Inferring Browser Activity and Status Through Remote Monitoring of Storage Usage

Hyungsub Kim[*,†] Sangho Lee[§] Jong Kim[*]

[*] Department of Computer Science and Engineering, POSTECH, Republic of Korea
[†] Agency for Defense Development (ADD), Republic of Korea
[§] School of Computer Science, Georgia Institute of Technology, USA

hyungsubkim@postech.ac.kr sangho@gatech.edu jkim@postech.ac.kr

ABSTRACT

Web applications use the local storage of a web browser to temporarily store static resources for caching and persistently store personalized data for stateful services. Since different web applications use the local storage differently in terms of size and time, attackers can infer a user's browser activity and status if they can monitor storage usage: for example, which web site a user is viewing and whether a user has logged in to a certain web site. In this paper, we explore passive and active web attacks that exploit the Quota Management API to extract such information from a web browser, as the API allows us to continuously monitor the size of available storage space. We develop two web attacks: a cross-tab activity inference attack to passively monitor which web site a user is currently visiting and a browser status inference attack to actively identify the browser status such as browser history and login information. Our attacks are successful at stealing private information from Chrome running on various platforms with ∼90% accuracy. We further propose an effective solution against the attacks.

1. INTRODUCTION

Modern web applications heavily use the local storage of a client web browser to temporarily store static web resources (*e.g.*, browser cache) and persistently store data for stateful services (*e.g.*, cookie). Since many multimedia and text resources composing a web page usually do not frequently change, a web browser does not need to repeatedly download them for each visit to avoid unnecessary network traffic. Instead, the browser caches such resources into its local storage and loads them from there until the corresponding remote resources have changed. Also, a web browser needs to keep cookie information to maintain login status and other stateful information (*e.g.*, items in a shopping cart).

Moreover, HTML5 provides *offline storage APIs* as many people tend to use mobile devices to browse the Web, suffering from connection failures or slow connection speed. For example, the APIs allow a web application to specify resources to be cached in the local storage of a web browser (Application Cache [18] and Service Worker [36]), and request an amount of storage space (*quota*) and monitor *how much storage space it takes* (the Quota Management API [47]).

However, such heavy usage of the local storage of a web browser could be exploited to breach user privacy if attackers can monitor it, as different web pages consist of different resources such that their storage usage patterns are distinguishable. Researchers have discovered that network usage patterns to download web resources [5–7, 16, 17, 22, 24, 29, 33, 39, 43, 44] and memory usage patterns to load downloaded resources [20] can be used to infer which web site a victim browser visits. Since downloaded resources are temporarily stored into the local storage and then loaded into the memory, attackers have possibility to conduct similar attacks if they can monitor storage usage patterns.

Surprisingly, we identify the Quota Management API can be used to perform such attacks because it allows a web application to know *how much storage space remains in a user's local device* by querying the quota of *temporary storage*—a shared pool among web applications—without any user confirmation. When explaining the `quota` attribute of the `StorageInfo` interface, the specification states that "For temporary storage this value may reflect the actual storage space available on the user's local device and may change from time to time" [47]. Further, the Quota Management API is a JavaScript method, so attackers do not need to have access to the network [5–7, 16, 17, 22, 24, 29, 33, 39, 43, 44] or machine [20] to which a client web browser belongs.

In this paper, we explore how web attackers can exploit the Quota Management API as a *side channel* to infer sensitive information from web browsers. When visiting a web site, a web browser takes a portion of local disk space to cache the resources transferred from the web site. The *time-varying* amount of storage space to store such resources differs from each other, which is the unique feature of each web site. Therefore, by monitoring temporal changes in the quota of temporary storage (we call it *storage footprint*, see §2.2) through an attack web site, attackers can *remotely* determine the feature of a web site that a victim web browser visits. For example, Figure 1 shows temporal changes in the size of storage footprints when we visited the front pages of four popular web sites (google.com, facebook.com, yahoo.com, and youtube.com) by using Chromium 34 running on an Ubuntu 12.04 desktop. The temporal changes in the size of the storage footprints differ from each other, so we can distinguish web sites by comparing them.

