Journal of Information Processing Vol.32 801–816 (Sep. 2024)

[DOI: 10.2197/ipsjjip.32.801]

Regular Paper

Understanding the Breakdown of Same-origin Policies in Web Services That Rehost Websites

Takuya Watanabe[1,a)] Eitaro Shioji[1] Mitsuaki Akiyama[1] Tatsuya Mori[2,3,4,b)]

Received: December 5, 2023, Accepted: March 5, 2024

Abstract: Intermediary web services such as web proxies, translators, and archives have become pervasive as a means to enhance the *openness* of the web. These services aim to remove the intrinsic obstacles to web access; i.e., access blocking, language barriers, and missing websites. In this study, we refer to these services as *web rehosting* services and make the first exploration of their security flaws. Web rehosting services use a single domain name to rehost several websites that have distinct domain names; this characteristic makes web rehosting services intrinsically vulnerable to violating the same origin policy if not operated carefully. Based on the intrinsic vulnerability of web rehosting services, we demonstrate that an attacker can perform five different types of attacks that target users of web rehosting services: persistent man-in-the-middle attack, abusing privileges to access various resources, stealing credentials, stealing browser history, and session hijacking/injection. Our extensive analysis of 21 web rehosting services, which have more than 200 million accesses per day, revealed that these attacks are feasible. In response to this observation, we provide effective countermeasures against each type of attack. Through our efforts, several major web services have successfully completed improvements to make their architecture more secure.

Keywords: Web, browser, same-origin policy, service worker

1. Introduction

While the architecture of the web follows the fundamental Internet design framework — *the End-to-End principle* [2]— several web services that deviate from the principle have been developed to enhance the openness of the web. Typical examples of such services include *web proxy*, *web translator*, and *web archive*, which all aim to enhance the openness of the web in the following ways: Web proxies enable a user to access websites that are blocked by nations and institutes. Web translators help a user to understand a web document written in a foreign language that is difficult for the user to read. Web archives enable a user to access a version of previously published web content, which is not available presently due to several reasons such as expiration, maintenance, or blocking. All these technologies work on top of a middle box, which we call "*web rehosting*" throughout this paper.

With the increase in the number of web users, web rehosting services have become pervasive. For instance, according to the web traffic statistics provided by SimilarWeb [3], one of the most popular web proxy services, ProxySite [4] had more than 20 million accesses per day in September 2019. The world's top search engine companies such as Google, Microsoft, Baidu, and Yandex offer web translator services. Google website translator alone supports over 100 languages [3], [5] and serves over 80 million accesses per day from all over the world. Through an analysis of anonymized access logs collected from Wayback Machine servers in February 2012, AlNoamany et al. [6] reported that the service had about 82 million accesses per day. All these web rehosting services have become popular because they enhance the openness of the web. Furthermore, they are easy to use, as a user can simply access such services by using a normal browser and inputting a URL of interest; unlike other alternative solutions such as an HTTP proxy [*1], VPN, or Tor, they do not require setting changes and the installation of special applications.

As web rehosting services offer enhanced web access to various websites, an attacker has an incentive to exploit them because users may input privacy-sensitive information while accessing *rehosted websites* via a web rehosting service. For instance, a user who uses a web proxy to access a webmail service needs to input the credential of the webmail account via the web proxy. Similarly, content rewriting on web rehosting services is useful for inciting and intimidating users. An attacker may want to abuse the web translator to create fake news by completely rewriting an original news article while giving the fake article the appearance of the translated version of the original article shown by the web translator. Given this background, this work addresses the generic security flaws of web rehosting services. To the best of our knowledge, this is the first study to focus comprehensively on the services with the property of rehosting websites and iden-

[1] NTT Social Informatics Laboratories, Musashino, Tokyo 180–8585, Japan
[2] Waseda University, Shinjuku, Tokyo 169–8555, Japan
[3] NICT, Koganei, Tokyo 184–8795, Japan
[4] RIKEN AIP, Chuo, Tokyo 103–0027, Japan
[a)] watanabe@nsl.cs.waseda.ac.jp
[b)] mori@nsl.cs.waseda.ac.jp

This paper is an extended version of the paper presented at NDSS'20 [1].

[*1] In this paper, we call proxies with web-based interfaces (i.e., a type of web rehosting service) web proxies and distinguish them from HTTP proxies.

tify the security problems in common. Based on the observations found through an analysis of various rehosting services, we propose a Proof-of-concept (PoC) attack model and evaluate its feasibility.

The key idea of the attacks is to leverage the fact that a single domain name provided by a web rehosting service is used to access multiple rehosted websites; this "*melting pot of origins*" situation allows an attacker to bypass the filtration of the same origin policy (SOP). A malicious website rehosted by an attacker to the web rehosting has the same origin as the rehosted websites. The malicious site enables the attacker to tamper, steal, and/or take control of the various resources of a victim's browser when the victim accesses rehosted websites. Using this vulnerability, we demonstrate that an attacker can perform the following five attacks, which exploit different resources: (1) persistent man-in-the-middle (MITM) attack, (2) abusing privileges to access resources, (3) stealing credentials, (4) stealing browser history, and (5) session hijacking and injection.

Our attacks exploit both traditional and modern browser features: service workers [7] and application cache (AppCache) [8] for (1), browser permissions for (2), password manager for (3), localStorage [9] for (4), and cookies for (4) and (5). We note that of the five attacks, the persistent MITM attack, which is enabled by abusing the service worker or AppCache, is quite powerful in the sense that it can be executed even under a secured channel — HTTPS. In this attack, an attacker, who should be outside the network path in a realistic attack scenario, does not need to intercept the HTTPS channel. Once a victim accesses a rehosted malicious website, an attacker can manipulate any requests/responses made by the victim and rehosted websites. The attack is sustained until the browser data is manually cleaned up; hence, the attack is persistent.

To verify the feasibility of the attacks, we collected 21 popular web rehosting services and tested whether or not these web rehosting services are vulnerable to the attacks. We found that 18 services were vulnerable to at least one of the attacks. The persistent MITM attack was effective on 13 web rehosting services, including prominent services such as Google Translate, Wayback Machine (web archive), and Hide My Ass (web proxy). We also revealed that around 40% of the top-10 K websites including sensitive categories (e.g., porn, dating, and piracy) had a unique fingerprintable record in their cookie and localStorage, implying that the browser history theft, which leverages fingerprints, will succeed when these websites are accessed by a victim via a web rehosting service.

The contributions of our study are summarized as follows:

- This is the first study to shed light on the security flaws of web rehosting services, which offer enhanced accessibility to various web services.
- We present five attacks derived from the vulnerability intrinsic to web rehosting services.
- We demonstrate the feasibility of the attack through the extensive analysis of 21 web rehosting services.
- We provide effective countermeasures against the attacks.
- Based on our report, countermeasures have been implemented for actual web rehosting services.

The remainder of the paper is organized as follows: Section 2 presents a background of the web technologies targeted in this study. In Section 3, we present the threat model and the descriptions of derived attacks under the threat model. Section 4 demonstrates the feasibility of attacks through experiments using 21 of the popular web rehosting services. Section 5 discusses the coverage of our study, human factors, and ethical considerations including our disclosure process and responses from service providers. Section 6 presents effective countermeasures against the threats. In Section 7, we review related works in comparison with ours. We conclude our study in Section 8.

2. Background

In this section, we review the background of web technologies that are targeted in the proposed attack.

2.1 Web Rehosting in the Wild

Figure 1 presents a high-level overview of web rehosting services. Upon receiving a request from a user to access a website, the web rehosting service fetches the content of the website on behalf of the user, transforms the content following the context of the service, and presents the transformed content to the user. Most web rehosting services with this characteristic can be classified into three categories: web proxy, web translator, and web archive (several exceptions are discussed in Section 5.1). While web rehosting services are used for various purposes, their usage is common — a user simply accesses a web rehosting service and inputs a URL of interest. The simplicity of the web rehosting services has made it possible for many people to use it easily. Besides, the services are available to people who are non-tech-savvy, people on a network with limited ports, and even people using a shared PC at libraries or schools. These advantages well explain the reasons why other tools such as Tor or VPNs, which require a change of setting and/or installation of a special application, may not be adopted as an alternative solution under certain circumstances.

Table 1 summarizes the 21 web rehosting services we studied in this work. The list contains 11 web proxy services, 7 web translator services, and 3 web archive services. The estimated number of daily accesses is over 200 million in total according to SimilarWeb [3], implying that these services are widely used. Of the 21 services, only three services have not adopted the HTTPS scheme as of September 2019. Note that we anonymized two services at the request of their providers: Service-α denotes web-proxy ser-

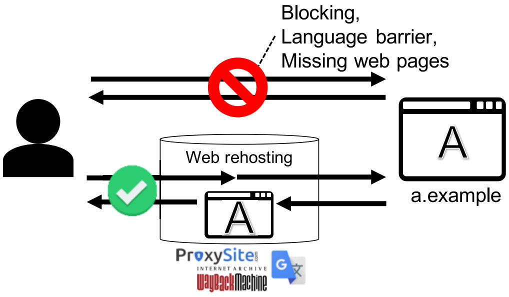


Fig. 1 Overview of web rehosting services.

Table 1 A list of web rehosting services examined in this study. Service-α and Service-β have been anonymized at request of their providers.

Category	Rehosting Service	Scheme	#Accesses / Day [3]
Proxy	ProxySite [4]	HTTPS	20.14 M
	Hide My Ass! [10]	HTTPS	4.64 M
	Hide me [11]	HTTPS	4.49 M
	Sitenable Web Proxy [12]	HTTPS	2.50 M
	FilterBypass [13]	HTTPS	1.26 M
	ProxFree [14]	HTTPS	1.18 M
	toolur [15]	HTTPS	0.92 M
	hidester [16]	HTTPS	0.76 M
	GenMirror [17]	HTTPS	0.41 M
	UnblockVideos [18]	HTTPS	0.38 M
	Service-α	HTTP/S	–
Translator[*2]	Google Translate [19]	HTTPS	80.45 M
	Bing Translator [20]	HTTPS	2.62 M
	Weblio [21]	HTTPS	2.30 M
	PROMT Online [22]	HTTP	0.58 M
	Service-β	HTTPS	–
	Yandex.Translate [23]	HTTPS	0.18 M
	Baidu Translate [24]	HTTP	N/A
Archive	Wayback Machine [25]	HTTPS	45.42 M
	Google Cache [26]	HTTP/S	41.50 M
	FreezePage [27]	HTTP	N/A

vices using an open-source software as the backend system. The software has been redistributed by a third party and has remained unmaintained for several years. Service-β represents a web translator which is popular in a certain country.

