

Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications

Jiyeon Lee
School of Computing, KAIST

Hayeon Kim
School of Computing, KAIST

Junghwan Park
School of Computing, KAIST

Insik Shin
School of Computing, KAIST

Sooel Son*
School of Computing, KAIST

ABSTRACT

Progressive Web App (PWA) is a new generation of Web application designed to provide native app-like browsing experiences even when a browser is offline. PWAs make full use of new HTML5 features which include push notification, cache, and service worker to provide short-latency and rich Web browsing experiences.

We conduct the first systematic study of the security and privacy aspects unique to PWAs. We identify security flaws in main browsers as well as design flaws in popular third-party push services, that exacerbate the phishing risk. We introduce a new side-channel attack that infers the victim's history of visited PWAs. The proposed attack exploits the offline browsing feature of PWAs using a cache. We demonstrate a cryptocurrency mining attack which abuses service workers. Defenses and recommendations to mitigate the identified security and privacy risks are suggested with in-depth understanding.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; *Spooofing attacks*; *Phishing*; *Browser security*;

KEYWORDS

progressive web application; web push; phishing; history sniffing; cryptocurrency mining

ACM Reference Format:

Jiyeon Lee, Hayeon Kim, Junghwan Park, Insik Shin, and Sooel Son. 2018. Pride and Prejudice in Progressive Web Apps: Abusing Native App-like Features in Web Applications. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*, October 15–19, 2018, Toronto, ON, Canada. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3243734.3243867>

1 INTRODUCTION

Progressive Web App (PWA) is a new generation of Web applications. It offers a seamless native-app experience when browsing a

website that employs PWA features. Specifically, a PWA provides offline Web browsing experiences as well as interactive user services by making full use of cache [49], push notification [30] and service worker [31]. The harmony of these new HTML5 features blurs the boundary between native and Web applications particularly in mobile devices, promoting short-latency rich Web experiences.

Figure 1 illustrates two representative PWA features, push notification and offline browsing. A PWA site can send a Web push message, and a user's browser shows the push notification to notify the user as shown in Figure 1 (a). Figure 1 (b) shows a unique PWA feature offering an offline browsing experience, whereas a standard website supports no functionality when a browser is offline. Both push notification and offline usage features are built on the key technical component of service worker. A service worker is an event-driven Web worker that runs in the background. PWAs implement their native app-like features in various event handlers of service workers.



(a) An example of a Web push notification



(b) An illustration of offline usage

Figure 1: Representative features of PWAs

Google introduced PWA in 2015 and has encouraged website owners to migrate into PWAs [15]. Instantaneous installation, offline browsing experience, and user notification features attract website owners and motivate them to implement their sites with PWAs. Numerous Web services have promoted their PWA deployment success stories along with their technical advances [18]. Representatively, AliExpress and Flipkart, two large e-commerce sites, attested that their PWAs contributed to significant increases in the conversion rates and customers' shopping times [13, 14].

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CCS '18, October 15–19, 2018, Toronto, ON, Canada

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-5693-0/18/10...\$15.00

<https://doi.org/10.1145/3243734.3243867>

Despite the vast attention that PWAs have gained, to our knowledge, there has been no research that analyzed the security and privacy risks unique to PWAs. Previous research investigated spam and phishing campaigns [57, 67] and stolen credentials via Web phishing kits [70, 79]. Many researchers have also assessed the privacy risk of side-channel attacks, which allows network and Web attackers to learn visited websites and privacy-sensitive information [1, 6, 7, 22, 26, 61, 64]. However, all of these analyses were based on HTML5 features on standard websites and not on the unique risks brought by PWA features including push notification, cache, and service worker.

Our contributions. We conducted the first systematic analysis of the security and privacy risks on new HTML5 features unique to PWA. Furthermore, we addressed malpractices in third-party push services that expose PWA users to new phishing risks.

We carried out an empirical study on the prevalence of PWA websites on the Internet. By analyzing the front pages of the Alexa top 100,000 domains, we found 3,351 PWA sites using push notifications and 513 sites providing offline services. We collected a dataset for further analysis of PWA sites in the wild.

We started by analyzing the phishing risk via push notification, which has been overlooked in the context of PWAs. Based on the observed push notifications from the collected PWAs, we determined that 56% of PWA sites use their corporation or brand logos for their push notifications. This trend opens a door for a phishing attacker to imitate well-known brand logos for phishing via push notifications, causing users to misunderstand message senders. We found that several PWA websites have already conducted phishing attacks by exploiting WhatsApp and YouTube icons. Our finding assures that the domain name shown in a push notification is the only component that tells its recipient the origin of the notification sender.

Despite the importance of the domain name in a push notification, we found that popular browsers including Firefox for Linux-based desktop and Samsung Internet for Android do not show the domain name in a push notification, but only the thumbnail icon and message. Firefox for Android also shows no domain when the push notification panel is full of other notifications.

Furthermore, the current malpractice of prevalent third-party push services has been leading users not to check push notification domains. Based on the collected PWAs and third-party push services, all such services support push notifications on HTTP sites. Because all browsers allow only an HTTPS site to employ push notification, a third-party push service redirects a user to its own HTTPS site, then asks the user to grant push notification permission for the redirected website. However, the user usually has no clue how the redirected HTTPS site is associated with the HTTP site which the user visited in the first place. The user thus makes an uninformed decision based on the redirection and not based on its domain.

We also investigated eight popular third-party push services and their library scripts. While analyzing how they ensure the integrity of their push messages, we discovered a security flaw that allows a network attacker to spoof the domain of push messages. The addressed vulnerability is caused by their inherent design flaws, which expose PWA users to new phishing risks.

Cache is another core HTML5 feature that enables the browsing of PWA sites offline. A PWA site caches Web contents when a device is online, and uses the cached contents later when the device is offline. We propose a new side-channel attack that exploits the inherent PWA feature of offline browsing. The attack allows a Web attacker to learn the visited PWA sites of their victim. The attacker lures a victim to visit the PWA site, causing the instantaneous installation of the attacker’s service worker on the victim’s device. The PWA then loads other offline PWA sites within its iframes, the origins of which differ from that of attacker’s PWA site. The successful loading of a PWA within an iframe when the device is offline represents that a user has visited the PWA site before. We experimented on side-channel attacks on the collected PWA sites with diverse kinds of desktop and mobile browsers. We found that the Firefox Android and desktop browsers as well as the Safari desktop browses are vulnerable to our side-channel attack.

We introduce a way of abusing the persistency of a PWA service worker. Because a service worker is able to perform arbitrary computations in the background even after a user leaves the PWA site, a Web attacker is able to abuse such a condition to complete their choice of computations. To demonstrate the practical usability of such an attack, we implemented a PWA site that mines cryptocurrencies with its service worker. We used push messages to distribute transactions and let the service worker verify each cryptocurrency transaction by finding the proper hash value. Thus, the attacker is able to abuse the computation resources of user devices that visit the attacker’s PWA site. As a proof of concept, we mined Monero coins [69] for 24 hours and verified 225,000 transactions by using one service worker.

We concluded with our proposed defenses that mitigate the identified security and privacy risks to guide proud and prejudiced PWA developers.

In summary, our contribution is as follows:

- We present the first systematic study on the security and privacy risks of PWAs from the Alexa Top 100K sites.
- We analyze the phishing risk via push notifications by inspecting 4,163 PWAs in the wild. We discover a security flaw by which the Firefox desktop/Android and Samsung Internet Android browsers show no push notification domain, thus exacerbating the phishing risk.
- We conduct an in-depth security analysis of eight popular third-party push services that cover 69.9% of PWAs from the Alexa Top 100K domains. We point out a malpractice that exacerbates the phishing risk and a design flaw that results in push domain spoofing.
- We introduce a new side-channel attack that abuses a cache. The attack allows a Web attacker to learn the victim’s browsing history on PWAs. We demonstrated that our attack works on the Firefox and Safari browsers.
- We present a new abusive attack that takes advantage of service workers. We implement a cryptocurrency mining attack that abuses the computation power of each page visitor’s service worker.
- We suggest mitigations for the addressed security and privacy risks.

2 BACKGROUND

Progressive Web App (PWA) generally refers to a website that utilizes a list of new HTML5 features including the service worker, Web push, and cache features. Majchrzak *et al.* defined a PWA as a website that provides offline usage and a new user interface [40]. Because the definition is based on the execution behaviors of a website and depends on the completeness of feature implementations, we provide a simple technical definition of a PWA. Throughout the paper, we define a PWA as a website that registers a service worker at the browser of a page visitor. Because the service worker is a key technical component that enables native-app experiences including offline usage and push notifications, our definition captures all PWAs designed for various purposes.

2.1 Service Worker

A service worker is a new technology component that facilitates the main PWA functionality. It is an event-driven Web worker implemented in JavaScript [31]. An HTTPS website registers a service worker at a browser, binding the service worker to the HTTPS website origin defined by the HTTPS protocol, domain, and port. Thus, each service worker has its own Web origin that bounds internal resources through the same-origin policy (SOP).

A unique feature of a service worker is that each registered service worker runs in a thread that differs from the browser's main thread. Therefore, it runs in the background, independent of the main thread of the associated HTTPS website. In particular, the thread of a service worker runs persistently in the background even when a user closes the website associated with the registered service worker.

A service worker has an event-driven execution model, which requires implementing event handlers for various events exclusive to the service worker. For instance, *fetch* and *push* events are triggered when initiating an HTTP(S) request and receiving a push message, respectively. By leveraging these events and their event handlers, a service worker is able to intercept network requests from its main website, to receive push messages, and periodically to sync cached local contents with a server in the background.

Because the service worker is a fundamental component, a PWA site first registers its service worker when a user visits the website by calling the `navigator.serviceWorker.register` function. The service worker is then installed and activated in the browser of a page visitor without any disruption of granting permissions. The service worker becomes idle when all event handler operations are over, but it continuously wakes up every time when events for the service worker are invoked.