We consider two attacks using storage footprints. The first attack, a *cross-tab activity inference attack*, allows an attack web site to identify the other web site a victim web browser is currently visiting via a different tab or window. While a victim web browser opens an attack web page in a browser tab or window, the web page *passively*

ACM ISBN .
DOI:

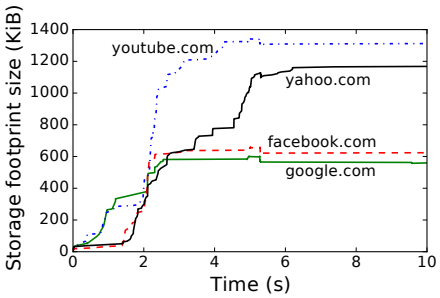


Figure 1: Temporal changes in the size of storage footprints belonging to the front pages of four popular web sites, obtained through the Quota Management API. Different web sites use the local storage differently in terms of size and time.

monitors temporal changes in the size of the victim web browser's storage footprints to identify which web site the victim web browser visits through a new tab or window. The accuracy of the cross-tab activity inference attack is up to 97.3% when a victim web browser is Chrome running on Linux (§3).

The second attack, a *browser status inference attack*, allows an attack web site to identify the *history* and *login status* of a victim web browser associated with a target web site. When accessing a web site visited in the past, a web browser occupies no or small additional disk space because the browser cache has already stored the web site's resources. By visiting a target web site while monitoring temporal size changes in the storage footprints, an attack web site can identify whether the browser has visited the web site before. The accuracy of our browser history stealing attack is up to 99% when a victim web browser is Chrome running on Android (§4.3).

Moreover, by using a similar technique, an attacker can recognize the login status of a victim web browser. Many web sites have privileged web pages only accessible by permitted users who belong to some groups, companies, universities, or societies. A web browser either cannot fully fetch the entire resources of the privileged web page or is redirected to login or error page if it has no permission to access the web page. By attempting to visit such a web page while monitoring storage usage, we can recognize whether a browser has a permission to access the web page (§4.6).

We consider a countermeasure against the explored attacks: providing coarse-grained quota information. We suggest *rounding quota size down* to the nearest multiple of several kilobyte or megabyte. Our countermeasure considerably decreases overall inference accuracy while introducing negligible overhead.

This work makes the following contributions:

- **Novel security problem.** To the best of our knowledge, this is the first study that handles the security problems of the Quota Management API, and considers a web side-channel attack exploiting temporal changes in the size of available storage space.
- **Novel web attack.** Our attack allows a remote web attacker to passively monitor the current activity of a victim web browser and to actively inspect the status of the browser.
- **Effective countermeasure.** We suggest an effective countermeasure against our attack: a round-down method. The countermeasure can substantially decrease attack accuracy even in an ideal scenario while only demanding minor modification of the Quota Management API.

The remainder of this paper is organized as follows. §2 explains background information of our work. §3 introduces our cross-tab activity attack. §4 introduces our browser status inference attack.

```
1 //Request storage usage and capacity left.
2 navigator.webkitTemporaryStorage.
3   queryUsageAndQuota(onSuccess, onError);
4
5 function onSuccess(usedSpace, remainingSpace) {
6   console.log("Used: " + usedSpace + ", remaining: " +
7     remainingSpace);
8 }
```

Figure 2: JavaScript pseudocode to monitor temporal changes in storage footprint size.

§5.1 discusses countermeasures against our attacks. §6 introduces related studies of our work. Lastly, §7 concludes this work.

2. BACKGROUND

In this section, we explain the browser cache, the Quota Management API, and an optimal subsequence bijection algorithm.

2.1 Web Browser Cache

Modern web browsers use the browser cache for reducing network traffic and load time of web pages. When a user first visits a web site, the user's web browser fetches the resources of the web site, stores them in the browser cache, and renders and displays them on a screen. Later, when the user visits the web site again, instead of fetching the resources again, the web browser loads cached resources from the browser cache if the web site does not change them.

We briefly explain resource types that Chrome caches in the local storage. Chrome stores (1) HTTP responses from a web site (*e.g.*, HTML and JavaScript code, images, CSS, and media files); (2) resources specified by a web application using AppCache or IndexedDB; (3) SSL sessions to skip round trips of SSL handshake [11]; and (4) GPU shaders to reduce GPU rendering time [40].