2.2 Advanced Web Features

In this section, we describe the advanced web features that are targeted by attacks that compromise web rehosting services.

Service Worker. A service worker [7] is a modern web feature for both desktop and mobile platforms. It is supported by major browsers such as Chrome, Edge, Safari, Firefox, and Opera. It is an event-driven web worker written in JavaScript. It works independently with the main browser thread and provides rich features such as background data synchronization and push notification handling. A notable feature of a service worker is that it can proxy all the requests and responses between a web client and servers, and modify the content. Thus, it offers quite powerful capabilities.

A service worker inherently implements strong security constraints to prevent its powerful capabilities from being exploited. First, it only works on web services that are operated in a secure context, i.e., HTTPS or local. Second, the service worker, which ensures compliance with the SOP, is associated with the origin and URL path and only operates on a URL whose path contains the service worker script or the lower. Accordingly, if a web server environment separates operation areas by their subdomain names or URL paths like generic hosting services do, one cannot register a service worker that targets a website that is operated under the same web server but has a different subdomain name or URL path. Finally, a browser requires that the MIME type of the service worker script be specific to JavaScript, `text/javascript`, `application/javascript`, and `application/x-javascript`; otherwise, it does not register the service worker.

Application Cache. The HTML5 standard provides an application caching mechanism (AppCache [8]) that allows a web application to run offline. This feature has the following three advantages: (1) *Offline browsing* — users can browse websites even when they are offline. (2) *Speed* — as the cached resources reside in storage of browsers, they are loaded quickly. (3) *Reduced workload* — a browser downloads only changed resources from the server; hence, the cache mechanism reduces the workload of the network and server. To achieve (1), the offline browsing AppCache provides a cached alternative resource instead of a fallback page displayed due to network or server errors.

Since the HTML 5.1 standard was released in November 2016, AppCache has been deprecated [8]. It is recommended that developers use the service worker API as an alternative solution. However, as of September 2019, AppCache still works with the latest versions of browsers, including Chrome, Firefox, Opera, IE, and Safari. The constraints of AppCache are similar to those for a service worker. The chief difference is that AppCache works independently of paths for pages on the same origin.

Browser Permissions. In HTML5, web browsers support accessing various resources such as geolocation, camera, microphone, and notifications. Access to these resources requires permission through user interaction, and permission is granted to each resource for the realms with the same origin [28]. As with the features shown above, this access control assumes that a different website runs on a different domain name. However, the access granted via a web rehosting service violates this assumption.

Browser-based Password Managers. Presently, major browsers such as Chrome, Firefox, Opera, IE, and Safari all come with a built-in password manager. In general, a password manager works in the following manner. First, a user visits a website and enters a password to be authenticated to a service running on the website. Then, the browser will ask the user whether or not to save the password. If the user permits saving of the password, the browser will store it on the user's device or associated database running on cloud. When you revisit the website later, the browser autofills the stored password. As the password managers attempt to identify the password associated with a website by checking the domain name of the website, web rehosting services, which use a single domain name to host multiple websites, could violate the fundamental assumption made by the password managers, i.e., different websites should have different domain names.

2.3 Pitfalls of Cookies

Finally, we present several pitfalls of cookies that lead to their exploitation by attacks.

Access from JavaScript. Cookies written in HTTP headers have an `HttpOnly` flag. If this flag is set to true, the cookie cannot be accessed from the JavaScript. However, for most cookies handled by web rehosting services, the flag is set to false, implying that cookies set when a victim visited a malicious website can be accessed by the JavaScript running on the malicious website. Meanwhile, cookies written by a JavaScript can be accessed by another JavaScript.

[*2] These web translator services except for Weblio and PROMT Online have two separated domain names for website translation and text translation. We only showed the data for the website translation.

Session Cookie. In general, by leaving the expiration date unset, a cookie is treated as a session cookie [29]. Technically, a session cookie should disappear when a browser ends the session. In practice, however, configuring the "Continue where you left off" setting in a browser keeps the session cookie alive. As we will show in Section 4.3, some browsers set this as the default during installation. Besides, in mobile platforms, the session cookie does not disappear automatically even when a browser ends a session. Thus, deploying session cookies is not a countermeasure against attacks that exploit cookies.

Cookie Bomb. Web servers deny requests with request headers that are too large, e.g., the maximum size in Apache is 8,190 bytes by default. If a browser has many and large cookies for a website, it will always return a server error, as the cookies are attached to the request header. Homakov [30] named the DoS attack resulting from this behavior as *Cookie Bomb*. We will demonstrate in section 3.2.1 that by combining it with the ability of AppCache to rewrite fallback pages, a more advanced attack can be established against users of web rehosting services.

3. Threat Model and Attacks

In this section, we first present a threat model based on origin unification, which is a property common to web rehosting. We then propose five attacks targeting web rehosting users under our threat model.

3.1 Threat Model based on Origin Unification

The mechanism used by web rehosting to handle web pages of different domain names is shown in **Fig. 2**. Each *origin* of a web page is converted into the same origin as the web rehosting service. The origin contains a schema, domain, and port. The SOP, the major principle of web security, restricts a document or script loaded from one origin to interact with a resource from another origin [31].

Because of the SOP, a script placed on the page with `evil.example` was not able to access the resource stored by pages with `a.example` and `b.example`. Given pages with different domain names, however, web rehosting rehosts the pages under a single domain name; hence, this mixture of origins makes the SOP ineffective in rehosted pages. In general, services that concern SOP violation adopt the sandbox mechanism to isolate the domain name of the service itself (`web-rehosting.example`) from the sandboxed domain name for untrustworthy rehosted pages (`rehosted.example`). We argue that the SOP problem still remains; among rehosted pages in the sandboxed domain name (`rehosted.example`) a script of a certain page still affects resources of other pages, while the browser resource in `web-rehosting.example` is isolated from that of `rehosted.example` owing to the SOP.

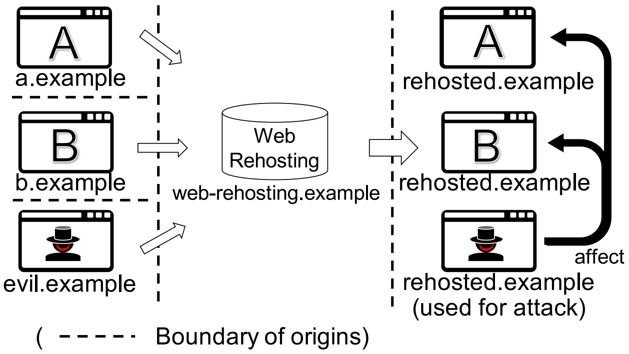


Fig. 2 Origin unification that occurs when web pages are rehosted.

We assume that an attacker first rehosts the prepared page with the malicious script on the web rehosting service that has the aforementioned problem. The attacker inputs `evil.example` to the web rehosting service; then, a URL like `https://rehosted.example/rehost?url=https://evil.example` is generated. Next, the attacker induces the victim user to access that rehosted malicious page through the conventional web attack scenarios (e.g., drive-by download, phishing, and cross-site request forgery). The attacker can efficiently attract victims by using malvertising, spam email, and social media posts with the link to the rehosted malicious page. Note that some web rehosting services, which prohibit hotlinking (i.e., direct link to the rehosted page) by validating referrers or HTTP sessions, make the attack difficult [*3]. Finally, the attack is triggered when the victim user accesses the generated URL.

To make the attack more efficient, attackers can use a landing page embedding multiple `iframe` tags pointing to malicious pages rehosted by various web rehosting services. With a single visit to that page, the victim user becomes susceptible to attacks targeting multiple web rehostings. The landing page does not need to be rehosted. There are several options for impersonating a domain name: wrapping by a shortened URL or redirect service and injection to legitimate sites by XSS and website falsification. These techniques are not unique to our attacks, but are generally used for deceiving web users.

In this study, we provide novel attacks targeting web rehosting services that leverage both traditional resources such as cookies and the recent powerful resources of HTML5 and progressive web apps (PWA), which caused unexpected usage of resources when web rehosting architecture was designed.

3.2 Attacks against Web Rehosting

Based on our threat model, we propose five attacks widely applicable to web rehostings. We first summarize these attacks in **Table 2**. Our attacks affect both *past* and *future* activities of the victim visit to the rehosted malicious page. We classified these attacks into two types: exploiting resources to know what web rehosting users have read/written before now (i.e., before visit), and *parasitizing* resources to monitor and tamper with the victim's browser from this time (i.e., after visit).

We explain each of the five attacks as follows.

3.2.1 Persistent MITM

This is a new kind of MITM attack that works persistently after being attacked by an *off-path* attacker by exploiting service workers or AppCache. We summarize the differences between a service worker and AppCache in **Table 3**.

We found that an attacker can register a malicious service worker on the origin provided by web rehosting services. List-

[*3] If the victim user accesses the rehosted malicious page via another rehosted page, the attack still works.

Table 2 Attacks against web rehosting services. Impacted Timing indicates whether an attack is carried out before or after the target user's visit to a malicious site.