A service worker requires browser support. Currently, major browsers including Chrome 45+, Firefox 44+, Opera 32+, and Edge 17+ support service workers. For security concerns, the service worker is only supported on HTTPS websites. It indicates that each registered service worker script is delivered over TLS,—thus preventing a script injection from a man-in-the-middle (MITM) attacker who attempts to abuse the service worker functionality.

2.2 Web Push

A Web push notification is a fundamental PWA feature, designed to re-engage users with customized content [17]. Unlike mobile

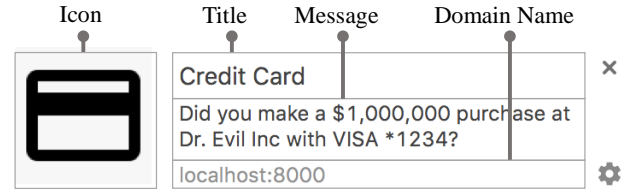


Figure 2: A general appearance of a Web push notification

push notifications managed by user-installed applications, Web push notifications are controlled by desktop or mobile browser instances. Therefore, PWA site owners do not require users to install applications to show push notifications. In this paper, we focus on Web push notifications and use the term *push notification* interchangeably.

A push notification is a browser window alert that contains a push icon, a push message, and its sender's domain. There has been no standard UI for push notification, however WHATWG has specified a list of required elements including title, body, and origin [32]. Figure 2 shows the general appearance of a push notification that most browser vendors implement. Many Web services, including Gmail, Facebook, and Twitter, have already deployed push notifications that inform users of important notices, or display interesting icons for users to click, thus re-engaging the users by redirecting them to particular web pages.

PWA visitors generally go through the following steps to receive a push notification. First, when a user visits a PWA site, the browser automatically registers a service worker of the site. The website then asks the user for permission to receive a push message. If the user approves, the website owner becomes able to send push notifications. The registered service worker running in the background receives a push message from the PWA site and shows a pop-up push notification to the user. The PWA site owner can still send push messages even after the user closes the PWA website tab or the browser window, as long as the browser process continues running.

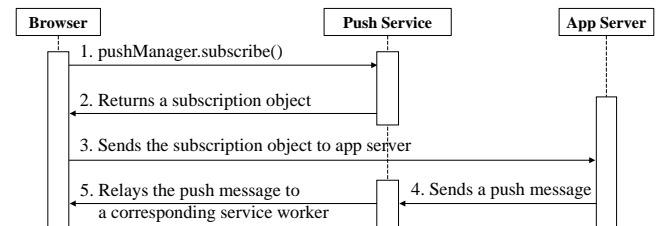


Figure 3: The basic procedure of a Web push notification

Figure 3 illustrates the basic procedure of how a Web push works. There is a new entity called *push service*, a sub-system that each browser vendor manage to support push notification services. Push service serves as a broker that receives push messages from a PWA website server and delivers them to the subscribed users.

- (1) When a user grants the push notification permission for a website, the user's browser is subscribed to its push service after the client-side script calls `pushManager.subscribe()`.

- (2) The push service then returns a *subscription object* that includes an *endpointURL* over TLS. The *endpointURL* is a capabilityURL, composed of the address of the push service and a unique identifier. This identifier represents the user’s service worker, a recipient of push messages originated from the website.
- (3) The script at the client-side browser sends the subscription object to the website server.
- (4) With the subscription information, the website owner can send push messages to the subscribed users.
- (5) When the push service receives a push message from the website server, the push service resolves the unique identifier from the *endpointURL* and relays the push message to the corresponding service worker at a user’s browser.
- (6) The user’s browser wakes up the service worker, which is responsible for displaying the push notification by invoking a *push* event.

VAPID. The integrity of a push message depends on the secrecy of an *endpointURL* in the Web push protocol above. Consider that a website leaks an *endpointURL* at Step 3 in Figure 3 when the client-side script sends it. An adversary who obtains this *endpointURL* becomes capable of sending push notifications to the subscriber with the valid domain name of the website. Because the basic push protocol does not bind an *endpointURL* to its creator, a PWA owner, anyone with a valid *endpointURL* can send a valid push message to the subscriber that the *endpointURL* indicates.

VAPID, a Web Push protocol extension, is designed for a push service to authenticate an application server that sends a push message [19]. When VAPID is employed, the push service blocks push messages from entities without proper authentication.

Specifically, VAPID utilizes an asymmetrical key pair. The public key, termed *applicationServerKey*, is passed to a push service when a service worker subscribes to push service (see Step 1 in Figure 3). When the PWA owner sends a push message, the owner signs the push message with the private key and sends it to the push service. The push service checks the validity of push messages with the stored public key and relays the push message with their valid signatures.

Unfortunately, using the VAPID protocol is not a requirement. It is optional for each PWA developer to check the authenticity of push message senders via VAPID.

Push Message Encryption. The Web Push protocol also supports encrypting a push message payload so that a push service is unable to see its content while relaying the push message.

When a client’s browser sends a subscription object from a PWA website to its server (see Step 3 in Figure 3), the browser appends the two keys *auth* and *p256dh* to the subscription object and sends it to the server. *p256dh* is a client public key that the PWA server uses to encrypt a push message payload. *auth* is a shared authentication secret between the PWA server and the client. Thus, a subscription object that consists of *endpointURL*, *p256dh* and *auth* should not be tampered with or directly inspected by any entity except for the PWA server from which a user elects to receive push notifications.

2.3 Cache

The network dependency of Web applications has hindered browsing experiences. The offline Web Application (or AppCache) is one of the attempts to free Web applications from inherent network dependency. AppCache [75] enables a Web application to cache resources in local storage for offline access. However, it is error-prone, and also hard to provide a complete offline experience because of the overhead of managing numerous manifest-typed resources. It is being deprecated by most browser vendors [48].

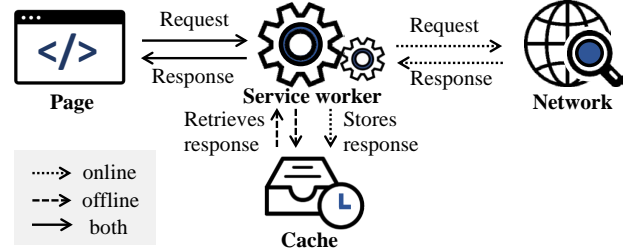


Figure 4: An illustration of cache usage

Recently, a new HTML5 feature, termed *cache*, was introduced. Cache [49] is an origin-bounded local storage that is accessible regardless of the network status. This new feature becomes more powerful when combined with a service worker. For example, as shown in Figure 4, a service worker can either load resources from the cache storage or fetch them through the online network according to the network conditions. These programmable interfaces dramatically improve the online and offline browsing experiences of Web application users.

Cache is supported by most major browsers including Chrome 46+, Firefox 44+, Opera 33+, Safari 11.1+, Edge 16+, and also Samsung Internet 4+ for mobile environment. We cover all of these browsers in our experiments.

3 A METHODOLOGY OF COLLECTING PWAS

Despite the wide attention that PWA has gained, there is little information on the current deployment of PWAs on the Internet. This lack of information hinders understanding the security and privacy impacts brought by vulnerable PWAs.

We investigated the front pages of the Alexa top 100,000 domains and collected PWAs in the wild. Recall that our definition of PWA is a website that registers a service worker (see Section 2). For each main page of the 100,000 domains, we checked whether a website registers a service worker of its own.

We ran a script that forces a Firefox desktop browser to visit 100K websites sequentially. We then extracted all registered service workers shown in the `about:debugging#workers` page, and crawled the JS files that registered service workers.

We observed that scripts from several third-party push services often registered their service workers only after certain user interactions such as clicking on the *allow* button placed in the css-styled permission dialog as shown in Figure 7 (a). To cover such websites with third-party push services, we first identified third-party push services among the Alexa top 100K websites.

Features Used	# of Websites (% Percentage)
Push	440 (10.6%)
Push with library	2,911 (69.9%)
Cache	513 (12.3%)
Both	196 (4.7%)
Others	495 (11.9%)
Total	4,163 (100%)

Table 1: PWA statistics for the Alexa top 100,000 sites

For each of the crawled JS files, we checked its source domain and found prevalent domains appearing across the crawled JS files. We then performed keyword search at Google with such prevalent domains to check if the domains are third-party push vendors. We found 2,911 websites with third-party push services. For those websites, the authors manually visited them and clicked buttons that grant push permission.

We conducted a further analysis to check whether a PWA uses a cache. We modified the Firefox browser to emit the logs when accessing any cache object. With the modified Firefox, we visited each PWA identified from the previous step and decided whether the PWA uses a cache.

Our collection method has limitations. It may miss PWAs that require certain user events to register service workers. Such events may include clicks on certain DOM elements or keyboard events. However, the missed PWAs pose less of a threat because it becomes more difficult for an attacker to exploit their service worker, push notification, or cache.

Table 1 shows the statistics of our collected PWAs. Among the Alexa top 100,000 domains, 4,163 are PWAs that install service workers at the browser. Among the 4,163 PWA websites, 3,351 (80.5%) use push notifications and 513 (12.3%) use the offline cache functionality. Others represent websites with service workers that uses neither push notifications nor cache.

We observed that 2,911 sites (69.9%) of the PWAs implement the push notification functionality by deploying scripts from third-party push services. These services offer script libraries so that a standard website is able to support a push notification by embedding one of their libraries. The top eight most prevalent services are OneSignal [53] (2,046 sites), SendPulse [62] (364 sites), Pushcrew [54] (126 sites), Izooto [34] (65 sites), Pushengage [55] (53 sites), Pushwoosh [56] (47 sites), Foxpush [25] (28 sites), and Urbanairship [2] (20 sites). They cover 86.9% of PWAs out of 3,351 sites that support push notification. Our analysis on PWAs in the wild confirms the prevalence of third-party push services, which also pose security and privacy risks caused by their potential vulnerabilities.