2.2 Quota Management API

The Quota Management API is proposed to manage and monitor the available storage space in a web browser to support other HTML5 storage APIs (*e.g.*, AppCache [18], ServiceWorker [36], and IndexedDB [31]). This API provides two types of storage space. The first one is *persistent storage* that enables a web site to store persistent data in a user's local storage. The second one is *temporary storage* that enables a web site to store temporary data in the local storage. In this paper, we focus on the temporary storage which allows an attacker to infer a user's secret information. Also, we consider Chrome because, up to now, only Chrome implements the Quota Management API. [1]

The temporary storage space, also known as the shared pool [14], is *freely accessible* by all web applications running on Chrome without any user confirmation. The temporary storage is approximately 50% of available storage space and each web application can use up to 20% of the temporary storage. Consequently, a web application can use up to 10% of remaining storage space.

Figure 2 describes JavaScript code that uses the Quota Management API to check the available temporary storage space. At Line 2, the script calls a `queryUsageAndQuota()` method to obtain the storage information of a web browser. The method has two parameters: success callback `onSuccess()` and error callback `onError()`. When the method successfully obtains the temporary storage information, it calls `onSuccess()`. Otherwise, it calls `onError()`. The success callback function has two arguments: `usedSpace` to inform how much storage space a web application occupies; `remainingSpace` to inform remaining temporary storage space.

[1] Opera uses Chrome's Blink engine, so it has the same problem.

Although the Quota Management API is useful, it has security problems because it gives the fine-grained storage space information of a web browser to a web application. By monitoring the temporal changes of the information, attackers can develop attack methods to reveal sensitive user information. We will explain the details of the attack methods in §3 and §4.

Note that the security problems considered in this paper are due to not the security bugs of Chrome but the problems of the Quota Management API specification. Therefore, other web browsers, such as Firefox, Internet Explorer, and Safari, are supposed to vulnerable to the security problems if their vendors implement the Quota Management API to meet the current specification.

2.3 Optimal Subsequence Bijection

We used an optimal subsequence bijection (OSB) algorithm [25, 32] to compare change logs of storage footprint size. The OSB algorithm is a well-known algorithm not only to measure the similarity between time-series, but also to effectively deal with noise elements of them. Whenever a victim web browser visits a web page, an attacker can obtain a change log of storage footprint size that reflects the changes in available storage space size. However, each instance of the obtained change logs slightly differs from each other in terms of length and size due to background disk activity (§3.3.3) and network latency (§3.3.4). Unlike a dynamic time warping (DTW) algorithm [38], another well-known algorithm to compare time-series, the OSB algorithm can skip outliers of query and target time-series, making the inference accuracy of our attacks better. We compared the OSB algorithm with the DTW algorithm in our evaluation settings (§3); the OSB algorithm always showed better accuracy than the DTW algorithm. Consequently, we decided to use the OSB algorithm for our attacks.

3. CROSS-TAB ACTIVITY INFERENCE

In this section, we explain assumptions we have made and a passive attack to infer a user's current browsing activity in a different tab or window, called as a *cross-tab activity inference attack*.

3.1 Threat Model

We assume an attacker who prepares an attack web site to deceive visitors and runs attack scripts in a background tab. The attack web site could be a compromised web site, a phishing web site, or a semi-honest web site that tracks its users browsing patterns. The attacker's goal is to know the very next web site the visitor will visit to know detailed information of the visitor in real time, which can be used to perform targeted attacks (*e.g.*, spear phishing and personalized advertisement). The attacker does not compromise the network or machine a visitor is currently using, so the attacker cannot analyze network traffic [5–7, 16, 17, 22, 24, 29, 33, 39, 43, 44] and monitor local resource usage [8, 20, 27] to achieve the goal. Further, a visitor's browser has no well-known vulnerability such that the attacker cannot break the same origin policy.

3.2 Attack Procedure

We explain the procedure of our cross-tab activity inference attack that exploits storage footprints with Figure 3.

1. A victim opens a browser tab to visit an attack site Attack.com to view or download interesting contents.
2. Attack.com deceives the victim into keeping the tab open and initiates a monitoring script (Figure 4). For example, many web sites display countdown timers to expose advertisements for a long time before delivering actual contents, such as one-click hosting sites (*e.g.*, `rapidshare.com`) and URL shortening services (*e.g.*, `adf.ly`).