Attacks	Exploited Resources	Impacted Timing	Assumption of User Behavior
Persistent MITM	Service Worker, AppCache	after visit	(none)
Privilege Abuse	Camera, Microphone, Location, Notification, etc.	both	gave permission at any rehosted website
Credential Theft	Password Manager	both	saved password at any rehosted website
History Theft	Cookie (written by JavaScript), localStorage	before visit	(none)
Session Hijacking and Injection	Cookie (written by HTTP header)	both	is logged in (hijacking)

Table 3 Comparison between service workers and AppCache

Resource	Service Worker	AppCache

MIME-Type	`text/javascript`
	`application/javascript`
	`application/x-javascript`
		`text/cache-manifest`

Origin scope	- Same origin	- Same origin
Path scope	- Same and lower directory of SW script	- Any path
Page scope	- Any page	- Fallback page - Any page (with Cookie Bomb)

Listing 1 Code to register service worker

```
<script>
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js')
        .then(function (registration) {
        }).catch(function (error) {
          // registration failed
        });
  };
</script>
```

Listing 2 How to assign the rehosted service worker. The origin of this HTML and `sw.js` is `rehosted.example`.

```
    navigator.serviceWorker.register('https://rehosted.example/rehost?url=https://evil.example/sw.js')
```

ing 1 shows an example of a general HTML to register a service worker. In this case, `sw.js` under the root path (/) is registered corresponding to the origin of this HTML (same as the origin of `sw.js`). An attacker can implement the functionality into `sw.js`, which reads and rewrites web requests and responses. Listing 2 shows how to register a malicious service worker on the origin provided by web rehosting (`rehosted.example`). We note that a service worker on a certain origin, i.e., `rehosted.example`, cannot be registered from a page on another origin, i.e., `evil.example`. When the target visits the rehosted malicious page as we presented in our threat model, the malicious service worker script is registered and it begins the persistent MITM attack because the origin of the service worker script is the same as that of the page the victim visited.

An overview of the attack abusing a service worker is shown in **Fig. 3**. An attacker rehosts the malicious page that contains the above HTML to register a malicious service worker on a victim user. Once a victim user visits the rehosted malicious page, the malicious service worker compromises the victim's browser and permanently compromises all web communications via that web rehosting. Therefore, the victim's sensitive information is leaked to the attacker as long as the victim uses that web rehosting. The service worker can perform diverse attack scenarios, such as modify the nuance of new articles, replace movies, inject ads, display a phishing page, and replace downloaded files with malware. While this persistent MITM attack gives attackers almost the same benefits as the traditional MITM attack, the persistent MITM attack is more powerful for the following reasons: there is no need to directly intercept communications on the network path, permanent eavesdropping once the service worker is registered occurs, and HTTPS-enabled pages are affected. Unfortunately, current browsers do not have an easily understandable user interface to check registered service workers. To determine whether the service worker is registered, a user should open the developer console and carefully inspect the setting of the service worker. As we mentioned in Section 2.2, a service worker has restrictions that ensure secure execution. We present, in Section 3.3, a method to circumvent the restrictions by leveraging the rehosting rules of web rehosting.

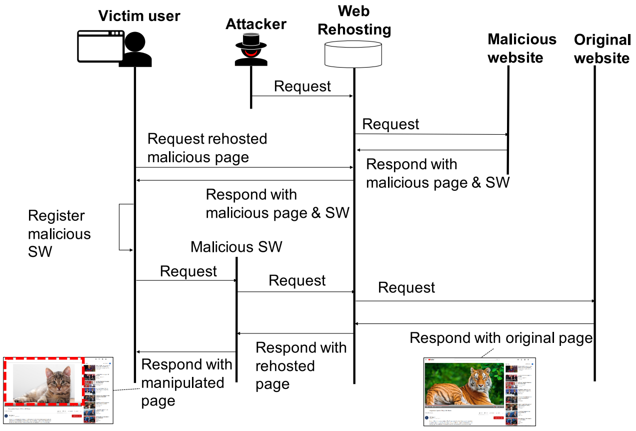

Victim user
Attacker
Web Rehosting
Malicious website
Original website
Request
Request
Request rehosted malicious page
Respond with malicious page & SW
Respond with malicious page & SW
Register malicious SW
Malicious SW
Request
Request
Request
Respond with original page
Respond with rehosted page
Respond with manipulated page

Fig. 3 Overview of the attack abusing service worker.

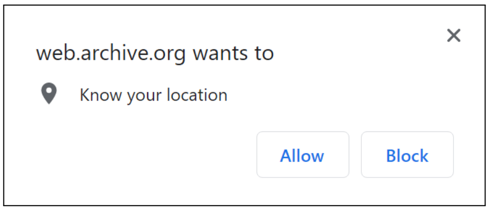
Listing 3 AppCache Manifest File to replace fallback pages

```
CACHE MANIFEST

FALLBACK:
* /rehost?url=https://evil.example/replace.html
```

AppCache also enables the attacker to perform a persistent MITM attack without a service worker. The procedure of the attack using AppCache is similar to that using a service worker. The attacker first rehosts a malicious *manifest* file and then rehosts an HTML file including the URL of the rehosted manifest file. At this time, the attacker writes the *fallback* rule in the manifest file, which lists two URIs: the first is the page to be rewritten (wildcard available), and the second is the fallback page as a Listing 3.

The attacker, in the rule, defines pages to be rewritten and the rehosted malicious page for rewriting. Both URLs must be relative and in the same origin. Owing to using a fallback rule, AppCache can only rewrite fallback pages to return an error status


web.archive.org wants to

Know your location

Allow

Block

Fig. 4 Example of location permission request on a (legitimate) rehosted page in Wayback Machine.

code (e.g., 400, 403, 404, and 500). However, by writing huge cookies to the browser of the victim on the rehosted malicious page, the attacker can force all requests from the victim to the web rehosting server to fall back (similar to the Cookie Bomb in Section 2.3). The attacker tampers with all the pages visited by the victim via the web rehosting service, even with AppCache.

As we described in Section 2.2, the scope of AppCache (i.e., the origin) is wider than that of service workers (i.e., the origin and the path), so AppCache may be effective for web rehostings for which an attack using a service worker is unsuccessful. The scope indicates the range of pages where a service worker or AppCache work.

3.2.2 Privilege Abuse

Web pages sometimes request permission to make use of the hardware resources of a web user such as a camera or GPS, even if the page is rehosted. The permission corresponds to the origin. Therefore, in the case of web rehosting, the permission is shared with all the rehosted pages. **Figure 4** shows an example of a location permission request when a user visits a legitimate website rehosted on Wayback Machine. Once the user clicks "Allow," an attacker can divert the permission by using the rehosted malicious page later. In other words, the attacker can stealthily access resources (e.g., GPS, camera, and microphone) previously permitted on other rehosted benign pages.

Moreover, the attacker can distribute Web Push notifications by combining the malicious service worker with the permission for the notification. In such a case, while the browser process is running, there is a risk that the browser will always receive the Web Push notifications from the attacker. The notifications can contain messages for phishing, harmful images, and URL links to malicious websites [32].

This attack does not affect certain web rehosting services where a rehosted page is loaded in `iframe` and its domain is different from that for the top-level browsing context. In this case, `iframe`'s sandbox mechanism automatically denies any permission requests without a user's interaction; hence, the attack always fails.

3.2.3 Credential Theft

The credentials stored in the browser's built-in password manager [*4] correspond to the origin of the page. When a browser accesses a certain page, credentials corresponding to the origin of the page are automatically input to the login form by the autofill function of the password manager. There are a lot of pages of originally different services on web rehosting. Once a user logs into Facebook or Twitter via web rehosting and saves the credentials in the browser, the password manager automatically fills the credential in the form when the user accesses the rehosted malicious page with the form for the ID and password. By using malicious JavaScript, the attacker can steal the credential that is input to the form even if it is not submitted to the page. We note that whether password manager autofills credentials depends on not only the origin but also the structure of the form (e.g., string in the action attribute) to be filled [35]. However, the attacker can construct an ideal form in rehosted malicious pages that satisfy the conditions for the browser to autofill credentials.

This attack is successful on mainly web proxy services because they should support login function for rehosted pages. The services that require users to log into the service to browse content like Facebook are often subject to state-sponsored censorship. The most popular web proxy service we examined advertises the ability of that service to circumvent censorship against Facebook. Therefore, our assumption that the user logs into the service through a web proxy is realistic. We discuss the feasibility of this attack in detail in Section 5.2.

3.2.4 History Theft

A JavaScript code using cookies and localStorage is common among modern websites. The data is separately stored in each origin, so rehosted pages consequently share the data in cookies and localStorage. An attacker can abuse such data for fingerprinting visited websites; then, a victim user's browsing history is stolen by the attacker.

The data usually consists of the `key`–`value` pair. The characters of the `key` used in cookies or localStorage are statically defined in each website, and the characters of the `value` are often dynamically assigned. Therefore, we use the `key` for fingerprinting. We empirically found that value in localStorage in a certain page is stored in a JSON format. If the JSON data has an associative array structure, the attacker can recursively parse it and extract additional `keys` for fingerprinting.

We found that the expiration time of cookies is usually set as the time generated by adding a certain period (e.g., one month) to the time of website access. For a page that uses such a cookie, an attacker can also possibly infer the accessed time of a target user from the expiration time of the cookie. The effectiveness of the fingerprinting approach will be examined in Section 4.

3.2.5 Session Hijacking and Injection

As we explain in detail later in Section 3.3, a web proxy relays the cookies in the HTTP header to transparently ensure HTTP sessions between the browser and original page. A web proxy attaches a newly issued cookie with the HTTP header from the rehosted page. When a user logs in to a certain service through a web proxy, the user's browser stores the cookie newly issued by the web proxy. The attacker can steal this cookie and hijack HTTP sessions of the original page from the rehosted malicious page using JavaScript. The cookies used in history theft described in the previous subsection are written using JavaScript and those used in this attack are written using HTTP header; however, both can be stolen using JavaScript. Even though setting the `HttpOnly` flag on the relayed cookie is effective as a countermea-

[*4] Though details are omitted owing to space limitations, we confirmed that our attack also works on third-party password managers that have an autofill function such as Lastpass [33] and DashLane [34] in the same manner.

sure against this attack, most of the web rehostings do not adopt it. Note that a session cookie does not work as a countermeasure, as mentioned in Section 2.3.

An attacker can also inject a session into the victim's browser and force the victim to log in to the account prepared by the attacker. This concept, session injection [*5], is useful for tracking the victim's future behavior. For example, Google stores the search history log of Google accounts on the user activity page. Video sites and shopping sites automatically record the history log of content viewed by users like YouTube and Amazon for each account. Thus, such a log indicates the victim's online behavior, which can be retrieved at any time by the attacker.

Session hijacking only applies to users who actively log in to a web service through web rehosting, whereas session injection applies to users who do not log into the service on their own will. We note that the victim may suspect the session injection because they will browse the service while logging into a strange account prepared by the attacker.