To support open science and further research, we publish the list of collected PWAs tagged with push notification and cache usage at <https://www.github.com/ppp-ccs2018>.

4 THREAT MODEL

We assume two attack models: *PWA attacker* and *Network attacker*.

PWA Attacker. PWA attacker is a classic *Web attacker* [3]. The attacker controls his/her own PWA website and entices users into

visiting the website. The attacker’s service worker is instantly registered at a victim’s browser once a victim visits the site as explained in Section 2.1.

We additionally extend the Web attacker model and assume that a user may grant permission for push notifications on an attacker-controlled PWA. As a result, the attacker has the ability to *send* and *customize* push messages which notify their visitors even when they are not on the attacker-controlled PWA. Furthermore, the attacker has no limitation of abusing their own service worker and cache.

Network Attacker. We assume an active network adversary who is capable of monitoring, intercepting and modifying network traffic over the HTTP protocol. Specifically, the attacker can eavesdrop on messages as well as alter HTML or JS code sent over the HTTP protocol. In previous research [6, 7, 43, 61, 63], an active network adversary has shown to be a practical threat to Internet users, exfiltrating passwords and inferring online behaviors. We assume that a network attacker can monitor or selectively revise an HTTP website with a third-party push library.

5 PHISHING VIA PUSH MESSAGES

Phishing is one of the most effective and devastating Web threats that harvest users’ credentials as well as privacy-sensitive information [70]. A PWA attacker can launch a phishing campaign by abusing push notifications. The attacker entices users with innocuous Web content and requests push permissions on the attacker-controller PWA. Later, the attacker crafts a push message with her choice of destination URL to redirect victims, and then sends it to all past visitors. All past visitors with service workers from the attacker’s PWA receive the phishing push notifications that redirect the victims once clicking the notifications.

From the perspective of a phishing attacker, a Web push is a juicy content delivery system. Phishing via push messages has two advantages over classic email phishing: (1) the attacker can actively show a push notification at a time of her choice, and (2) it is difficult for a push message recipient to determine the origin of a received message. Because a push notification pops up even when a victim is not on the attacker-controlled PWA, a phishing attacker can effectively show a push notification at the time that the victim is most likely to click the push notification.

The only information for a push message recipient to know the message origin is the domain appearing in the push notification dialog. However, its portion in the dialog is relatively small compared to other visible components (See Figure 2). Note that the previous research demonstrated that users paid little attention on a small display in the peripheral area of a browser, compared to the large main window [73, 77]. It is also highly likely for users to place little attention on a push notification domain.

In this section, we introduce a phishing method via push message that exploits the current trend of using company and brand logos for push notification icons. We also present browser security flaws of showing no domain name in a push notification, thus exacerbating the phishing risk.

Push Icon Category	# of Websites (% Percentage)
Company/Brand Logo	390 (56.2%)
Article Thumbnail	226 (32.5%)
Default (Bell-shaped)	22 (3.2%)
None (Blank)	56 (8.1%)
Total	694 (100%)

Table 2: Push icon usage statistics for 694 PWAs

5.1 Phishing by Manipulating Push Notification Icons

Generally, a push notification has a domain name component that indicates where the push message originated. We argue that besides a domain name, a push icon contributes to the user’s understanding of the origin of a received push message. We collected push notification icons from 3,351 PWAs from our dataset (see Section 3). Because we have no control over enforcing such PWAs to send push messages, we collected icons from the received push messages for three days.

Among the 694 websites that showed push notifications, 390 (56%) sites used their corporation logos for push icons as shown in Table 2. 32% of the domains use push icons for summarizing articles, or advertising products. Thus, it is natural for users to educate themselves to infer a push message sender based on its push icon.

While examining push icons, we came across real-world push notifications that attempted phishing as well as, two domains that imitate popular brand logos including WhatsApp and YouTube for their push icons. Figure 5 shows such captured instances. megafilmesonlinehd.org uses the YouTube icon to welcome their subscribers. pornkino.to promotes online-dating opportunities in German with the WhatsApp logo. We note that the Chrome logo displayed in the third push notification appears in the Chrome browser under the MacBook environment.

We also received a push notification claiming “New iPhone X is reserved for you. Delivery to your doorstep for 1\$ only!” with an iPhone image as a push icon. Another phishing example with the Chrome icon says “Google Chrome Premium,” enticing users to click on the “DOWNLOAD” button, which leads to installing a Chrome extension.

Our findings confirm that phishing via a push message targets naive Web users and leads users to misplace their trust by manipulating push notification icons. Therefore, the only way for users to know the authentic sender of a push notification is to check its domain name.

5.2 Domain Name in a Push Notification

A domain name in a push notification should be visible because it is the only component for users to check the origin of a received push message. We conducted a comprehensive study of investigating how the domain in a push notification is shown under various execution environments.

We examined the Firefox, Chrome, Opera and Edge browsers under the Windows 10, Ubuntu 16.04, and MacBook Sierra 10.12.2 operating systems. For mobile browsers, we checked UC Browser, Opera,

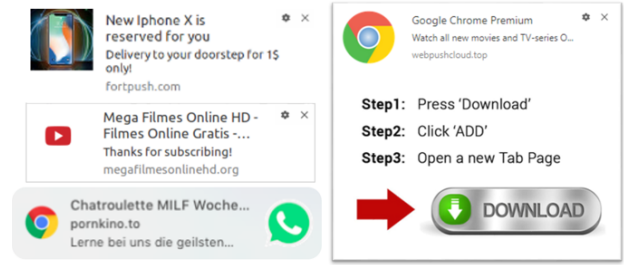


Figure 5: Real-world push examples that imitate popular brand logos and phishing attempts

Brave, Firefox, Chrome and Samsung Internet on Android. Note that the Apple push notification service is revoked recently [11]. Therefore, we excluded the Safari browser and mobile browsers in iOS environment from our study.

We found that the Firefox desktop browser under five Linux-based environments shows no domain in the push notification. Because the desktop browsers in Linux-based environments use an external OSD (On-Screen Display) to show push notifications, they make use of the D-Bus (Desktop Bus) to pass a push notification message to the external OSD. We intercepted RPC calls from browsers to the external OSD varying different desktop environments. We found that Firefox under *GNOME*, *Ubuntu MATE*, *Cinnamon*, *Budgie*, and *Pantheon* doesn’t pass the site URL on a push notification message while Chrome and other browsers do. As a result, Firefox desktop browsers in such Linux-based environment do not show the domain information in their push notifications.

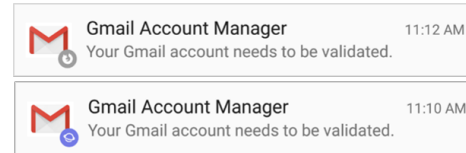


Figure 6: Crafted push notifications in the Firefox and Samsung Internet Android browsers

For Android browsers, we found that the Firefox browser shows no domain in certain cases and the Samsung Internet browser always shows no domain in their push notifications. When an Android device is locked or the notification panel is full, Android abbreviates push notifications. Otherwise, it displays notifications with more details such as a settings button. The Firefox browser on Android shows no domain in the first case. The Samsung Internet browser never shows a domain in their push notification. We include captured images of all usage scenarios in the Appendix.

Figure 6 shows our phishing message displayed on Firefox and Samsung Internet on Android. We implemented the phishing push message to induce a victim to change the password of their Gmail account. The notifications show the Gmail logo without its domain origin, which would reveal the attacker’s domain when displaying them in these browsers.

We reported these security flaws to Mozilla and Samsung, developers are assigned for this issue and they are looking into the issue. Samsung promised the patch for their next version.

We acknowledge that a phishing victim who already clicked a phishing push notification may still see the full URL of a redirected website before entering sensitive information. However, we argue that phishing via a push message is a critical threat. Thomas *et al.* showed that popular Web phishing kits harvest 230K credentials every week [70]. Phishing websites emulating Gmail, Yahoo, and Hotmail logins have managed to steal 1.4 million credentials despite the victims' browsers not showing any valid service domain. A well-crafted phishing push message with no message origin certainly favors the chance of a successful phishing attack.

6 RISK OF THIRD-PARTY PUSH SERVICES

In this section, we address security risks that arise from a third-party push service. Such a service provides a convenient and fast way of enabling push notifications at their client's websites. Generally, a website owner includes a script from a third-party push service, which automatically performs a series of procedures that enable push notifications. The site owner sends a push message to their subscribers by utilizing the Web interface provided from the third-party push service. The site owner can also customize push message titles, message, and icons by utilizing a handy interface provided by the third-party push service.

We analyze the current practice of enabling push notifications on HTTP websites by third-party push services. Section 6.1 explains that the unhealthy practice of redirecting users from a client HTTP site to a third-party HTTPS website has been leading a user to misunderstand the valid origin of a push message that the user wants to receive. A phishing attacker is certainly able to exploit such misunderstandings against innocuous users.

We also investigated how third-party push services handle a push subscription object to preserve its secrecy. Section 6.2 describes two security design flaws that allow spoofing a push notification domain by a network attacker.

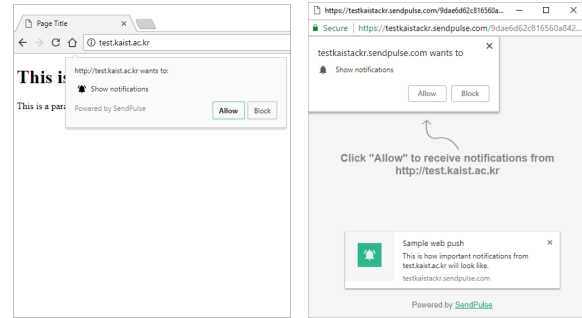
6.1 Prejudice against Third-party Domains in Push Notifications

Popular third-party push services provide various services including sending a push notification, scheduling a push notification, and reporting the statistics of subscribers. One of the most common supports is to enable push notifications for HTTP websites.

