedings on Privacy Enhancing Technologies*, 2020:394 – 413, 2020.

[67] Gertjan Franken, Tom Van Goethem, and Wouter Joosen. Who left open the cookie jar? a comprehensive evaluation of Third-Party cookie policies. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 151–168, Baltimore, MD, August 2018. USENIX Association.

[68] Meng Luo, Oleksii Starov, Nima Honarmand, and Nick Nikiforakis. Hindsight: Understanding the evolution of UI vulnerabilities in mobile browsers. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 149–162, New York, NY, USA, 2017. Association for Computing Machinery.

[69] Wu Yuanyi, Dongyu Meng, and Hao Chen. Evaluating private modes in desktop and mobile browsers and their resistance to fingerprinting. pages 1–9, 10 2017.

[70] julioverne. julioverne/screendump. https://github.com/julioverne/screendump, November 2021.

[71] Pallets. Welcome to Flask; Flask Documentation (2.1.x). https://flask.palletsprojects.com/en/2.1.x/, May 2022.

[72] Bo Peng, Hongxing Fan, Wei Wang, Jing Dong, Yuezun Li, Siwei Lyu, Qi Li, Zhenan Sun, Han Chen, Baoying Chen, Yanjie Hu, Shenghai Luo, Junrui Huang, Yutong Yao, Boyuan Liu, Hefei Ling, Guosheng Zhang, Zhiliang Xu, Changtao Miao, Changlei Lu, Shan He, Xiaoyan Wu, and Wanyi Zhuang. Dfgc 2021: A deepfake game competition, 2021.

[73] Yuezun Li, Xin Yang, Pu Sun, Honggang Qi, and Siwei Lyu. Celeb-df: A large-scale challenging dataset for deepfake forensics. In *IEEE Conference on Computer Vision and Patten Recognition (CVPR)*, 2020.

[74] Matt Gaunt. Permissions API for the Web. https://developer.chrome.com/blog/permissions-api-for-the-web/, March 2019.

[75] Google. Temporarily stop permission requests after 3 dismissals - Chrome Platform Status. https://chromestatus.com/feature/6443143280984064, April 2022.

XI. APPENDIX

A. 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 XII summarizes the details of each client device and OS version. To establish remote controlling on iOS, screendump [70] 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 XIII lists the web browser versions. The entire framework was implemented in Python and the Flask framework [71].

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.

TABLE XIII. 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

At this time, PERMIUM determines whether the permission status is granted, denied, or unset by observing the property 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 [39]. 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 [72], [73], a widely used benchmark dataset for Deepfake.

B. Breakdown of the Identified Implementation Inconsistencies

We counted the implementation inconsistencies identified in Section IV for each of the 22 browsers. Table XIV presents the breakdown of the implementation inconsistencies identified for each browser. We can see that each browser has, on average, 8.5 implementation inconsistencies.

C. 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 XV. 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.

TABLE XV. 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.

D. 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 access the GPS [16]. The Permissions API is currently being standardized as a mechanism to unify these fragmented APIs [48]. Since 2022, most websites request permission using legacy APIs, not the Permissions API, because no browser enables 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 [16]. 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 [74], the risk of using the feature as a browser fingerprinting technique is also being discussed [48].

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 XVI. 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.

E. Browsers that Support Packing Multiple Permissions

Using the PERMIUM framework, we investigated browsers that support packing multiple permissions. Table XVII 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.

TABLE XIV. 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 XVI. 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,
○ : All permission states are not 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

TABLE XVII. SUPPORT OF 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	●	●	●	●	●

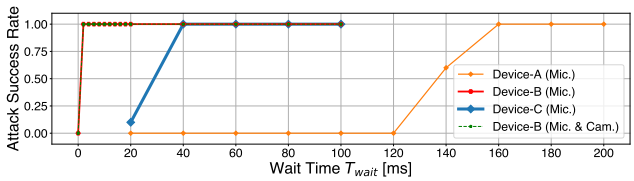

Fig. 4. Relationship between T_{wait} and attack success rate

F. 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 multiple permissions. In the normal attack, Microphone permission was used, and in the packing multiple 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 4 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 multiple 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 have an effect on processing time, even though two permissions are requested at the same time.

G. Further Evaluation and Discussion of the Permission-based User Tracking Attack

1) 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. That is, 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 the "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. Figures 5 present the results. We can see that the use of sorted index optimizes the time required for the attack, regardless of the user space to be attacked.

2) 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

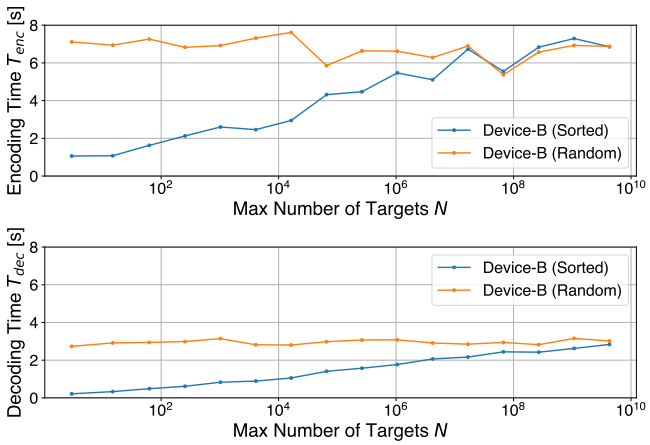

Fig. 5. N vs. required time. Top: encoding and Bottom: decoding.

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 V, 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. These observations conclude that the permission-based user tracking attack is highly covert.

3) 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 XVIII 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 [75]. 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 XIX. 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

TABLE XX. 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

TABLE XVIII. 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.

H. Demography of Participants in the Online User Study

Table XIX and Table XX show the demographics of the participants in the online user study experiment conducted in Section V and Section VII-B2, respectively.