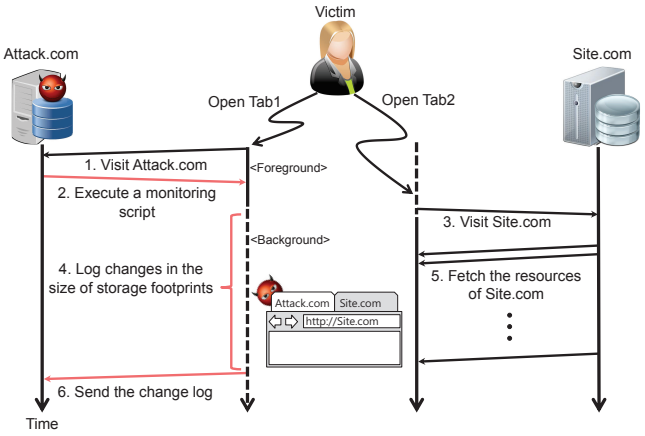


Figure 3: The procedure of a cross-tab activity inference attack using storage footprints. We assume that a victim visits Site.com via the front tab while maintaining a background tab for Attack.com.

```
1  document.addEventListener('visibilitychange', function(event) {
2    if (document.hidden) {
3      // This web page is hidden. Initiate monitoring.
4      navigator.webkitTemporaryStorage.
5        queryUsageAndQuota(onSuccess, onError);
6    }
7  });
8
9  function onSuccess(usedSpace, remainingSpace) {
10   // Post "remainingSpace" and current time to an attack server.
11   navigator.webkitTemporaryStorage.
12     queryUsageAndQuota(onSuccess, onError);
13 }
```

Figure 4: Pseudocode to infer sensitive user information with the Quota Management API.

3. The victim opens a new browser tab or window to visit another web site Site.com.
4. When the monitoring script recognizes that the tab of Attack.com becomes a background tab, it starts to continuously record changes in storage footprint size.
5. The victim web browser fetches the resources of Site.com and stores them in the browser cache.
6. The script sends the change log of storage footprint size to Attack.com. The attacker can infer that the web page the victim has visited is Site.com by comparing the change log of storage footprint size with the database.

Figure 4 describes a monitoring script to record a change log of storage footprint size. At Lines 1–7, the script declares an event handler of the Page Visibility API [30] to recognize whether a victim visits another web site or stays in an attack web site. When the victim visits another web site via a new tab, the tab of the attack web site becomes invisible and the script starts to recursively call `queryStorageState()` to log the changes in the size of storage footprints.

3.3 Identifying Non-cached Web Sites

We first consider the cross-tab activity inference attack against web sites a victim web browser has not recently visited (*i.e.*, not cached or cold). Assuming non-cached web sites increases the inference accuracy of our attack because a web browser has to fetch their entire resources and store them into the local storage.

3.3.1 Data collection

We prepared attack databases by collecting storage footprints of candidate web sites a victim web browser highly likely visits on

	Linux	Windows	Android
OS	Ubuntu 12.04	Windows 7	Android 4.0
Web browser	Chromium 34	Chrome 34	Chrome 34

Table 1: Experiment environment.

Algorithm 1 Matching algorithm

Input: A storage footprint database D, a victim's storage footprint f_v
Output: the inferred web page

```
d_min ← ∞    // the minimum distance
p_c ← none   // a candidate page
for each page p in D do
    for each storage footprint f_p of p in D do
        d = OSB(f_v, f_p)    // compute a distance
        if d < d_min then
            d_min ← d
            p_c ← p
return p_c
```

three different target platforms with Linux, Windows, and Android (Table 1). The data collection procedure for each front page of Alexa Top 100 web sites is as follows. First, using a Chrome web browser, we visit our attack page and then open one of the front page via a new tab. Second, we monitor temporal changes in storage footprint size for one minute (because we cannot know for sure when page loading finishes.) Third, we send the change log to our attack server. Lastly, we clear the browser cache for later experiments. We repeat this procedure 10 times for each front page on each platform and regard the 1000 change logs of storage footprint size per each platform as attack databases. The average size of storage footprints is approximately 3 KiB. Note that we use Chrome's default browser settings when collecting data, namely, we do not modify any settings such as cache size and privacy settings.

3.3.2 *Inference accuracy*

To evaluate the inference accuracy of our cross-tab activity inference attack against non-cached web sites, we visited each front page of Alexa Top 100 web sites 10 times on each platform, and compared their storage footprints with the attack databases by using Algorithm 1. As shown in Figure 5, Linux achieves the highest inference accuracy among the three platforms (97.3%) whereas Windows achieves the lowest inference accuracy (86.3%). We presume that background disk activity (§3.3.3) and wireless network (§3.3.4) make Windows and Android have lower inference accuracy than that of Linux, respectively.