As mentioned in Section 2.2, only HTTPS sites are able to register their service workers. Because the presence of a service worker is mandatory to show a push notification, HTTP websites are intrinsically unable to show a push notification. Third-party push services bypass this restrictions by placing a service worker for their own HTTPS domain. For each HTTP site that embeds a script from a third-party push service, the third-party push service assigns an HTTPS domain, a subdomain of their HTTPS domain. The third-party push service then enables the HTTP site visitors to receive push messages from this HTTPS subdomain.

Figure 7 demonstrates this trust transition in two steps. (1) A user visits an HTTP website that implements push notifications using a third-party script from a third-party push service. The script shows

a css-styled dialog that asks the user to accept push messages from the HTTP website. It is noteworthy that the css-styled dialog is not a browser dialog asking for push permission, but a notifying window to inform the user. (2) If the user clicks on "allow", the script redirects the user to the subdomain of the third-party push service HTTPS domain, assigned to the HTTP website. The redirected HTTPS website then pops up a browser dialog asking the user to grant push permission for the HTTPS domain. For HTTP websites with third-party push services, a user who seeks push notifications should give her/his consent twice.



(a) A css-styled permission dialog on a HTTP website that user visited (b) A push permission dialog on a HTTPS websites that library provided

Figure 7: An example of the two-step push permission granting procedure

Risk. The problem arises from users' ignorance of the relationship between an HTTP website and the third-party push service that the HTTP website uses. Users may understand the first consent request because the consent seeks the push permission for the visited HTTP website domain. However, the HTTPS domain name that appears in the second permission dialog partially matches the prefix of the HTTP website domain or uses a random domain prefix with the third-party push service HTTPS domain suffix. Such domain relation between an HTTP website and its third-party push service domain is chosen by the HTTP website owner and not by the website visitors. It is natural for HTTP website visitors to be ignorant. Based on the redirection from the HTTP website to its corresponding HTTPS domain, users should decide whether to accept push notifications from the third-party push service HTTPS domain, of which they may be unaware.

We argue that the current practice of getting a push consent by redirection contributes to the trend of not checking a domain name for granting the push notification permission. Normal Internet users have no way to understand this complicated trust transition chosen by a HTTP site owner, but make an uninformed decision based on the redirection and not on the HTTPS domain in the permission dialog.

Furthermore, a network attacker can take advantage of this trust transition from an HTTP domain to an HTTPS domain. Consider that the network attacker changes the redirection URL after the first consent window from a valid third-party HTTPS domain to the attacker's HTTPS domain. A page visitor should decide whether to receive messages from the attacker's HTTPS domain. Unless the victim who visited the website knows the valid third-party push

service domain in advance, the victim naturally trusts the attacker’s HTTPS domain based on the fact that the first push permission consent redirects the victim to the attacker’s domain. A phishing attacker who seeks the push permission consent on the attacker-controlled HTTPS domain can exploit this malpractice by changing the redirection URLs of popular HTTP websites with third-party push services.

We demonstrated the attack of changing the redirection URL on websites with popular push services in Section 9.1

6.2 Domain Name Spoofing in a Push Notification

We investigated the VAPID protocol deployment in popular third-party push services. Based on the occurrences of third-party script sources in the collected PWA (see Section 3), we selected the eight most prevalent third-party push libraries and checked whether they use *applicationServerKey* when they subscribe to push service (see Section 2.2). Unexpectedly, among the eight third-party push libraries, only two (OneSignal and Urbanairship) implement their Web push systems with the VAPID protocol. One explanation for its low adoption rate is that the VAPID protocol requires an additional step of performing the ECDSA p-256 signing on push messages, which brings performance overheads [68] on vendors’ push servers.

When no VAPID protocol is present, the only required component for a phishing attacker to send a forged message with a spoofed domain is a subscription object leaked from the target domain (see Section 2.2). Thus, we further investigated a possible leakage of subscription objects accessible to a network attacker. We analyzed in/outbound network payloads from/to PWAs with the six third-party push services that do not deploy the VAPID protocol. We used mitmproxy [44], an open-source interactive HTTPS proxy, to inspect and modify Web traffic to mock the capability of a network attacker.

Because a third-party push service internally uses a browser-provided push service¹ underneath a curtain, the subscription object created at the client-side should be delivered to a third-party push service by any means. Therefore, we focused on the subscription object transmission channel from a service worker at the client-side to a third-party push server.

We found two leakage paths that allow a network adversary to obtain the complete subscription information: (1) the transmission of a subscription object over HTTP, and (2) the reflected transmission of a subscription object over HTTPS.

Transmission of Subscription Objects over HTTP. The first leakage path is where a subscription object is sent over HTTP. Any network adversary is capable of harvesting such a subscription object in plain-text. We found that the *Izooto* [34] push service corresponds to this case. Figure 8 describes the overall process of how a network adversary sends a push message with a spoofed domain in a push notification. Consider a vulnerable website with the *Izooto* library. After the *Izooto* script at the client-side generates a subscription object after Step 2, it sends the subscription object to its push service server over HTTP. A network attacker inspects this transmission and extracts *endpointURL* in the subscription

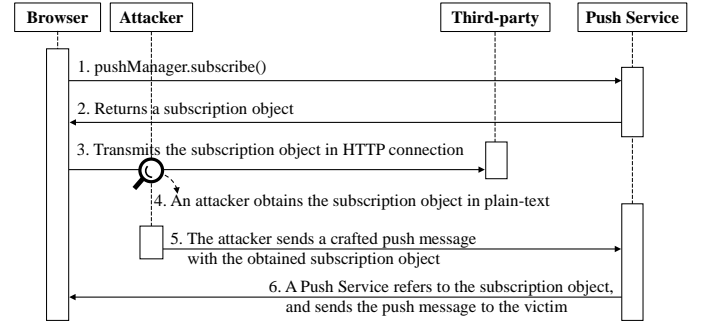


Figure 8: An exploitation of a subscription object leaked over HTTP for a push domain spoofing attack

object. She can send a push message through Steps 5 and 6 and the recipient will see a push notification, the domain of which shows *subdomain.izooto.com* assigned to the target HTTP domain.

Reflected Transmission of Subscription Objects over HTTPS. We present a new attack that exfiltrates subscription objects over HTTPS. The attack exploits a design flaw in popular third-party push libraries. According to our analysis of the eight third-party push libraries, the SendPulse and Pushwood third-party libraries use a variable to hold the destination HTTPS URL for a subscription object to be sent, as shown in Listing 1. However, the problem is that the script that holds this variable is sent over HTTP so that the network adversary changes this variable.

```
var n="https://pushdata.sendpulse.com:4434";
```

Listing 1: A script of defining the subscription object destination URL from SendPulse

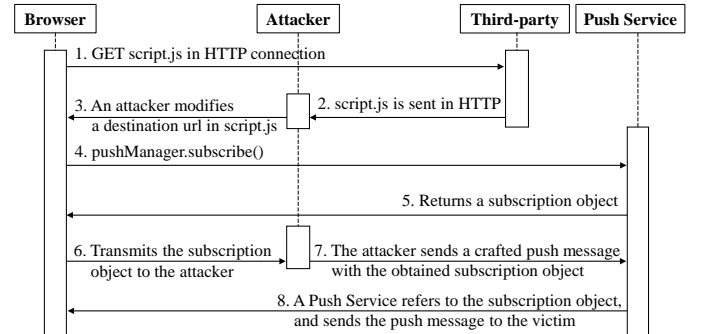


Figure 9: An exploitation of a reflected subscription object sent over HTTPS for a push domain spoofing attack

Figure 9 illustrates the overall network flow of our attack. During Step 6, the network attacker obtains the subscription object, the result of a reflected request originated from *script.js* altered by the attacker in Step 3. Steps 7 and 8 show that the attacker sends a push message by abusing the obtained subscription object.

We conducted the experiments which exploited both leakage paths on real-world PWAs and successfully sent push messages

¹<https://fcm.googleapis.com/> and <https://updates.push.services.mozilla.com/> are the addresses of push services for Chrome and Firefox respectively.

with a spoofed domain name. Section 9.2 explains the details of our attack and its results on third-party push libraries.

7 SIDE-CHANNEL ATTACK ON BROWSING HISTORY

History sniffing attack that leaks a Web user's browsing history has been considered a critical privacy threat [22, 64, 72]. The inferred browsing history can reveal its owner's personal interests, political preferences, medical history, dating preferences, and so on.

In this section, we present a new method that a PWA attacker can use to infer the browsing history of PWAs where his/her victim visited in the past. This new side-channel attack takes advantage of the cache, which a PWA uses to support offline browsing usage. In this attack, we assume that a victim already visited the attacker-controlled PWA and that its service worker automatically stores the attack code in the cache for its offline usage.

Attack. When a victim opens the attacker-controlled PWA in offline, the attack PWA prepares multiple iframes whose sources are the HTTPS URLs of the target PWAs. The attacker also registers an *onload* event handler for each iframe so that the top attacker-controlled PWA knows the loading completion of a cross-origin target website in each iframe.

If the victim visits a target PWA that supports offline usage, an *onload* event handler will be called. Otherwise, an *onload* event handler will not be invoked. We tested our attack against Chrome, Firefox, Safari, Edge, Internet Explorer, UC Browser, Opera and their Android versions as well.

We confirmed that our side-channel attack is effective on the Mozilla Firefox 59.0.2 (Windows 10, Ubuntu 16.04, and High Sierra 10.13) and Safari 11.1 (High Sierra 10.13) browsers. Fortunately, unlike two vulnerable browsers, all other browsers invoke their *onload* event handlers regardless of whether the loading of a target PWA is successful or not.

The difference in the handling the *onload* event stems from each user agent's event handling policy, and not from simple implementation bugs [5, 8]. The living HTML standard describes that *load* event should be fired when a *Document* in an iframe is completely loaded [74]. It also states that it is up to user agents to implement a strict cross-origin policy of firing the event when loading cross-origin resources within an iframe. However, such a policy may not aligned with existing Web content. Our attack exploits this subtle policy difference in the context of PWA offline usage.