3.3 Rehosting Rules

Attacks described in Section 3.2 strongly depend on the *rules* adopted by web rehosting services; we call such rules *rehosting rules*, and they include the mechanism used by a web rehosting service to rewrite a URL, the file type that can be rehosted, and how the browser resources are handled. We introduce commonly adopted *rehosting rules* and how attackers abuse these rules to manipulate browser resources.

URL Rewriting. The most fundamental rule is URL rewriting. We present how each component of a URL, i.e., scheme, domain name, and path are changed to be rehosted.

In most cases, the scheme of the rehosted page is changed to the scheme provided by the web rehosting services. Exceptions include Google Cache (web archive), which uses the original scheme of the rehosted website, and Service-α (web proxy), which depends on the server's configuration owing to the OSS application. If a web rehosting service uses only an HTTP scheme, there is, of course, the risk of a MITM attack from remote attackers on the network path. Our key insight is that both the persistent MITM attack (Section 3.2.1) using a service worker and AppCache, and privilege abuse (Section 3.2.2) using resource requesting permissions are effective on only HTTPS-enabled web rehosting services because those resources need to be operated in a secure context.

We introduced the threat model (Section 3.1) on the basis of the origin unification for rehosted pages. In addition to this case, there are some web rehosting services that use several subdomains, e.g., `us.rehosted.example` and `eu.rehosted.example` for rehosted pages, which are mainly for load-balancing. This is not an obstacle for our attacks because an attacker can rehost malicious pages on each subdomain by repeating requests to rehost even in this case and make a victim access those rehosted malicious pages by using a single landing page containing `iframe` tags.

Web rehosting provides mainly two types of URL naming conventions for rehosted pages: (1) URL query convention (used by web proxy and web translator) and (2) UNIX-path like convention (used by web archive). An example of the former is `https://rehosted.example/rehost?url=evil.example` and an example of the latter is `https://rehosted.example/evil.example/`. Between them, there is only a trivial difference in URL parsing at the server-side, but there is a significant difference at the client-side. The scope of the service worker refers to the path located at the service worker script (i.e., `sw.js`) as mentioned in Section 3.2.1. In the case of URL query convention, a registered service worker affects all the rehosted pages. On the other hand, in the case of UNIX-path like convention, a registered service worker cannot affect them except for the rehosted page of `evil.example`.

In addition, some web rehosting services eliminate JavaScript code from the page; hence, the attacks cannot succeed in such services.

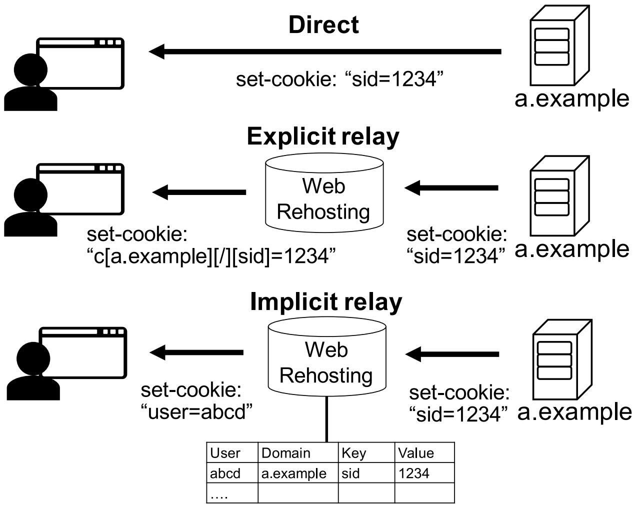
Rehostable File Type. Web proxy and web archive services can rehost any kind of content with the original MIME Type. An attacker can exploit this rule to place a malicious service worker script or an AppCache manifest file on the origin of rehosted pages. On the other hand, web translator services generally only rehost files with a MIME Type for HTML or plain text. However, we found that, for web translator, the JavaScript file is automatically rehosted when the HTTP-scheme URL of the JavaScript is set at the `src` attribute of the `script` tag in the page. The reason for this exceptional behavior is to avoid security errors caused by a mixed content problem [36], that is, the inconsistency of the scheme of the rehosted HTML page (HTTPS) and that of the rehosted resource page (HTTP). According to such exceptional behavior, the URL at the `src` attribute is converted into the URL of the rehosted JavaScript, so an attacker can use this URL as the malicious service worker script on the origin of the rehosted pages.

Handling Browser Resources. While the origin of the rehosted page will be changed to that provided by web rehosting, web rehosting does not rewrite the JavaScript code [*6]. Thus, all resources and permissions that are stored via JavaScript on the rehosted benign page are simply accessible from the rehosted malicious page.

The handling of cookies derived from the HTTP header depends on the web rehosting category. Wayback Machine disables cookie storing by adding a specific prefix to the header name such as `x-archive-orig-set-cookie`. Other web archives and web translators simply discard the `Set-Cookie` header. In contrast, the web proxy implicitly or explicitly relays cookies in the HTTP header in order to reconstruct the HTTP session between a browser and the rehosted page. The handling of the cookies among browsers, web services, and web rehosting is shown in **Fig. 5**. When a browser logs in to a web service, a cookie named `sid` with key, `1234` as the value is generated. If the browser accesses the web service (e.g., `a.example`) via a web proxy *ex-*

[*5] This is the same type of attack called *session fixation*.

[*6] Although it does not affect attack success, Wayback Machine rewrites Google analytics code on archived pages to optimize access analysis as an exceptional behavior.



User	Domain	Key	Value
abcd	a.example	sid	1234
....			

Fig. 5 HTTP cookie relaying in a web proxy.

plicitly relaying cookies, the name of the cookie is changed to `c[a.example][/][sid]`. At this time, the relayed cookie stored in the web browser is linked to the domain name of the web proxy. When the browser tries to access `a.example`, the server of the web proxy converts the name of the cookie into its original name and adds it to the request to access `a.example`. At the web proxy that *implicitly* relays cookies, the cookie between `a.example` and the server of the web proxy is mapped to a corresponding cookie between the browser and the server of the web proxy from a translation table. The table associates users using the web proxy with the sessions of rehosted website. Both methods of cookie relay transparently maintain HTTP sessions between web browsers and web servers. Our attack can directly hijack sessions or inject session IDs for web proxy services that explicitly relay cookies. For web proxy services that implicitly relay cookies, our attacks can still hijack sessions or inject the generated session IDs although the original session IDs are hidden.

4. Feasibility Analysis

In this section, we analyze the feasibility of the attacks we introduced in Section 3.2. We first examine whether the 21 popular web rehosting services are vulnerable to our attacks. We then investigate the fingerprintability of websites to assess the effectiveness of the history theft attack. Finally, we analyze the differences in the resource access behaviors of browsers.

4.1 Vulnerable Rehosting Services in the Wild

We conducted vulnerability checks on services by actually attacking them. We targeted 21 web rehosting services listed in Section 2.1. Note that neither the servers nor users were impacted, as our experiment only checked for unexpected accessibility to client-side browser resources in our own testing environment.

Surprisingly, against 18 of the investigated services, at least one of our attacks succeeded. Of those, three services denied hotlinking, limiting the feasibility of the attacks; however, the rest, including those run by world-famous providers such as Google, Bing (Microsoft), and Baidu, allow hotlinking. On the other hand, against FreezePage, GenMirror, and FilterBypass, all the attacks were infeasible as the services all force-remove JavaScript code included in webpages when rehosting them.

The three web rehosting services adopting HTTP are vulnerable to typical MITM attacks over the network path, while the remaining 16 only needed to consider the risks of attacks feasible only in a secure context. There were 13 services vulnerable to persistent MITM attack, and 12 of them are vulnerable to the service worker attack, which is the most powerful of our attacks.

For these services, the attacker can steal the victim's activities and privacy-sensitive information, rewrite a part or all of the viewed content, and replace binary files or movies; these can lead to malware infection, phishing, or even political instigation. If the web rehosting service and rehosted website are both trustworthy, e.g., reading a CNN news article through Google Translate, it is difficult for a user to notice if the content has been modified by an attacker. For web rehosting services such as UnblockVideos, it is possible to replace a movie file with an attacker-prepared one. To monetize this attack more directly, the attacker can usually insert advertising and a cryptominer [37], [38], [39].

As another interesting case study, Google Translate has a feature that translates user-uploaded local files such as PDF and Word documents. We noticed that the domain of the website showing the translation result is the same as that of the rehosting website. This means that a user, who has accessed an attacker-prepared malicious site before and has a service worker implanted, can have the translated version of their uploaded documents stolen by the attacker. Thus, if a document with classified or privacy information is uploaded for translation, which is not a rare use case, it will be stolen by a third party under this attack.

The rest of the services are not vulnerable to the attacks that use the service worker owing to the path constraints on the service workers or the inability to rehost JavaScript code. Of these, for Wayback Machine, a URL is specified UNIX-path like, but an attacker can instead use AppCache, which works regardless of the path to replace all fallback requests with a webpage prepared by an attacker. Furthermore, we confirmed that by saving 100 cookies of 200 bytes with a JavaScript code on the rehosted malicious page, all pages fall back on the Wayback Machine. All the pages viewed by the victim, therefore, can be replaced with pages prepared by the attacker, even in the Wayback Machine.

We noticed that translators provided by Google, Yandex, Bing, Baidu, and PROMT place rehosted content in iframe, which is protected by a sandbox attribute. These services were safe against privilege abuse attacks, but the rest of the services were vulnerable to them. We found that a flaw enabling the credential theft attack is present in all of the investigated web proxy services. Furthermore, a browsing history theft is feasible against all investigated services that had JavaScript enabled. For the number of fingerprintable websites, refer to Section 4.2.

For all web proxy services that adopt explicit relay of sessions described in Section 3.3, we found that it is possible to hijack the session or inject an attacker's session. Moreover, even in cases where the implicit relay is adopted, such as in Sitenable Web Proxy and ProxFree, it is still possible to steal the session used for managing web rehosting users; consequently, it is possible to hijack the session of rehosted websites that are internally associ-

Table 4 Vulnerabilities of the 21 services investigated. Vulnerable (●) and Secure (○). "hotlink" shows whether a third party who did not generate the URL of the rehosted page can access it, while "sw" stands for service worker. "*note*" indicates an exceptional vulnerability detailed in the body.