3.3.3 *Background disk activity*

We anticipate that the lower inference accuracy of our attack on Windows than on Linux and Android is due to frequent background disk activity of Windows. Frequent disk activity (specifically, writing activity) can spoil our cross-tab activity inference attack because it monitors available storage space size. We measured background disk activity of the three operating systems by monitoring their storage footprints and compared them in terms of idle period where no changes in storage footprint size are observed. Figure 6 shows idle period statistics of the three operating systems. The average idle periods of Android, Linux, and Windows are 67 s, 22.5 s, and 1.5 s, respectively. From these results, we conclude that change logs of storage footprint size derived from Windows contain a large amount of noise.

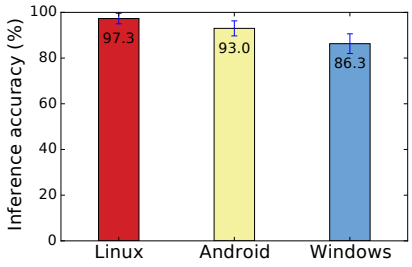


Figure 5: Inference accuracy of cross-tab activity inference attacks against victim web browsers that visit the front pages of Alexa Top 100 sites on Linux (LAN), Android (Wi-Fi), and Windows (LAN). Error bars represent 95% confidence intervals. An attack against Linux showed the best accuracy.

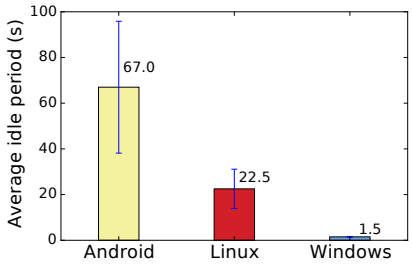


Figure 6: Statistics of idle periods that the size of storage footprints maintains in each OS during one hour. Windows suffered from frequent background disk activity.

To exclude the effects of background disk activity, we changed the location of the Chrome browser cache to a separate disk and evaluated the inference accuracy of cross-tab activity inference attacks against Alexa Top 100 web sites. We used a Chrome command line switch `-disk-cache-dir` for this experiment. We observe that the inference accuracy on Windows increases by 1.06× when we use the separate disk cache whereas Linux has no benefit (Figure 7). Therefore, the low inference accuracy of cross-tab activity inference attacks on Windows is due to heavy background disk activity.

3.3.4 *Wireless network*

We think the reason of the lower inference accuracy of our attack on Android than Linux is different network condition: Wi-Fi versus LAN. Network condition can affect the inference accuracy of our attack on Android because it usually uses Wi-Fi or cellular network whose network latency is less stable than that of a wired LAN. To analyze how network condition affects the inference accuracy of our cross-tab activity inference attack, we conducted our attack on Wi-Fi and LAN. Figure 8 shows the inference accuracy of our attack when visiting the front web pages of Alexa Top 100 web sites. The inference accuracy of our attack on Wi-Fi are 1.03×–1.05× lower than on LAN. Thus, we conclude that the bad network condition makes the inference accuracy on Android worse than that on Linux.

3.3.5 *Early inference*

The evaluation on the cross-tab activity inference attack explained so far has a shortcoming: it monitors changes in storage footprint size for one minute, but attackers cannot guarantee that a victim stays in a web page for more than one minute. To analyze how fast our attack can infer a web page visited by a victim, we varied monitoring time from 3 s to 60 s when attacking Alexa Top 100 web sites. As shown in Figure 9, when we monitor storage footprint size changes for approximately 5 s on Linux and Windows and for approximately 10 s on Android, the inference accuracy of our attack

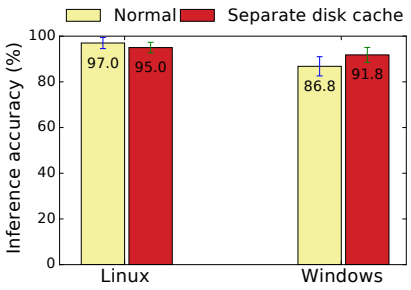


Figure 7: Inference accuracy of cross-tab activity inference with separate disk cache to ignore background disk activity. The separate disk cache increased the inference accuracy on Windows; namely, Windows suffered from the background disk activity.