The proposed side-channel attack has several limitations. Because of its dependency on the cache, the attacker can only infer visited PWAs that offer offline usage. Frame busting techniques, X-Frame-Options header [58], and Content Security Policy [28] also make our attack ineffective.

The proposed attack also has unique advantages over previous history sniffing attacks [22, 72]. (1) *Accuracy*: Our attack is more accurate than a sniffing attack that exploits the load time differences on cached resources [22]. It is well-known that exploiting the loading time differences is not practical because the loading time is greatly affected by network environments [38]. On the contrary, our attack is deterministic due to its simplicity of checking for offline usage support from a target site. (2) *No outgoing requests*: Because the attacker conducts the attack in the offline mode, there is no

outgoing network request toward a target PWA with any referer header that reveals the attacker's domain. This makes the detection of our attack difficult. (3) *Coverage*: As the offline usage prevails among PWAs in the wild, the coverage of our attack becomes larger.

Above all things, our side-channel attack is a brand new category of history sniffing attacks unique to PWAs.

8 ABUSING SERVICE WORKER PERSISTENCY

A service worker persists in performing event handlers until they are complete even after a user closes or leaves its website. This persistency is a key requirement when syncing local Web contents in the background and showing push notifications in time. At the same time, a PWA attacker is able to abuse such persistency to perform arbitrary computations. The attacker entices a victim to visit an attacker-controlled PWA and thus installs a service worker onto the victim's host. At this point, the attacker is able to perform arbitrary computations on the victim's hosts by triggering registered event handlers in the service worker.

Fortunately, there are limitations to abusing PWA service workers. Major browsers such as Chrome and Firefox provides limited built-in browser objects and API for a service worker to access. For instance, Web socket [76], GPS, and, gyro sensors are inaccessible from a service worker. The *SetTimeout*, *setInterval*, and *XMLHttpRequest* built-in methods are also unavailable.

In this section, we demonstrate a cryptocurrency mining attack that abuses service workers regardless of how limited built-in objects and APIs are provided to them. The proposed attack is designed to exploit computation resources of victims who once registered a service worker from an attacker-controlled PWA.

8.1 Cryptocurrency Mining

Cryptocurrency mining has become a popular way of utilizing surplus computing resources [39]. It also becomes an alternative way of monetizing a popular website instead of exposing advertisements that can annoy the website visitors. The website assigns each visitor a list of cryptocurrency transactions to verify and the visitor's browser then finds valid hash values that validate the assigned transactions by performing numerous trial-and-error hash computations. CoinHive [9] is a popular JavaScript cryptocurrency mining service for website owners who seek mining opportunities from their website visitors.

Once a website embeds a CoinHive cryptocurrency mining script, a host browser that renders the website becomes a cryptocurrency miner. The miner initially connects to a central CoinHive mining pool and then receives a list of transactions to validate via Web-Socket [76]. It then runs multiple Web Workers [29] that validate the received transactions. CoinHive also requires browser supports for WebAssembly [51] to make full use of computation resources. If the miner finds a valid hash value, the miner script sends the hash value to claim its reward for the performed computation.

A PWA attacker is capable of abusing a service worker with push messages when validating cryptocurrency transactions, thus mining coins. The benefits of using service workers for mining cryptocurrencies are two-fold. (1) The attacker has no need to compromise the user's local machine, but requires a victim to visit her PWA and gets the consent for a push notification. (2) The

Monero price(Apr 23, 2018, close): \$283.30

Browser	Execution Environment	Number of Solved Hashes		Amount of Monero	
		Total (24h)	Average (1h)	Total (24h)	Average (1h)
Chrome 65	Windows 10 Desktop	225,024	9,376	0.00001266 (\$0.00358657)	0.00000053 (\$0.00014944)
Firefox 59	Windows 10 Desktop	195,840	8,160	0.00001119 (\$0.00317013)	0.00000047 (\$0.00013209)
Chrome 65	Android 8.0 Google Pixel Phone	50,176	2,091	0.00000282 (\$0.00079891)	0.00000012 (\$0.00003329)
Chrome 65	macOS High Sierra 10.13.4	138,496	5,771	0.00000778 (\$0.00220407)	0.00000032 (\$0.00009184)

Table 3: Monero mining rewards for 24 hours by one service worker

attacker is able to continuously mine cryptocurrency coins even after the victim leaves the website.

To demonstrate the feasibility of mining coins via service workers, we implemented a service worker that mines Monero coins [69]. We refactored the CoinHive mining script to make it workable by a PWA service worker. Instead of WebSocket to fetch transactions from a CoinHive server, we used a cross-origin *fetch* API to make a HTTP request to our proxy server where communicating with the CoinHive server via WebSocket.

The technical challenge of using a service worker for cryptocurrency mining is to keep the service worker running for a long time. Once the service worker registration completes, it lives in a browser “indefinitely” and the browser instantiates a new service worker process when there is an associated event including push event. The process runs continuously in the background even if the tab on the corresponding website is closed. The Chrome browser terminates this service worker process if it has been idle for 30 seconds [21].

Due to the nature of cryptocurrency mining, a service worker cannot start with a long list of transactions to work with because other miners may validate those transactions before the service work completes the task. Therefore, we use a push messages to distribute cryptocurrency transactions as well as to wake idle service workers.

An unfortunate downside of exploiting push messages is that push messages trigger displaying push notifications, which is undesirable for a stealthy mining operation. We thus investigated how not to show a push notification when a service worker receives a push message.

A straightforward way is not to purposely call any Notification API (i.e., `showNotification()`) upon receiving a push message to hide its push notification. We tested our method against all browsers supporting Web push: Whale, Edge, Brave, UC Browser, Samsung Internet for Android, Chrome, Firefox and Opera. We confirm that only UC Browser, Firefox and Edge allow receiving push messages without displaying any push notification. The other browsers show a default warning notification. Chrome shows the message: “*This site has be updated in the background.*”

We observed that Firefox, and Edge revoked their push subscriptions if a service worker ignored displaying a push notification 15 and 3 times, respectively upon receiving a push message. UC Browser did not revoked its subscription as well even when showing no push notification for 100 push messages. Therefore, to maintain continuous stealthy mining operations, we periodically renewed the subscription objects after receiving several consecutive transactions via push message. However, we found that Edge does not allow re-subscription on the background and UC Browser does not

support WebAssembly, which the CoinHive mining script requires. Therefore, our mining attack works against Firefox for a stealthy mining operation.

A PWA attacker is not necessarily limited to conducting her mining attack against victims with Firefox. She is able to conduct a cryptocurrency mining campaign at the time when victims are not likely to be present such as 3:00 AM.

Table 3 shows the experimental result of mining Monero cryptocurrency for 24 hours only by using one service worker. The experiments are performed on *MacBook Air* with 1.3 GHz Intel Core i5 processor (4250U) and 8 GB memory machine, *Windows10* desktop with 3.6 GHz Intel Core i7 processor (7700) and 16GB memory, and Google Pixel Phone. The CoinHive mining algorithm is not optimized in ARM architecture [59], thus resulting in poor performance in the Android 8.0 Pixel device. Using one service worker for mining coin is not as efficient as using multiple Web workers. However, the service worker persists even if a user leaves its website. The more victims visit the website, the more computation capability the attacker has. The attacker is capable of building her/his own service worker botnet, designed to mine cryptocurrencies, neither compromising victims’ machines nor letting victims install malwares.

9 ATTACKS ON PWA IN THE WILD

In this section, we demonstrate the feasibility of the push permission delegation attack in Section 6.1, the push domain spoofing attack in Section 6.2, and the side-channel attack via cache in Section 7 against real-world PWAs.

9.1 Push Permission Delegation Attack

In this section, we demonstrate a push permission delegation attack that redirects a user to an attacker-controlled site. The presented attack exploits the ignorance of a victim about the relationship between a visited HTTP website and its redirected HTTPS website.

As explained in Section 6.1, a user should give consent twice to grant push permissions on an HTTP website that uses a third-party push library. We checked whether a redirection URL is spoofable by a network attacker.

We investigated the eight most popular third-party push libraries (See Section 3). We confirmed that a network attacker is certainly able to manipulate the redirection URLs from Foxpush, SendPulse, Pushwoosh, and Izooto since these library scripts are delivered through HTTP. However, our attack is not necessarily limited to vulnerable third-party push libraries. Because a network attacker

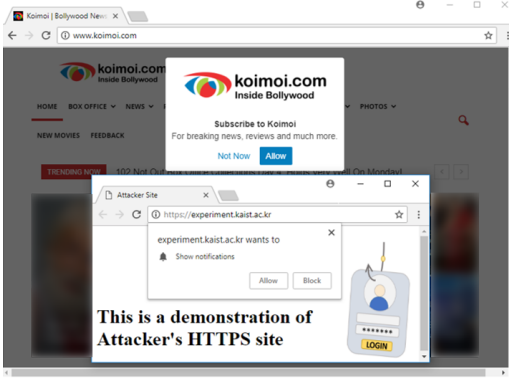


Figure 10: A demonstration of a push permission delegation attack against <http://www.koimoi.com>

has the ability to change the intended semantics of an HTTP website, the attacker can block the consent dialog shown by any third-party push library and display their own consent dialog with the choice of redirection URL.

Figure 10 shows a successful attack launched against <http://www.koimoi.com> that deploys the Pushwoosh library. An attacker takes advantage of the blind trust transition of users from <http://www.koimoi.com> to <https://a756c-03273.chrome.pushwoosh.com>². The attacker can modify the redirection destination from <https://a756c-03273.chrome.pushwoosh.com> to <https://experiment.attacker.com> so that victims will grant push permission to the attacker-controlled domain.

9.2 Push Domain Spoofing by EndpointURL Hijacking

We undertook push domain spoofing attacks that leverage the two leakage paths described in Section 6.2. We assumed the presence of an active network attacker, capable of altering scripts sent over HTTP.