Category	Rehosting Service	Scheme	Hotlink	At least one Vulnerability	Persistent MITM		Privilege Abuse	Credential Theft	History Theft	Session Hijacking & Injection
					SW	AppCache				
Proxy	ProxySite	HTTPS	no	●	●	●	●	●	●	●
	Hide My Ass!	HTTPS	yes	●	●	●	●	●	●	○
	Hide me	HTTPS	no	●	●	●	●	●	●	●
	Sitenable Web Proxy	HTTPS	yes	●	●	●	●	●	●	●
	FilterBypass	HTTPS	no	○	○	○	○	○	○	○
	ProxFree	HTTPS	yes	●	●	●	●	●	●	●
	toolur	HTTPS	yes	●	●	●	●	●	●	●
	hidester	HTTPS	no	●	●	●	●	●	●	●
	GenMirror	HTTPS	no	○	○	○	○	○	○	○
	UnblockVideos	HTTPS	yes	●	●	●	●	●	●	●
	Service-α	HTTP/S	yes/no	●	●	●	●	●	●	●
Translator	Google Translate	HTTPS	yes	●	●	○	○	—	●	—
	Bing Translator	HTTPS	yes	●	○	○	○	—	●	—
	Weblio	HTTPS	yes	●	○	○	●	—	●	*note*
	PROMT Online	HTTP	yes	●	○	○	○	—	●	—
	Service-β	HTTPS	yes	●	●	○	●	—	●	—
	Yandex.Translate	HTTPS	yes	●	●	●	○	—	●	—
	Baidu Translate	HTTP	yes	●	○	○	○	—	●	—
Archive	Wayback Machine	HTTPS	yes	●	○	●	●	—	●	*note*
	Google Cache	HTTP/S	yes	●	○	○	●	—	●	—
	FreezePage	HTTP	yes	○	○	○	○	—	○	—

ated. All of the relayed cookies were without an expiration date, but as mentioned in Section 2.3, this does not prevent a hijacking attack. On the other hand, Hide My Ass! adopts an implicit relay, and additionally, the session of web rehosting itself was protected by a cookie's `HttpOnly` option; in this case, there is no risk of session hijacking.

For Weblio and Wayback Machine, a user cannot login to a rehosted website as these services are not web proxies, but a user can still login to the service itself. These services provide additional features to logged-in users; for example, Weblio offers a vocabulary book or a console for viewing exam results, and Wayback Machine allows a user to view the list of uploaded or favorited webpages. For these services, we discovered that the login session of the service itself can be hijacked using procedures similar to those described in Section 3.2.5. With a hijacked session, an attacker can view the activities associated with the user account and steal personal information such as user names and email addresses from the profile page.

4.2 Evaluation of Fingerprinting

We have shown that many of the web rehosting services are vulnerable to the browsing history theft. The history theft exploits the availability of the website fingerprints, which are extracted from the data written to the browser storage by each website. This section evaluates the effectiveness of the fingerprinting technique used for the browsing history theft. We evaluate the effectiveness of the following three aspects: 1) fingerpritability of websites, 2) lifetime of fingerprints, and 3) fingerprints that leak the time of visits to websites.

4.2.1 Testing the Fingerprintability of Websites

To generate a fingerprint of a website, we combine the keys extracted from the following three sources: keys contained in the cookie, keys contained in the localStorage, and the keys contained in the JSON dictionary, which is extracted from the values contained in the localStorage. To evaluate the distinguishability of the generated fingerprints, we performed experiments using the websites listed in the Alexa top-10 K [40]. Of the 10 K websites, we eliminated the ones that did not complete the session within 15 seconds. As a result, we found 6,500 websites were reachable via the web rehosting services. We visited each website twice via a web rehosting service. We used ProxySite as a web rehosting service for this experiment because of its fast response. We do not believe that using any of the service significantly affects fingerprintability. The browser used for the experiments was cleaned up, i.e., delete browsing data, after we visited a website. This was performed every time.

Let $\mathbf{W}$ be a set of websites to be examined and $F(w)$ be a fingerprint of a website $w \in \mathbf{W}$. $F(w)$ is determined as

$$F(w) = \mathbf{K}(w, 1) \cap \mathbf{K}(w, 2),$$

where $\mathbf{K}(w, n)$ is a set of keys extracted from the website w for the n-th trial ($n \in \{1, 2\}$). That is, if we found a set of keys that appeared in both trials, we extract them as the fingerprint of the website. A fingerprint, $F(w)$, is distinguishable (unique) if it satisfies the following condition:

$$F(w) \not\subseteq \bigcup_{\forall x \in \mathbf{W} \setminus \{w\}} F(x).$$

We tested the condition shown above for the 6,500 websites which were accessible. We found that the fraction of websites uniquely identifiable by our proposed fingerprint was 39.1% (2,541). **Table 5** presents the top-10 categories for the fingerprintable websites. We have identified these categories based on the Alexa [40] list. We found that the categories of fingerprintable websites contained the ones that are preferred by people with specific attributes, such as anime and programming, as well as the ones that everyone visits, such as news and portals. This observation implies that an attacker can estimate the profile of victims by the history theft attack. Moreover, we found

Table 5 Top 10 categories of fingerprintable websites.

Category	# domains
E-mail	210
Chat	125
Adult	124
Videos	116
News	72
Animation	57
Portals	55
Encyclopedias	48
Programming	43
Photos	40

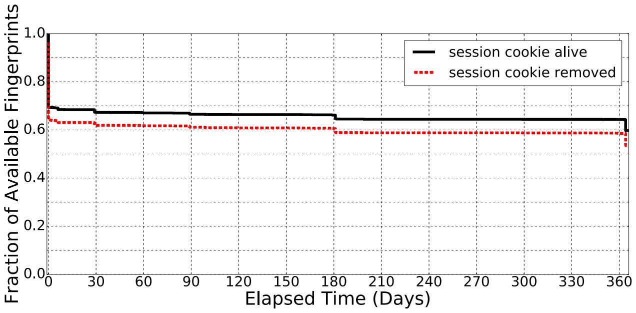


Fig. 6 Relationship between the elapsed time (days) after a user visits a website and the fraction of fingerprintable websites.

Table 6 List of investigated browsers.

Browser	Abbreviation	Version
Google Chrome	Chrome	76.0.3809.132
Mozilla Firefox	Firefox	69.0
Microsoft Internet Explorer	IE	11.0.140
Microsoft Edge	Edge	42.17134.1.0
Apple Safari	Safari	12.1.2
Opera	Opera	62.0.3331.116
Brave Browser	Brave	0.68.132
Google Chrome for mobile	Chrome-M	76.0.3809.132

fingerprintable websites, including sensitive sites such as porn, dating, and piracy websites. The examples taken into consideration were `porn[.]com`, `theporndude[.]com`, `tinder[.]com`, `match[.]com`, `pirate-bay[.]net` and `nyaa[.]si`. The visits of a user to these websites may be used for further attacks, such as social engineering.

Although we recognize that the identifiable rate depends on the size of **W**, setting it to 10 k is not an overestimation for the following reasons. First, as previous studies [41], [42] showed, most Internet traffic is concentrated on top-ranked websites; therefore, users rarely have keys issued by the websites less popular than the top-10 k. Second, a website often stores data with a unique key due to the use of third-party modules. For instance, a website with a code for the Google tag manager [43], which is part of the traffic analysis suite, writes a cookie with a unique ID for each website as a key. For more precise fingerprinting, we need to consider pages that change over time and pages other than the main page of each domain name. We leave these issues for our future work.

4.2.2 Lifetime of Fingerprints

The evaluation above did not consider the expiration of the cookie. However, cookies will be deleted after the expiration date, leading to a decrease in the uniqueness of a fingerprint. In addition, as mentioned in Section 2, cookies that do not have an expiration date are treated as session cookies, which will or will not be deleted when the browser process is terminated, depending on the user's environment. Note that unlike cookies, localStorage will keep the data persistently. To study the impact of elapsed time on the uniqueness of fingerprints, we performed an experiment that simulates the deletion of expired cookies after a user visits a website. We assume that a user visits each website just once.

Figure 6 shows the change of availability for website fingerprints. Two scenarios are shown: when all the session cookies are alive and when they are expired. When the elapsed time is zero, the percentages of available fingerprints are 100% (for session cookies that are alive) and 96.4% (for session cookies that are expired). After one day of user visits, the percentages drop to 69.4% and 64.2%, respectively. Cookies that do not need to be stored for a long time often have an expiration date of fewer than 24 hours. The declining trend after day two becomes mild as a cookie with a long lifetime and a persistent localStorage key contributes to the uniqueness of the fingerprints.

As several websites set the expiration date in units of months, such as one, two, or three months, we can see small spikes at the 30th, 60th, and 90th days. The percentages of available fingerprints on day 364 are 64.3% (session cookie alive) and 58.6% (session cookie expired), but on day 365, they drop to 59.7% (session cookie alive) and 53.6% (session cookie expired) in only one day. This phenomenon reflects the fact that many websites have set the expiration date for cookies to exactly one year. To summarize, we found that more than 50% of website fingerprints still work for the history theft for one year after the website visit.

4.2.3 Fingerprints Leaking User Visit Time

As the cookie expiration time is typically set to the visit time plus a period of time, the visit time can be deduced by subtracting this increment from the cookie expiration time contained in the fingerprint. This estimation makes it possible for an attacker to track users more precisely by reflecting the time series of website transitions. We identified a website with a cookie with the unique key that has a constant difference between the access time and the expiration date, that is, a website that adds a certain value to the access time and sets the expiration date. As a result, we found that 73.6% of fingerprints leaked visit time.

4.3 Resource Accesses Behavior for Each Browser

We now discuss the browser differences in resource access behavior and their impact on attack success and failure. We investigated eight browsers as shown in **Table 6**: Chrome, Firefox, IE, Edge, Safari, and Opera which are major PC browsers; Brave [44] which is known for its strong privacy policy; and the mobile version of Chrome. All versions are latest as of September 2019.