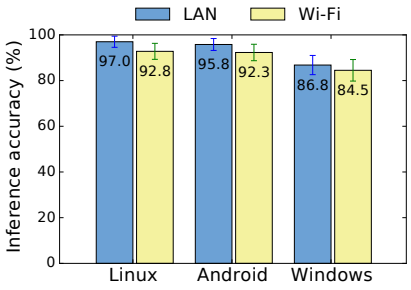


Figure 8: Accuracy of cross-tab activity inference attacks in wired and Wi-Fi networks. Using Wi-Fi slightly decreased the inference accuracy.

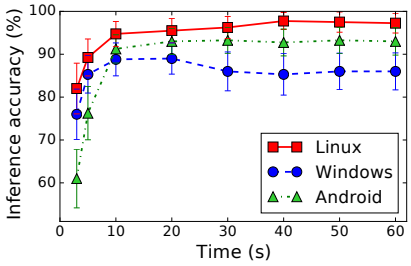


Figure 9: Accuracy of cross-tab activity inference attacks according to the length of monitoring time window. In Linux and Android, the inference accuracy became better as the length of monitoring time window increased, but, in Windows, the inference accuracy became worse due to the background disk activity.

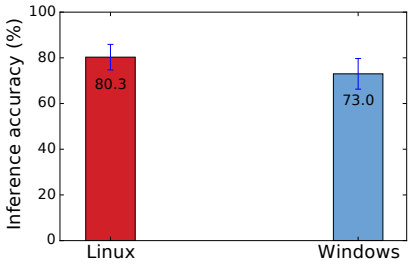


Figure 10: Accuracy of cross-tab activity inference attacks via Tor on Linux and Windows. Using Tor decreased the inference accuracy due to its geographical differences and network noise.

becomes around 90%. Thus, when conducting cross-tab activity inference attacks, attackers only need to assume a victim who stays in an attack page for more than 10 s.

Interestingly, on Windows, we observe the highest inference accuracy (89%) when monitoring storage footprint size changes for 20 s and decrease of inference accuracy as monitoring time increases (Figure 9). Windows has frequent background disk activity as explained in §3.3.3, so prolonging a monitoring period decreases the inference accuracy.

3.4 Identifying Web Sites Visited via Tor

Internet users can use an anonymity network (*e.g.*, Tor [42]) to protect their privacy. If a victim uses an anonymity network when visiting web sites, the accuracy of our cross-tab activity inference attacks would decrease because of long and unstable network latency.

To evaluate how an anonymity network affects the accuracy of our cross-tab activity inference attack, we conducted our attack against web sites visited through Tor. We adjusted the `MaxCircuitDirtiness` option of Tor to change a virtual circuit whenever we visit a web site. Figure 10 shows the inference accuracy of our attack when visiting the front web pages of Alexa Top 100 web sites. The inference accuracy of our attack on Linux and Windows is 80.3% and 73.0%, respectively.

We analyze why our cross-tab activity inference attack in a Tor network shows lower inference accuracy than that in a normal network, and figure out two reasons. First, Tor can change the geographical location of a victim web browser (an IP address belongs to a different country.) Many web sites customize their content according to the country information of visitors, so storage footprints of the browser can be completely changed even when it visits the same web site. To overcome it, an attacker should prepare a huge storage footprint database that covers a large number of countries. Second, we identify that a web browser on Tor occasionally cannot fetch all resources of a web page in one minute due to the long and unstable latency of Tor. An attacker needs to prolong monitoring time to mitigate the second problem.

3.5 Identifying Cached Web Sites

Next, we perform our cross-tab activity inference attacks against web sites that are stored in the browser cache (*i.e.*, warm). When a web browser visits a cached web site again, it only fetches and stores dynamic or updated resources of the web site. Thus, a change log of storage footprint size would contain restrictive information. We expect that the inference accuracy of our attack decreases when a victim visits cached web sites.

We prepared attack databases that contain change logs of storage footprint size when visiting cached Alexa Top 100 web sites, and compare the attack databases with a victim's visits for inference. Note that we did not test it with Tor because it usually does not cache web resources. We consecutively visited each web site 10 times for each platform while clearing the browser cache only before visiting a web site for the first time. Figure 11 shows the inference accuracy of our attack when a victim visits the front web pages of Alexa Top 100 web sites that are already cached. As expected, we observed decreased inference accuracy: 70.5% (Linux), 65.8% (Android), and 60% (Windows).