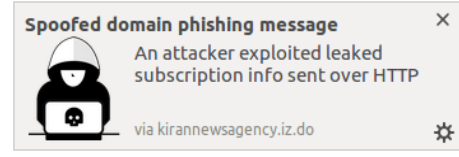
Subscription Object Transmission over HTTP. Among the six third-party push libraries with no VAPID protocol, Izooto is the only library that sends a subscription object over HTTP. Listing 2 shows in-plain text delivered over HTTP with all *endpointURL*, *p256dh* and *auth* information. Any network attacker with such subscription information is capable of sending a phishing message to the victim corresponding to the leaked *endpointURL* with a spoofed domain name.

```
http://events.izooto.com/api.php?s=0&...&bKey=ehWb8IzgWUo
:APA91bGoUSAve140c...&auth=GkyeyQIFenLnLg...&pk=
BPoN-JEpU-oYXmbGle_Q-EoEB...
```

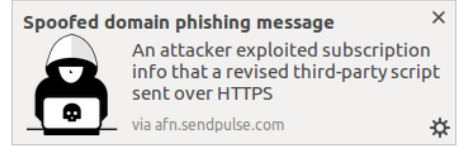
Listing 2: A subscription object instance sent over HTTP

We conducted an experiment with the <http://kirannnewsagency.com> PWA website, where the vulnerable Izooto [34] library was used. When an author grants the push permission for <http://kirannnewsagency.com>, another author exfiltrates a subscription object by inspecting deployed *mitm* proxy logs. We use this subscription

²<https://a756c-03273.chrome.pushwoosh.com> is an HTTPS domain assigned to <http://www.koimoi.com>.



(a) A push message with spoofed domain "kirannnewsagency.iz.do"



(b) A push message with spoofed domain "afn.sendpulse.com"

Figure 11: Demonstrations of push permission delegation and domain name spoofing

information to send a phishing push message to the author who grants the push permission. Remember that the attacker is able to control all visible components in a push notification including its title, message, push icon image, and even the landing URL that redirects a recipient when clicking the push notification. Figure 11(a) shows our crafted push notification with the spoofed domain of <https://kirannnewsagency.iz.do>.

Reflected Transmission of Subscription Objects over HTTPS.

We found that the JS libraries fetched over HTTP from SendPulse, PushWoosh and Izooto contained a variable that holds the destination HTTPS URL (see the Listing 1 in Section 6.2). We changed this value to an attacker-controlled HTTPS domain. Note that SendPulse and PushWoosh have been sending subscription objects over HTTPS. However, their JS libraries enabling push services have been delivered over HTTP.

We conducted an experiment of a push domain spoofing attack against <http://afn.az> with the SendPulse [62] push service. The successful attack on <http://afn.az> changed a variable that holds <https://pushdata.sendpulse.com:4434/> to have our HTTPS domain. This change causes a victim to hand over their subscription objects via a POST HTTPS request to our server. Listing 3 shows a retrieved subscription object, delivered to our HTTPS server. We used this subscription object to send a phishing push message with the spoofed domain. Figure 11(b) shows our push notification with the spoofed domain of <https://afn.sendpulse.com>.

```
{ action: 'subscription',
  subscriptionId: 'f4_4m0ef9gY:APA91bHeQyj0vtsV ... ',
  appkey: '5b0b85c4dd9d4ded16c73d9436fa494e',
  browser: { name: 'Chrome', version: '65' },
  lang: 'en',
  url: 'http://afn.az/',
  sPubKey: 'BOMfTTU/13bEPy1FXf ... ',
  sAuthKey: '8moW+qAXsAKjs0BR3F ... ',
  sPushHostHash: '7c977009d5861eebb711656eb7d87a74' }
```

Listing 3: A subscription object delivered due to the spoofed destination URL in a target library

Table 4 summarizes the feasibility of our attacks against eight third-party push HTTP services. The four libraries fetched their

Library	# of Affected HTTP Sites	VAPID	Push Permission Delegation	Domain Name Spoofing	
				Subscription over HTTP	Subscription over HTTPS
OneSignal	528	✓	×	×	×
SendPulse	93	×	✓	×	✓
Pushcrew	31	×	×	×	×
Pushengage	19	×	×	×	×
Izooto	18	×	✓	✓	✓
Pushwoosh	4	×	✓	×	✓
Urbanairship	2	✓	×	×	×
Foxxpush	1	×	✓	×	×

Table 4: Feasibility of push permission delegation and domain spoofing attacks across third-party HTTP push services

script over HTTP, which makes the websites with these libraries are vulnerable to push permission delegation attack. Also, little or no effort has been committed to protecting the secrecy of a subscription object (which is the Izooto case). Even transmitting a subscription object over HTTPS is not enough to protect users from phishing via push messages with spoofed domains as shown in Figure 11(b). The VAPID protocol blocks the domain spoofing attacks. However, only two vendors place the VAPID protocol, which exposes visitors on 166 HTTP websites to push domain spoofing attacks. The domain spoofing attack is critical. In the perspective of a push message recipient, there is no way of knowing that the message actually comes from the attacker because the push notification shows its valid domain. We recommend several mitigation to address the push attacks in Section 10.

9.3 Side-channel Attack on Browsing History via Cache

We implemented a new side-channel attack in which a PWA attacker can learn the PWA browsing history of a victim. As explained in Section 7, the attack code loads a target PWA website within an iframe on the attacker-controlled page, then checks the *onload* event callback corresponding to the target iframe is called.

To check the feasibility of our attack against various browsers, we experimented the side-channel attack against the Chrome, Firefox, Safari, UC Browser, Edge, Internet Explorer, and Opera browsers. We confirmed that our side-channel attack is effective on Mozilla Firefox 59.0.2 and Safari 11.1. Fortunately, unlike two vulnerable browsers, all other browsers invoke their *onload* event handlers regardless of whether loading a target PWA is successful or not.

Offline Cache Attack		# of Websites (% Percentage)
Vulnerable		187 (36.5%)
Not Vulnerable	Frame Busting	10 (1.9%)
	CSP	22 (4.3%)
	Corrupted Content	20 (3.9%)
	X-Frame-Options	132 (25.7%)
	Bad Cache	142 (27.7%)
Total		513 (100%)

Table 5: A Feasibility of a side-channel attack using the cache on PWAs in the wild

Against the 513 collected PWAs that use the cache (see Section 3), we conducted the side-channel attacks on inferring visited PWAs.

As Table 5 shows, 187 (36.5%) PWAs were identifiable by the side-channel attack. The attack did not work for 164 PWAs (31.9%) because of their frame busting techniques (10 PWAs), Content Security Policy [28] including the *frame-ancestors* [47] directive (22 PWAs) and X-Frame-Options header [58] (132 PWAs).

We further analyzed the categories of the 211 PWAs vulnerable to our side-channel attack. We include Table 6 in the Appendix. The privacy-sensitive categories including Education, Hobbies & Interests, Personal Finance, and Adult Contents contain 104 real-world PWAs, of which victims wish to keep private.

10 DEFENSE

In this section, we propose defenses and recommendations to mitigate the security and privacy risks of PWAs addressed earlier. We suggest practical defenses for third-party push library providers and PWA developers to act on immediately, while recommending a guideline for PWA users.

To library providers. Third-party push library providers should manage sensitive push subscription information with care, not to leak such information by any means. A simple but powerful defense against a push domain spoofing attack (see Section 6.2) is to place the VAPID protocol. The VAPID protocol prevents any unauthenticated entity from sending a push message to a browser push service.

Another defense to block leaking subscript objects via reflected channels is to prevent a network attacker from modifying the library script. HSTS [46] header can achieve this by enforcing a JS library to be delivered over HTTPS. We observed that OneSignal, Urbanairship, Pushcrew and Pushengage set up HSTS, providing a safer service than others.

We believe that the current practice of obtaining the push permission from a redirected website is unhealthy (see Section 6.1). Unless a user is aware of the explicit relation between his visited website and its redirected website, a user is compelled to grant push permission based on the redirection, and not on the explicit domain name.

Note that push notification is designed to support only HTTPS websites. Third-party push vendors have expanded their services to HTTP websites by blindly asking a user to grant push permission for a redirected website domain. This brings unfortunate consequences such that a user makes a permission granting decision based on the redirection, which is rooted at an untrustworthy source, a HTTP website. A practical solution is that a user’s browser whitelists

certain third-party push service domains, and only allows the permission requests from their subdomains. The Chrome browser provides the `contentsettings.notifications` property for an extension to specify whether the listed domains are allowed to show any notifications [12]. To compute such a whitelist, users can reference the reputation of websites collected via social clouding such as Web-of-Trust [52] or Google Safe Browsing [27].

To PWA developers. The practical defense against our history sniffing attack via cache (see Section 7) is to prevent being framed by cross-domain websites. Stock *et al.* demonstrated that X-Frame-Options adopted 53% of the Alexa top 500 sites in 2016 [66], which demonstrates the security awareness on the prevention of being framed. However, X-Frame-Options, CSP, frame busting techniques are known for blocking Clickjacking attack [37], not the side-channel attack on PWAs. We thus recommend PWA developers to actively place the *frame-ancestors* directive of CSP or X-Frame-Options header that prevents the websites from being framed by other PWAs.

Applying HTTPS is a powerful defense against our push attacks as shown in Section 6. Developers should fetch their third-party library scripts and send subscription objects over secure channels so that any network attacker cannot interfere with them. The recent dedications of security communities toward secure Web have been helping seamless migrations into HTTPS websites [16, 24]. We believe that applying HTTPS has become easier and cheaper on the modern Web.

To users. Users should be aware of the phishing risk incurred by push notifications. Because push domain spoofing and push permission delegation attacks are feasible as a consequence of security flaws in third-party push libraries, users should carefully check the domain appeared in a push notification and a push permission granting dialog.