The results are shown in **Table 7**. As a service worker and AppCache are both available on all browsers, with the only browser not supporting a service worker being IE, all browsers are susceptible to persistent MITM attacks.

We then investigated the behavior when attempting privileged access. Browsers labeled "ask once" have especially high risk of privilege abuse because once resource access is permitted, it will not require confirmation for accesses after that. For browsers with the label "selective", users are asked to select whether or not

Table 7 Differences in resource access behavior for each browser. Behaviors highlighted in bold are highly vulnerable. SW stands for service worker and PM stands for password manager.

	III-B1		III-B2	III-B3	III-B4 & B5
Browser	SW	AppCache	Privilege	PM	Session Cookie
Chrome	**available**	**available**	**ask once**	**autofill**	**keep by config**
Firefox	**available**	**available**	selective	**autofill**	**keep by config**
IE	unavailable	**available**	selective	**autofill**	never keep
Edge	**available**	**available**	**ask once**	**autofill**	never keep
Safari	**available**	**available**	selective	manual	**keep by config**
Opera	**available**	**available**	**ask once**	**autofill**	**keep by default**
Brave	**available**	**available**	**ask once**	manual	**keep by default**
Chrome-M	**available**	**available**	**ask once**	**autofill**	**keep by default**

to remember the permitted state, and the attack succeeds only if a user selects yes.

Regarding password managers, browsers labeled “autofill” automatically fill the password field with a saved password upon page load, making credential theft feasible. On the other hand, browsers labeled “manual” focus the login form element but do not fill unless explicitly directed by a user, so the attack is highly infeasible. Note that the password managers for Firefox, IE, and Edge do not autofill when multiple credentials are stored for a single origin. That is, the attack succeeds if the victim stores only a single credential in the password manager through single web rehosting. The password manager for Chrome, Opera, and Chrome-M always autofill the credential that was used most recently.

Lastly, regarding session cookies, browsers labeled “keep by default” do not remove session cookies on quitting the browser. For browsers with “keep by config,” by changing the configuration to save tabs on quitting, session cookies will not be removed. For these two groups, as an attacker will have a longer opportunity to steal session cookies, there is a higher risk of session hijacking and history theft. Note that cookies relayed in web proxy services are mostly session cookies, so are also vulnerable under this environment. On the other hand, browsers labeled “never keep” delete session cookies on quit, which will make website fingerprinting more difficult by decreasing the number of websites identified.

5. Discussion

5.1 Coverage of Our Experiments

In our experiment, we investigated 21 popular web rehosting services spanning across our three categories, and we believe that this coverage should be sufficient for capturing the overall characteristics of each category. For web translator and web archive, the services we chose are those with a dominant market share, and by their nature, the rest of the services in these categories are likely to be similar as the chosen ones. On the other hand, since web proxy services are often used for anonymization and censorship avoidance, they may exhibit regional or infrastructural differences, implying that services in the long-tail are non-negligible; we believe that our investigation of a wide variety of web proxies including OSS (Service-α) sheds light on unexplored services based on a common architecture, i.e., using a single origin to rehost many websites on different origins.

Another notable point is that there exist several web rehosting services that do not fit into our three categories. Tor2web [45] rehosts websites deployed on the Onion network to make them accessible from the public network. GitHack [46] rehosts user-specified GitHub files by giving them appropriate MIME types, making them renderable by a browser. Zone-H [47] archives compromised websites and allows users to conduct security analysis on them. More of such services are expected to exist, but we believe they are not so significant in terms of user impact. Most importantly, our analysis based on observation of rehosting rules will be able to check services that we did not directly cover in this paper. We claim that this is a broad contribution to the research community and web service providers.

Finally, we recognize that we did not discuss every possible attack on web rehosting. Lerner et al. [48] demonstrated that the iframe feature can be exploited to manipulate archived websites. This attack is based on the property that the origin of the iframe’s parent and child pages is unified in web archives. The persistent XSS proposed by Steffens et al. [49] is also useful for compromising web rehosting services. This is an attack that persistently establishes XSS for websites that reflect the strings contained in cookies or localStorage without sanitization. Although its ability (inserting JavaScript into some rehosted websites) is a subset of our persistent MITM (manipulating the content of all rehosted websites), it is harder for web rehosting services to defend against persistent XSS resulting from flaws in the rehosted website, not the web rehosting service.

5.2 Human Factors

Web rehosting services, with tens of millions of users, have become quite popular on the modern Internet. In this paper, we have presented a critical threat that commonly lies within these services and proved that many services are actually vulnerable to it. On the other hand, we have not presented results that support our findings from the perspective of how users make use of these services in reality. For example, the privilege abuse and credential theft threat assume that users have explicitly granted permissions, such as allowing websites they have visited in the past through a web rehosting service to access browser resources or storing credentials in password manager. We are not certain to what extent this assumption holds true in reality, but considering the following points, we believe that it is safe enough to assume that users would grant permission without suspicion. The required permission resources are commonly requested by popular websites. For example, for geolocation information, one of the resources for which permission is required, is essential for route navigation and optimization of search results. In addition, the four most popular password managers have already had over 60 millions of users [50], and automated password filling has become a more common practice [51]. Furthermore, as the web rehosting service and rehosted page are legitimate websites, it is unlikely that users who are unaware of the threat may believe that the rehosted websites are legitimate.

Moreover, the success of some of our attacks, namely session hijacking and attacks against password managers, highly depends on whether a user would actually login to such services through web rehosting. SNSes offering freedom of speech are high-priority targets in countries with censorship [52], [53], [54], and actually, eight out of eleven popular web proxies we investigated in this paper advertise their capability to be compatible with Facebook, which requires users to login. Considering these points, the use of a web proxy for SNS with a login state to avoid censorship appears to make sense, though quantitatively showing this tendency would require additional research.

5.3 Ethical Considerations

As most of the web rehosting services investigated in this paper were found to be vulnerable to at least one of our proposed attacks, we have shared the details on the threat and possible countermeasures with the affected service providers. Several service providers responded to our findings and implemented countermeasures to address the vulnerabilities we demonstrated. Please note that the names of services requested by providers to not be mentioned in the paper have been anonymized. For instance, Google Translate reviewed its service architecture and adopted a method of isolating subdomains for each web page translated. When translating `example.com`, initially all content was rehosted under the domain name `translate.googleusercontent.com` in Google Translate. After applying our measures, they are now converted to the domain name <`hash`>`--example-com.translate.goog`. Service β implemented an approach of denying the `Service-Worker: script` request header. We will describe the technical features of these countermeasures in the next section. Additionally, we made suggestions to JPCERT/CC, a vulnerability coordinator in Japan, and jointly issued an advisory [55] to prompt awareness among developers and users about security issues related to web rehosting services.

6. Defenses

All Attacks. The root cause of these vulnerabilities is that web resources that are originally designed to be placed in different origins are mixed into the same origin. A straightforward solution to this threat would be to not only use a separate domain name for separating rehosting websites and rehosted websites, but also generate a different subdomain for *each* rehosted website. A major drawback of this solution is that already-generated URLs, which may be referred from somewhere else, will become invalid. This solution would especially impact web archive services, as many of the websites archived are linked from a large number of external websites. The service to investigate the number of backlinks [56] showed that the Wayback Machine archives are linked from more than 4 million pages as of September in 2019. Moreover, they may even be referred from printed publications such as academic papers and court records [48], making replacement of such links unrealistic. Redirecting access to an old URL to a new URL will solve this issue.

Furthermore, as some of the web proxy services listed in Section 4.1 have already been adopted, generating a tentative URL inaccessible by a third person would be viable mitigation. This can be implemented in a similar manner as a general CSRF prevention approach, i.e., by using a POST method instead of GET method for submitting a URL to a web rehosting service and authenticating with a token given only to the user who sent the request. A major drawback of this approach is that it cannot be applied to web rehosting services that assume the sharing of generated URLs to a third person like web archive services.

Switching to a browser's private mode is an effective countermeasure that can be taken on the user-side to completely prevent or mitigate some of our attacks. For example, service worker and AppCache are disabled in the private mode of Edge and Firefox. They are enabled in the private mode of Chromium-based browsers but are deleted on closing the browser window. In all browsers, password manager is disabled, and cookies and localStorage are deleted on closing the browser.

Persistent MITM. To prevent a persistent MITM attack, we need to restrict the behavior of the service worker and the AppCache. When requesting for a service worker script, the browser includes `Service-Worker: script` in the request header. A web rehosting server can prevent registration of a malicious service worker by denying requests having this header. This obviously would also block the registration of a legitimate service worker, but none of the current web rehosting services rewrite the URLs used as the argument of a function, that is, service workers do not function on current web rehosting services/rehosted websites. Thus, this countermeasure can be adopted without any substantial drawback. For AppCache, the use of this function is declared by the manifest attribute of an HTML tag, so force-removing this attribute from rehosted pages should prevent its abuse.

Privilege Abuse. Any access to a browser resource from a webpage inside a sandbox-attributed iframe will fail without raising a permission request. Thus, instead of directly loading the content of a rehosted page at the top level, loading it inside a sandbox-attributed iframe would ensure that users do not grant permission to web rehosting. This has already been adopted in some services such as Google Translate, though we are not certain if this is their intended purpose.

Credential Theft. A possible defensive approach for attacks against password managers is to associate credentials with domain-path pairs, instead of just domains. However, this has been adopted by only a few browsers [57], [58], and additionally, many web rehosting services specify the URL of the rehosted website in a query string instead of in the rehosted URL's path. Thus, this defense requires not only browser-side effort but also service-side effort to switch to a path-based scheme for rehosted URLs.

History Theft. As the history theft attack relies on the accuracy of website fingerprinting, preventing fingerprinting is key to preventing the attack. One possible way to prevent website fingerprinting is to force-remove, from all rehosted webpages, JavaScript code that invokes access to cookies or localStorage. This approach is not perfect as exhaustively detecting obfuscated JavaScript code is known to be difficult. A more aggressive approach would be to remove all JavaScript code, but this would likely result in critical deterioration of the website's appearance or functionality.

Session Hijacking and Injection. Cookies for managing login sessions are set via an HTTP Header according to the rules noted in Section 3.3. Therefore, they can be prevented from being loaded from JavaScript code in rehosted malicious websites by enabling the `HttpOnly` attribute. This is a conventional measure for mitigating session hijacking using XSS.