Several previous research suggested interesting ideas applicable for mitigating our attacks. As D. Florencio *et al.* [23] proposed, users may choose a trustworthy auditing service and send their push messages to this service. This auditing service aggregates phishing push messages from different users and informs users and phishing target websites on any suspicious activities. To address the cryptocurrency mining attack in Section 8.1, monitoring of fine-grained browser behaviors [71] can identify abnormal resource consumption from a specific website and its service worker.

More practically, we recommend to regularly check the browser settings to unregister unnecessary service workers who can be abused for performing arbitrary computations. Also, cleaning the cache frequently can be an effective defense to protect the side-channel attack as shown in Section 7.

11 RELATED WORK

To the best of our knowledge, no research has analyzed the security and privacy risks of PWAs. Several studies focused on inspecting the usability and efficiency of PWA features across different environments [4, 40–42, 65]. T. Steiner [65] examined Web Views support on PWA features in Android and iOS, different from stand-alone browsers. The author evaluated feature supports across different devices and operations systems. I. Malavolta *et al.* [42] assessed the impact of service workers on the energy efficiency of PWAs and

demonstrated the energy efficiency of these entities on selected mobile devices. Our work offers a better understanding of the security and privacy risks brought by PWAs.

Phishing and push related attacks. Phishing has been one of the most serious security problems for decades [10, 20, 33, 35, 36, 45, 70, 79]. Phishing attacks share a basic form in which the attacker crafts a fake website that mimics the appearance of an authentic website. Due to its effectiveness and technical simplicity of conducting these attacks, phishing attackers utilize a tool to develop phishing sites for numerous phishing campaigns. Such a phishing tool is called a *phishing kit*.

Numerous studies have investigated *phishing kits* [10, 33, 70]. M. Cova *et al.* [10] focused on analyzing various methods, used by phishing kits, while X. Han *et al.* [33] proposed sandboxing live phishing kits to completely protect the privacy of victims. Thomas *et al.* [70] showed that 12.4 million people are potential victims of phishing kits, and 1.9 billion usernames and passwords are exposed via data breaches.

In a similar, but different context, phishing attacks using customized push notifications on mobile devices was studied [78], but not on Web push notifications. The authors have shown that abusing the notification customization may allow installing a Trojan application to launch phishing attacks or to anonymously post spam notifications. On the other hand, a secure Web push system was suggested by G. Saride *et al.* [60]. The authors strengthen the authenticity of web push messages with additional components between content providers and applications. However, we note that our push attacks which derived from the careless implementation of Web push protocols still hold under their proposed system.

Side-channel leaks. Side-channel attacks have also posed a great threat to various Web applications [1, 6, 7, 22, 26, 38, 61]. Obtaining leaked sensitive information via a side-channel has been extensively studied. S. Chen *et al.* [7] took an advantage of the size distributions of transmitted packets to infer highly sensitive information (i.e. healthcare, taxation, web search queries), despite the presence of HTTPS protection. Another recent study [61] made use of packet burst patterns on encrypted video streams to fingerprint a video being streamed. P. Chapman *et al.* [6] proposed a way to measure the severity of information leakage in Web apps automatically.

On the other hand, using timing information has been a traditional mean of conducting side-channel attacks. It has been shown that the timing information of a user’s browser that exploits Web caching [50], allows revealing the browsing histories [22]. However, timing information can be error-prone due to unreliable page fetch latency affected by a number of error sources, such as network condition, web server loads, and client loads. Recently, exploiting cross-origin HTML5 AppCache was proposed [38], which allows identifying cross-origin resource statuses such as determining the login status of a victim browser. Similarly, T. Goethem *et al.* [26] showed that a web attacker can uncover users’ identification such as Twitter accounts by inspecting the cross-origin resource size stored in AppCache. Our work exploits the cache which is a new attack vector to uncover a victim’s browsing history on PWAs when the victim is offline.

12 CONCLUSIONS

We conducted the first study of analyzing the security and privacy risks of PWAs. We analyzed the phishing risk via push notification, identified security flaws in pervasive third-party push libraries, and proposed the new attacks of abusing cache and service worker. Our findings stem from the inherent PWA features that provide native-app like Web browsing experiences, which make the addressed risks unique to PWAs. We proposed our defense recommendations to enhance the safe use of PWAs in practice. We have also reported our findings to the corresponding vendors. Samsung, Firefox, and some of third-party push service providers involved in our attacks. We view the entire work in this paper as a step towards a better understanding of emerging native-app like features on Web applications and their security and privacy aspects.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their concrete feedback. We also appreciate our shepherd Ben Stock for guiding us in addressing the comments from the reviewers. This work was supported by National Research Foundation of Korea (NRF) Grant No.: 2017073934, IITP Grant No.: 2014-0-00065, and by the Naver corporation.

A PUSH MESSAGE DISPLAY DIFFERENCE

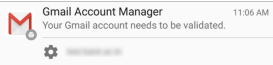
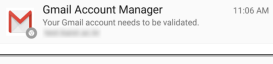
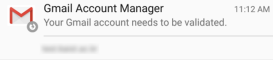
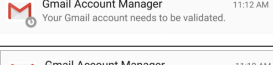
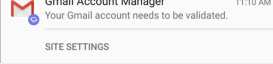
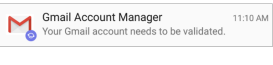
Browser	Notification Panel Condition	Web Push Notification
Chrome	Not busy	 11:06 AM
	busy	 11:06 AM
Firefox	Not busy	 11:12 AM
	busy	 11:12 AM
Samsung Internet	Not busy	 11:10 AM SITE SETTINGS
	busy	 11:10 AM

Figure 12: Push notifications on three different browsers on Android

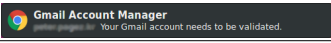
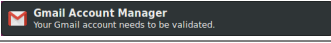
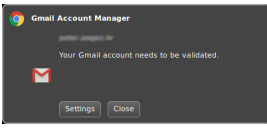
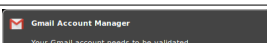
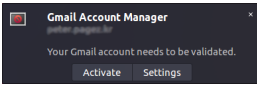
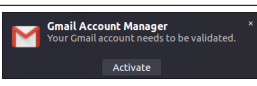
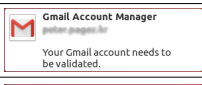
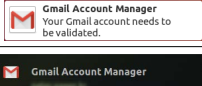
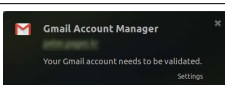
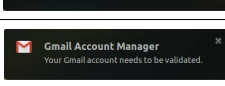
Desktop Environment	Browser	Web Push Notification
GNOME	Chrome	
	Firefox	
Cinnamon	Chrome	
	Firefox	
Budgie	Chrome	
	Firefox	
Pantheon	Chrome	
	Firefox	
MATE	Chrome	
	Firefox	

Figure 13: Push notifications on five different desktop environments

B WEB CATEGORIES OF PWAS VULNERABLE TO THE SIDE-CHANNEL ATTACK ON BROWSING HISTORY

Category	# of Websites
Technology & Computing	100
News / Weather / Information	71
Travel	45
Non-Standard Content	43
Arts & Entertainment	36
Hobbies & Interests	32
Personal Finance	29
Hotels	28
Shopping	37
Education	22
Food & Drink	20
Automotive	19
Society	17
Video & Computer Games	17
Business	17
File Sharing	16
Adult Content	15
Real Estate	14
Sports	14
Health & Fitness	14

Table 6: Top 20 categories of vulnerable PWAs

REFERENCES

- [1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. 2014. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. In *ACM Conference on Computer and Communications Security*. ACM.
- [2] Urban Airship. 2009. Retrieved April 25, 2018 from <https://www.urbanairship.com/>
- [3] A. Barth, C. Jackson, and J. Mitchell. 2008. Securing Frame Communications in Browsers. In *USENIX Security Symposium*. USENIX Association.
- [4] A. Biørn-Hansen, T. Majchrzak, and T. Grønli. 2017. Progressive Web Apps: the Possible Web-native Unifier for Mobile Development. In *International Conference on Web Information Systems and Technologies*.
- [5] Bugzilla. 2015. Iframe Onload Event Does Not Fire. Retrieved April 28, 2018 from https://bugzilla.mozilla.org/show_bug.cgi?id=444165
- [6] P. Chapman and D. Evans. 2011. Automated Black-box Detection of Side-channel Vulnerabilities in Web Applications. In *ACM Conference on Computer and Communications Security*. ACM.
- [7] S. Chen, R. Wang, X. Wang, and K. Zhang. 2010. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society.
- [8] Chromium. 2014. Javascript Iframe Onerror Event. Retrieved April 28, 2018 from <https://bugs.chromium.org/p/chromium/issues/detail?id=365457>
- [9] Coinhive. 2018. Coinhive - Monero JavaScript Mining. <https://coinhive.com/>
- [10] M. Cova, C. Kruegel, and G. Vigna. 2008. There is No Free Phish: An Analysis of "Free" and Live Phishing Kits. In *Proceedings of the Conference on USENIX Workshop on Offensive Technologies*. USENIX Association.
- [11] Apple Developer. 2016. Apple Certificates Support. Retrieved May 9, 2018 from <https://developer.apple.com/support/certificates/>
- [12] Chrome Developer. 2018. Chrome Extensions - Content Settings. Retrieved August 14, 2018 from <https://developer.chrome.com/extensions/contentSettings#type-ContentSetting>
- [13] Google Developers. 2016. AliExpress. Retrieved May 1, 2018 from <https://developers.google.com/web/showcase/2016/aliexpress>
- [14] Google Developers. 2016. Flipkart Triples Time-on-site with Progressive Web App. Retrieved May 1, 2018 from <https://developers.google.com/web/showcase/2016/flipkart>
- [15] Google Developers. 2016. Introduction to Progressive Web Apps. Retrieved May 9, 2018 from <https://codelabs.developers.google.com/pwa-dev-summit>
- [16] Google Developers. 2016. Mythbusting HTTPS. Retrieved April 25, 2018 from <http://www.codechannels.com/video/Chrome/chrome/mythbusting-https-progressive-web-app-summit-2016/>
- [17] Google Developers. 2018. Introduction to Push Notifications. Retrieved May 9, 2018 from <https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications>
- [18] Google Developers. 2018. PWA Case Studies. Retrieved April 26, 2018 from <https://developers.google.com/web/showcase>
- [19] Google Developers. 2018. Web Push Protocol. Retrieved May 9, 2018 from <https://developers.google.com/web/fundamentals/push-notifications/web-push-protocol>
- [20] R. Dhamija, J. Tygar, and M. Hearst. 2006. Why Phishing Works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.
- [21] Chromium Documents. 2018. Do Service Workers live forever? Retrieved August 14, 2018 from <https://github.com/chromium/chromium/blob/master/docs/security/service-worker-security-faq.md#do-service-workers-live-forever>
- [22] E. Felten and M. Schneider. 2000. Timing Attacks on Web Privacy. In *ACM Conference on Computer and Communications Security*. ACM.
- [23] D. Florencio and C. Herley. 2006. Password Rescue: A New Approach to Phishing Prevention. In *1st USENIX Workshop on Hot Topics in Security*. USENIX Association.
- [24] Linux Foundation. 2018. Let's Encrypt. Retrieved April 25, 2018 from <https://letsencrypt.org/>
- [25] FoxPush. 2016. Retrieved April 25, 2018 from <https://www.foxpush.com/>
- [26] TV. Goethem, M. Vanhoef, F. Piessens, and W. Joosen. 2016. Request and Conquer: Exposing Cross-Origin Resource Size. In *USENIX Security Symposium*. USENIX Association.
- [27] Google. 2018. Google Safe Browsing. Retrieved August 11, 2018 from <https://developers.google.com/safe-browsing/>
- [28] W3C Groups. 2016. Content Security Policy Level 3. Retrieved May 9, 2018 from <https://www.w3.org/TR/CSP3/>
- [29] W3C Groups. 2017. Web Workers. Retrieved April 24, 2017 from <https://w3c.github.io/workers/>
- [30] W3C Groups. 2018. Push API. Retrieved May 9, 2018 from <https://w3c.github.io/push-api/>
- [31] W3C Groups. 2018. Service Workers Nightly. Retrieved April 24, 2018 from <https://w3c.github.io/ServiceWorker/>
- [32] W3C Groups. 2018. the Notification API. Retrieved May 7, 2018 from <https://notifications.spec.whatwg.org/>
- [33] X. Han, N. Kheir, and D. Balzarotti. 2016. PhishEye: Live Monitoring of Sandboxed Phishing Kits. In *ACM Conference on Computer and Communications Security*. ACM.
- [34] Izooto. 2016. Retrieved April 25, 2018 from <https://www.izooto.com/>
- [35] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer. 2007. Social Phishing. *Commun. ACM* (2007).
- [36] M. Jakobsson and S. Myers. [n. d.]. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft*. Wiley-Interscience.
- [37] Huang L, A. Moshchuk, H. J. Wang, S. Schechter, and C. Jackson. 2012. Clickjacking: Attacks and Defenses. In *USENIX Security Symposium*. USENIX Association.
- [38] S. Lee, H. Kim, and J. Kim. 2015. Identifying Cross-origin Resource Status using Application Cache. In *Proceedings of the Annual Network and Distributed System Security Symposium*.
- [39] T. Lee. 2017. How Bitcoins Became Worth \$10,000. Retrieved May 9, 2017 from <https://arstechnica.com/tech-policy/2017/11/how-bitcoins-became-worth-10000/>
- [40] T. Majchrzak, A. Biørn-Hansen, and T. Grønli. 2018. Progressive Web Apps: the Definite Approach to Cross-Platform Development?. In *Hawaii International Conference on System Sciences*.
- [41] I. Malavolta. 2016. Beyond Native Apps: Web Technologies to the Rescue! (Keynote). In *Proceedings of the 1st International Workshop on Mobile Development*. ACM.
- [42] I. Malavolta, G. Procaccianti, P. Noorland, and P. Vukmirovic. 2017. Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps. In *International Conference on Mobile Software Engineering and Systems*.
- [43] R. McPherson, S. Jana, and V. Shmatikov. 2015. No Escape From Reality: Security and Privacy of Augmented Reality Browsers. In *International World Wide Web Conference*.
- [44] mitmproxy. 2018. Retrieved April 25, 2018 from <https://mitmproxy.org/>
- [45] T. Moore and R. Clayton. 2012. Discovering Phishing Dropboxes using Email Metadata. In *eCrime Researchers Summit*.
- [46] Mozilla Developer Network. 2016. HSTS - Strict Transport Security. Retrieved April 25, 2018 from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security>
- [47] Mozilla Developer Network. 2017. CSP: frame-ancestors - HTTP. Retrieved May 9, 2018 from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors>

- [48] Mozilla Developer Network. 2018. AppCache is deprecated. Retrieved May 9, 2018 from https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache
- [49] Mozilla Developer Network. 2018. Cache - Web APIs. Retrieved May 9, 2018 from <https://developer.mozilla.org/en-US/docs/Web/API/Cache>
- [50] Mozilla Developer Network. 2018. HTTP caching. Retrieved April 25, 2018 from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>
- [51] Mozilla Developer Network. 2018. WebAssembly. Retrieved April 25, 2018 from <https://developer.mozilla.org/en-US/docs/WebAssembly>
- [52] Web of Trust. 2018. Web of Trust - Website reputation and review service. Retrieved August 11, 2018 from <https://www.mywot.com/>
- [53] OneSignal. 2018. Retrieved April 25, 2018 from <https://onesignal.com/>
- [54] pushcrew. 2015. Retrieved April 25, 2018 from <https://pushcrew.com/>
- [55] PushEngage. 2015. Retrieved April 25, 2018 from <https://www.pushengage.com/>
- [56] PushWoosh. 2018. Retrieved April 25, 2018 from <https://www.pushwoosh.com/>
- [57] A. Ramachandran and N. Feamster. 2006. Understanding the Network-level Behavior of Spammers. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM.
- [58] D. Ross. 2013. HTTP Header Field X-Frame-Options. Retrieved May 9, 2018 from <https://tools.ietf.org/html/rfc7034>
- [59] N.V. Saberhagen. 2013. CryptoNote v 2.0. Retrieved May 9, 2018 from <https://cryptonote.org/whitepaper.pdf>
- [60] G. Saride, J. Aaron, and J. Bose. 2016. Secure Web Push System. In *International Conference on Communication Systems and Networks*.
- [61] R. Schuster, V. Shmatikov, and E. Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security Symposium*. USENIX Association.
- [62] SendPulse. 2015. Retrieved April 25, 2018 from <https://sendpulse.com/>
- [63] D. Silver, S. Jana, E. Chen, C. Jackson, and D. Boneh. 2014. Password Managers: Attacks and Defenses. In *USENIX Security Symposium*. USENIX Association.
- [64] S. Son, D. Kim, and V. Shmatikov. 2010. What Mobile Ads Know About Mobile Users. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society.
- [65] T. Steiner. 2018. What is in a Web View? An Analysis of Progressive Web App Features When the Means of Web Access is not a Web Browser. In *International World Wide Web Conference*.
- [66] B. Stock, M. Johns, M. Steffens, , and M. Backes. 2017. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security. In *USENIX Security Symposium*. USENIX Association.
- [67] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna. 2011. The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-scale Spam Campaigns. In *Proceedings of the Conference on Large-scale Exploits and Emergent Threats*. USENIX Association.
- [68] Symantec. 2013. Elliptic Curve Cryptography Certificates Performance Analysis. Retrieved May 9, 2018 from https://www.websecurity.symantec.com/content/dam/websecurity/digitalassets/desktop/pdfs/whitepaper/Elliptic_Curve_Cryptography_ECC_WP_en_us.pdf
- [69] Monero.org Team. 2018. Introduction to Monero (XMR) Coins. Retrieved May 9, 2018 from <https://monero.org/>
- [70] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. 2017. Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials. In *ACM Conference on Computer and Communications Security*. ACM.
- [71] P. Vadrevu, J. Liu, B. Li, B. Rahbarinia, K. Lee, and R. Perdisci. 2017. Enabling Reconstruction of Attacks on Users via Efficient Browsing Snapshots. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society.
- [72] Z. Weinberg, E.Y. Chen, P.R. Jayaraman, and C. Jackson. 2011. I Still Know What You Visited Last Summer: Leaking Browsing History via User Interaction and Side Channel Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society.
- [73] T. Whalen and K. Inkpen. 2005. Gathering evidence: use of visual security cues in web browsers. In *Proceedings of the Graphics Interface*. ACM.
- [74] WHATWG. 2018. HTML Living Standard. Retrieved May 8, 2018 from <https://html.spec.whatwg.org/multipage/iframe-embed-object.html#the-iframe-element>
- [75] WHATWG. 2018. Offline Web Applications. Retrieved April 26, 2018 from <https://html.spec.whatwg.org/multipage/offline.html>
- [76] WHATWG. 2018. the WebSocket API. Retrieved May 7, 2018 from <https://html.spec.whatwg.org/multipage/web-sockets.html>
- [77] M. Wu, R.C. Miller, and S.L. Garfinkel. 2006. Do Security Toolbars Actually Prevent Phishing Attacks?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM.
- [78] Z. Xu and S. Zhu. 2012. Abusing Notification Services on Smartphones for Phishing and Spamming. In *Proceedings of the Conference on USENIX Workshop on Offensive Technologies*.
- [79] S. Zawoad, A. Dutta, A. Sprague, R. Hasan, J. Britt, and G. Warner. 2013. Phish-Net: Investigating Phish Clusters using Drop Email Addresses. In *APWG eCrime Researchers Summit*.