7. Related Work

7.1 Intermediary Services and their Security

While intermediary services ensure users to bypass restrictions based on geographic regions or circumvent censorship, they enable attackers to mount man-in-the-middle attacks at the vantage points of such services.

HTTP Proxy. The use of open HTTP proxies that allow access from any user is a popular way to counter restrictions and censorship, because it is easy for users to use with minimal browser's configuration. However, many studies found evidence that many open HTTP proxies modify web traffic. Mani et al. disclosed malicious examples of proxies that modify web traffic in various ways, which include injecting ads or cryptominer into HTML content, inserting malware into non-HTML files, and modifying TLS certificates [39]. Tsirantonakis et al. found that about 5% of proxies are engaged in some form of malicious content modification, such as injecting ads, collecting user information, and redirecting the user to malware pages [38]. End users of HTTP proxies are required to use end-to-end encryption (e.g., HTTPS everywhere) and check TLS certificates to ensure end-to-end integrity.

VPN and Tor. Similar to HTTP proxies, VPNs and Tor are often used as tools to counter censorship and preserve users' privacy, although users are required to install dedicated software or configure system settings. A VPN establishes a virtual point-to-point connection, which is encrypted, between end users and the remote network. Previous studies showed evidence of traffic manipulation and information leakage in many VPN services [59], [60]. Tor is an overlay network composed of volunteer relay nodes that ensures anonymous communication (anonymizing the sender of the traffic). Tor circuits, which are essentially encrypted tunnels, terminate at exit relays, and the user's traffic travels all over the Internet to its final destination. Winter et al. developed scanning modules to provoke Tor exit relays to tamper with or snoop on their decoy connections and detected malicious, misconfigured, and sniffing Tor exit relays [61]. Although VPNs and Tor have functionalities of encrypted communication and sender anonymization, malicious intermediary servers inside the services are detrimental to the integrity of communication similar to the case of the HTTP proxy.

Web Rehosting. Web rehosting is also susceptible to the same problems faced with the use of HTTP proxies, VPNs, and Tor that malicious intermediary nodes break the integrity of communication. However, an intermediary node of a web rehosting service in our threat model, which is not malicious but *vulnerable*, is abused by remote attackers and causes a lack of communication integrity. To the best of our knowledge, only a few studies have discussed exploiting vulnerable web rehosting. In 2002, Martin et al. were the first to demonstrate an attack against one of the web rehosting services, namely SafeWeb [62]. SafeWeb was a web proxy service that had the functionality to relay cookies and give web pages the same origin; hence, session hijacking can occur because malicious JavaScript in the rehosted page can read the cookie of other rehosted pages. Compared to the previous study [62], our work has made significant progress in expanding target web rehosting services (such as web translators and web archives) and finding new attacks, which exploit browser resources from traditional to modern. Another similar work by Lerner et al. [48] proposes attacks that rewrite pages on the Wayback Machine, a major web archive. The archive rewrites URLs in archived content to make them refer to archived versions of the same domain name. When a URL is dynamically generated by JavaScript in `<iframe>`, the URL rewriting fails, and a browser makes requests to the live web. If the response of the request from the live web contains malicious JavaScript, the whole archival page is compromised by that JavaScript owning to the ineffectiveness of the SOP. This attack requires a target website to refer to a third-party URL with the expired domain name that the attacker can obtain. Our attacks are more feasible; they are triggered once a victim user visits rehosted malicious pages.

7.2 Abusable Browser Resources

Browser resources provided after HTML5 such as service worker and AppCache, as well as traditional browser resources such as cookies, can be abused for various attacks to accomplish malicious purposes.

Service Worker. The Progressive Web App (PWA) is designed to provide native app-like rich browsing experiences even when a browser is offline. The service worker is a key technical component of the PWA. It is an event-driven worker script implemented in JavaScript [7] and has the ability to intercept network requests from the corresponding website. There have been concerns [63], [64] that this feature could be abused in *web hosting* services that share a single domain name for different users. However, the possibility of abusing a service worker has been overlooked for years in *web rehosting*, as evidenced by the fact that our attack works even on major services provided by the world's top companies. We have demonstrated that an attacker is able to deploy malicious content and scripts on web rehosting services by exploiting the rehosting rules, just like deploying them on web hosting services.

Recent studies have suggested other advanced attack scenarios. Lee et al. demonstrated cryptocurrency mining in the background that uses malicious service workers, and mined Monero coins through verified 225 K transactions in a day [32]. Moreover, Papadopoulos et al. developed a monitoring framework to allow malicious service workers to abuse browser resources and found the following harmful operations: DDoS attacks, distributed password cracking, malicious data hosting, proxies of a hidden network, and cryptocurrency mining [65].

Application Cache. AppCache, one of the features of HTML5, allows web applications to cache content in the storage of a web browser to enable offline access. As in the case of service workers, AppCache has also been known to be exploitable on web hosting services that share a single domain for different users [66]. In addition, Lee et al. demonstrated a timing side-channel attack using AppCache that allows a third-party attacker to identify a cross-origin resource status (e.g., login status of a victim browser) [67]. Goethem et al. also demonstrated that an attacker can reveal personally identifiable information of a target (e.g., social accounts) by inspecting the response time and the size of the cross-origin resource stored in AppCache [68], [69].

Password Manager. Password managers provide helpful functionalities for users: generating unique and strong passwords, storing passwords securely, and using passwords easily (e.g., autofill). Silver et al. examined autofill policies in browser built-in password managers and found that several autofill policies enable an attacker on a malicious website to extract passwords from the user's password manager without any user interaction [35]. Stock and Johns mentioned the risk of XSS-based credential stealing from autofilled login forms [58]. Our credential theft (Section 3.2.3) attack also exploits this vulnerable autofill behavior of password managers.

Cookie. It is well known that HTTP cookies can be stolen by cross-site scripting and eavesdropping. If the target user's cookie is stolen, an attacker with the stolen cookie will gain access to personal information or account functionalities of the target on the corresponding website. `HttpOnly` is an additional attribute of cookies that prevents attackers from remotely obtaining cookies through malicious scripts (i.e., XSS). HTTPS-enabled websites can protect cookies against eavesdropping. Moreover, the HTTP Strict Transport Security mechanism (HSTS) [70] is an HTTP header option (`Strict-Transport-Security`) that enables websites to enforce browsers to only use communication over HTTPS. Although many studies in recent years revealed that `HttpOnly`, `HSTS`, and `HTTPS`, which cannot ensure a cookie's integrity, were only partially deployed on websites [71], [72], [73], both academic and industrial efforts promote and increase the adoption of above technologies. Unfortunately, regardless of whether the above technologies are deployed at websites, an attacker is able to successfully perform our session hijacking and injection (Section 3.2.5) attack by leveraging the property of the web proxy (i.e., cookie relaying).

8. Conclusion

In this work, we explored the security flaws of web rehosting services. Their common characteristic is to offer the service using a single origin to unify many different origins. This "*melting pot of origins*" situation could violate the SOP if not carefully handled. Based on the intrinsic vulnerability of these services, we derived five attacks that target users who use web rehosting services. It is noteworthy that these five attacks exploit several browser resources, such as, service workers, AppCache, browser permissions, password manager, localStorage, and cookies. We argue that these modern and traditional resources enable the attacker to track both past and future activity of the victim. Through the extensive analyses of 21 web rehosting services, we demonstrated that the five attacks are feasible and thus need to be prevented.

Web rehosting, which originally aims to enhance the openness of web access, paradoxically deviates from the end-to-end integrity and gives the opportunity to compromise the communication to a third-party attacker. We successfully improved the architecture of world-leading global services by advancing measures in collaboration with vulnerability coordinators and web rehosting providers. Although we revealed the flaws of web rehosting, further unexpected flaws in the web ecosystem may also exist and will be expanded by the evolution of web features like HTML5 and progressive web apps. Identifying these flaws and developing countermeasures are required to be studied in the security research community. We hope that our work fosters future work on more secure solutions to ensure web security and the openness of web access.

References

[1] Watanabe, T., Shioji, E., Akiyama, M. and Mori, T.: Melting Pot of Origins: Compromising the Intermediary Web Services that Rehost Websites, *NDSS* (2020).

[2] Saltzer, J.H., Reed, D.P. and Clark, D.D.: End-to-end Arguments in System Design, *ACM Trans. Comput. Syst.*, Vol.2, No.4, pp.277–288 (1984).

[3] SimilarWeb: Website Traffic Statistics & Market Intelligence, available from ⟨https://www.similarweb.com⟩.

[4] ProxySite: Free Web Proxy Site, available from ⟨https://www.proxysite.com⟩.

[5] Google: Ten years of Google Translate (2016), available from ⟨https://www.blog.google/products/translate/ten-years-of-google-translate/⟩.

[6] AlNoamany, Y.A., Weigle, M.C. and Nelson, M.L.: Access Patterns for Robots and Humans in Web Archives, *Proc. 13th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL '13*, pp.339–348 (2013).

[7] Groups, W.: Service Workers Nightly, available from ⟨https://w3c.github.io/ServiceWorker/⟩.

[8] Web Hypertext Application Technology Working Group: HTML Standard, available from ⟨https://html.spec.whatwg.org/multipage/offline.html⟩.

[9] Mozilla: Window.localStorage - Web APIs — MDN, available from ⟨https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage/⟩.

[10] Hide My Ass!: Free Web Proxy, available from ⟨https://www.hidemyass.com/en-us/proxy⟩.

[11] HIDE me: The Fastest Free Proxy, available from ⟨https://hide.me/en/proxy⟩.

[12] Sitenable Web Proxy: Web Proxy to Bypass school, university and office firewalls. Facebook, Youtube, gmail unblocker, available from ⟨https://sitenable.com⟩.

[13] FilterBypass: FilterBypass - Your Anonymous, Free Online Web & Youtube Proxy, available from ⟨https://www.filterbypass.me⟩.

[14] ProxFree: ProxFree: Free Web Proxy — Surf Anonymously & Maintain Privacy, available from ⟨https://www.proxfree.com⟩.

[15] Toolur: Free Web Proxy, available from ⟨https://proxy.toolur.com⟩.

[16] Hidester: Hidester Proxy - Fast & Free Anonymous Web Proxy, available from ⟨https://hidester.com/proxy/⟩.

[17] GenMirror: Unblock YouTube Proxy - GenMirror free SSL Web Proxy, available from ⟨https://www.genmirror.com⟩.

[18] UnblockVideos: Unblock YouTube Videos - SSL Encrypted Video Proxy 2018, available from ⟨https://unblockvideos.com⟩.

[19] Google: Google Translate, available from ⟨https://translate.google.com/⟩.

[20] Microsoft: Bing Microsoft Translator, available from ⟨https://www.bing.com/translator⟩.

[21] Weblio: Weblio Translator, available from ⟨https://translate.weblio.jp⟩.

[22] PROMT Service LLC: PROMT - Free Online Translator and dictionary, available from ⟨https://www.online-translator.com⟩.

[23] Yandex: Yandex.Translate, available from ⟨https://translate.yandex.com/⟩.

[24] Baidu: Baidu Translator, available from ⟨https://fanyi.baidu.com/⟩.

[25] Internet Archive: Internet Archive: Wayback Machine, available from ⟨https://archive.org/web/⟩.

[26] Google: Google Cache, available from ⟨http://webcache.googleusercontent.com⟩.

[27] FreezePage: FreezePage, available from ⟨http://www.freezepage.com⟩.

[28] W3C: Permissions W3C Working Draft, available from ⟨https://www.w3.org/TR/permissions/⟩ (2017).

[29] Mozilla: HTTP cookies, available from ⟨https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies⟩.

[30] Homakov, E.: Cookie Bomb or let's break the Internet, available from ⟨https://homakov.blogspot.com/2014/01/cookie-bomb-or-lets-break-internet.html⟩.

[31] Mozilla: Same-origin policy, available from ⟨https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy⟩.

[32] Lee, J., Kim, H., Park, J., Shin, I. and Son, S.: Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications, *Proc. 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, pp.1731–1746, ACM (online), DOI: 10.1145/3243734.3243867 (2018).

[33] LastPass: #1 Password Manager & Vault App, Enterprise SSO & MFA, available from ⟨https://www.lastpass.com/⟩.

[34] DashLane: Start dashing online, available from ⟨https://www.dashlane.com/⟩.

[35] Silver, D., Jana, S., Boneh, D., Chen, E. and Jackson, C.: Password Managers: Attacks and Defenses, *23rd USENIX Security Symposium* (*USENIX Security 14*), pp.449–464, USENIX Association (online), available from ⟨https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver⟩ (2014).

[36] el van Bergen, J.: What Is Mixed Content?, available from ⟨https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content⟩.

[37] Chung, T., Choffnes, D. and Mislove, A.: Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet, *Proc. 2016 Internet Measurement Conference*, pp.199–213, ACM (2016).

[38] Tsirantonakis, G., Ilia, P., Ioannidis, S., Athanasopoulos, E. and Polychronakis, M.: A Large-scale Analysis of Content Modification by Open HTTP Proxies, *Network and Distributed System Security Symposium* (*NDSS*) (2018).

[39] Mani, A., Vaidya, T., Dworken, D. and Sherr, M.: An Extensive Evaluation of the Internet's Open Proxies, *Proc. 34th Annual Computer Security Applications Conference, ACSAC '18*, pp.252–265, ACM (online), DOI: 10.1145/3274694.3274711 (2018).

[40] Alexa: Keyword Research, Competitive Analysis, & Website Ranking, available from ⟨https://www.alexa.com⟩.

[41] Adamic, L.A. and Huberman, B.A.: Zipf's law and the Internet., *Glottometrics*, Vol.3, No.1, pp.143–150 (2002).

[42] Clauset, A., Shalizi, C.R. and Newman, M.E.: Power-law distributions in empirical data, *SIAM Review*, Vol.51, No.4, pp.661–703 (2009).

[43] Google: Tag Manager, available from ⟨https://tagmanager.google.com⟩.

[44] Brave: Hecure, Fast & Private Web Browser with Adblocker — Brave Browser, available from ⟨https://brave.com⟩.

[45] Swartz, A. and Griffith, V.: Tor2web: Browse the Tor Onion Services, available from ⟨https://www.tor2web.orgm⟩.

[46] Puchkin, P.: raw.githack.com, available from ⟨https://raw.githack.com⟩.

[47] Zhone-H: Zone-H.org - Unrestricted information, available from ⟨http://www.zone-h.org/archive/⟩.

[48] Lerner, A., Kohno, T. and Roesner, F.: Rewriting History: Changing the Archived Web from the Present, *CCS* (2017).

[49] Steffens, M., Rossow, C., Johns, M. and Stock, B.: Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild, *Network and Distributed System Security Symposium* (*NDSS*) (2019).

[50] ISE: Password Managers: Under the Hood of Secrets Management, available from ⟨https://www.securityevaluators.com/casestudies/password-manager-hacking/⟩.

[51] Pearman, S., Thomas, J., Naeini, P.E., Habib, H., Bauer, L., Christin, N., Cranor, L.F., Egelman, S. and Forget, A.: Let's Go in for a Closer Look: Observing Passwords in Their Natural Habitat, *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pp.295–310, ACM (online), DOI: 10.1145/3133956.3133973 (2017).

[52] Gillani, W.: Pakistan: Court Blocks Facebook (2010), available from ⟨https://www.nytimes.com/2010/05/20/world/asia/20briefs-Pakistan.html⟩.

[53] Bamman, D., O'Connor, B. and Smith, N.: Censorship and deletion practices in Chinese social media, *First Monday*, Vol.17, No.3 (2012).

[54] Chaabane, A., Chen, T., Cunche, M., De Cristofaro, E., Friedman, A. and Kaafar, M.A.: Censorship in the wild: Analyzing Internet filtering in Syria, *Proc. 2014 Conference on Internet Measurement Conference*, pp.285–298, ACM (2014).

[55] JPCERT/CC: JVNTA#96129397 Falsification and eavesdropping of contents across multiple websites via Web Rehosting services, available from ⟨https://jvn.jp/en/ta/JVNTA96129397/⟩.

[56] Moz: SEO Software, Tools & Resources for Smarter Marketing, available from ⟨https://moz.com⟩.

[57] Bugzilla: Passwords should be stored for host+path and not just host, available from ⟨https://bugzilla.mozilla.org/show_bug.cgi?id=263387⟩.

[58] Stock, B. and Johns, M.: Protecting Users Against XSS-based Password Manager Abuse, *AsiaCCS 2014* (2014).

[59] Ikram, M., Vallina-Rodriguez, N., Seneviratne, S., Kaafar, M.A. and Paxson, V.: An Analysis of the Privacy and Security Risks of Android VPN Permission-enabled Apps, *Proc. 2016 Internet Measurement Conference, IMC '16*, pp.349–364, ACM (online), DOI: 10.1145/2987443.2987471 (2016).

[60] Khan, M.T., DeBlasio, J., Voelker, G.M., Snoeren, A.C., Kanich, C. and Vallina-Rodriguez, N.: An Empirical Analysis of the Commercial VPN Ecosystem, *Proc. Internet Measurement Conference 2018, IMC '18*, pp.443–456, ACM (online), DOI: 10.1145/3278532.3278570 (2018).

[61] Winter, P., Köwer, R., Mulazzani, M., Huber, M., Schrittwieser, S., Lindskog, S. and Weippl, E.: Spoiled Onions: Exposing Malicious Tor Exit Relays, *International Symposium on Privacy Enhancing Technologies Symposium* (*PETS*) (2014).

[62] Martin, D. and Schulman, A.: Deanonymizing Users of the SafeWeb Anonymizing Service, *11th USENIX Security Symposium* (*USENIX Security 02*) (2002).

[63] w3c/ServiceWorker: Header for dissalowing SW registration (and implicitly removing SWs)?, available from ⟨https://github.com/w3c/ServiceWorker/issues/224⟩.

[64] Ellingsen, E.: ServiceWorker is dangerous, available from ⟨https://alf.nu/ServiceWorker⟩.

[65] Papadopoulos, P., Ilia, P. and Polychronakis, M.: Master of Web Puppets: Abusing Web Browsers for Persistent and Stealthy Computation, *Network and Distributed System Security Symposium* (*NDSS*) (2019).

[66] Monorail: Security: AppCache allows MITM of same-origin shared hosting, available from ⟨https://bugs.chromium.org/p/chromium/issues/detail?id=367812⟩.

[67] Lee, S., Kim, H. and Kim, J.: Identifying Cross-origin Resource Status Using Application Cache, *Network and Distributed System Security Symposium* (*NDSS*) (2015).

[68] Goethem, T.V., Vanhoef, M., Piessens, F. and Joosen, W.: Request and Conquer: Exposing Cross-Origin Resource Size, *25th USENIX Security Symposium* (*USENIX Security 16*), pp.447–462, USENIX Association (online), available from ⟨https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/goethem⟩ (2016).

[69] Van Goethem, T., Joosen, W. and Nikiforakis, N.: The Clock is Still Ticking: Timing Attacks in the Modern Web, *Proc. 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS '15*, pp.1382–1393, ACM (online), DOI: 10.1145/2810103.2813632 (2015).

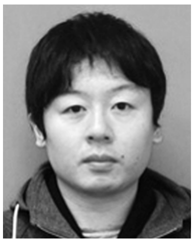
[70] Hodges, J., Jackson, C. and Barth, A.: HTTP Strict Transport Security (*HSTS*), RFC 6797 (2012).

[71] Sivakorn, S., Polakis, I. and Keromytis, A.D.: The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information, *IEEE Symposium on Security and Privacy* (*SP*) (2016).

[72] Kranch, M. and Bonneau, J.: Upgrading HTTPS in Mid-Air: An Empirical Study of Strict Transport Security and Key Pinning, *Network and Distributed System Security Symposium* (*NDSS*) (2015).

[73] Zheng, X., Jiang, J., Liang, J., Duan, H., Chen, S., Wan, T. and Weaver, N.: Cookies Lack Integrity: Real-World Implications, *24th USENIX Security Symposium* (*USENIX Security 15*) (2015).

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



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

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

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