3.6 Summary

As shown in §3.3.3, §3.3.4, §3.4, and §3.5, background noise in disk and network, Tor, and heavy usage of the browser cache (*e.g.*, increasing the browser cache size) can mitigate our cross-tab activity inference attacks, but we still have a chance to infer a victim's activity. A robust and effective countermeasure against our attack will be introduced in §5.1.

4. BROWSER STATUS INFERENCE

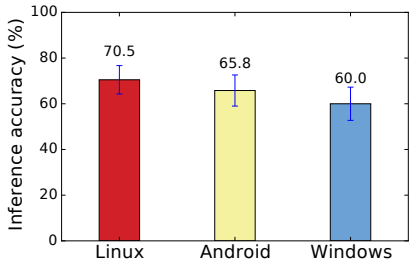


Figure 11: Accuracy of cross-tab activity inference attacks against cached web pages. The inference accuracy decreased because a browser did not fetch cached resources.

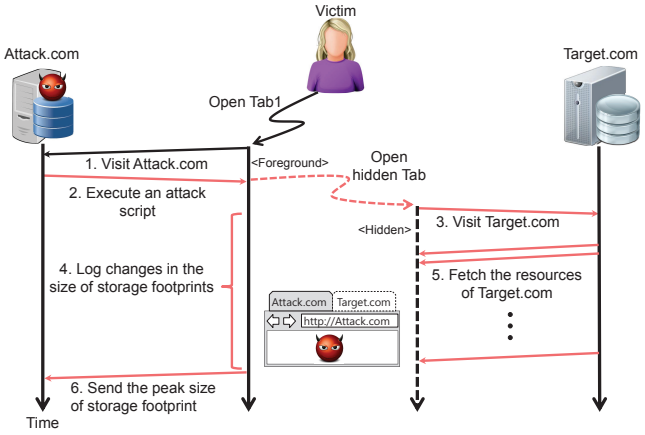


Figure 12: The procedure of a browser status inference attack using storage footprints. When a victim visits Attack.com, it creates a hidden tab to inspect browser status toward any web sites.

In this section, we explain how we can use storage footprints to develop two active attacks to infer browser status: *browser history stealing* and *login status identification*. Our browser status inference attacks resemble conventional cache timing attacks [4, 10, 19]. However, the accuracy of our attacks is better than the conventional attacks because our attacks use the total size of newly fetched resources that is not affected by network condition. In contrast, the cache timing attacks are highly vulnerable to network condition (see §4.5).

4.1 Threat Model

The threat model of the browser status inference attack is basically the same as §3.1, but an attacker has a slightly different goal: inferring browser history and login status. To obtain such information, the attacker generates additional requests to other web sites under consideration.

4.2 Attack Procedure

Figure 12 depicts the procedure of our browser status inference attack. This attack's procedure is almost the same as the procedure of our cross-tab activity inference attack except the methods to visit a target web site Target.com and to compare storage footprints differ. First, an attack script *directly loads* Target.com by using Prerendering [15], which allows a web application to preload a web page in a hidden browser tab. Second, an attacker uses *the peak size of storage footprints* instead of the change logs of storage footprint size to identify the status of a target web site.

4.2.1 Prefetching and prerendering

```
1  var PrerenderTimer;
2  var URL = ["google.com", "facebook.com", ...];
3  var idx = 0;
4
5  PrerenderTimer = setInterval("prerenderURL()",10000);
6
7  function prerenderURL() {
8    create_prerender(URL[idx]);
9    idx++;
10 }
11
12 function create_prerender(url) {
13   var link = document.getElementById('prerendering') ||
14     document.createElement('link');
15
16   link.id = 'prerendering';
17   link.rel = 'prerender';
18   link.href = url;
19
20   document.body.appendChild(link);
21 }
```

Figure 13: JavaScript pseudocode to prerender target web sites.

Modern web browser supports two functions, *prefetching* and *prerendering*, to reduce loading and processing time of web pages. First, prefetching makes a web browser preemptively download resources of a web page, but it only reduces network delay. A web site can prefetch a resource by using `<link rel="prefetch" href="[URL]">`.

Second, prerendering makes a web browser preload a web page in a hidden browser tab and display the loaded web page when a user attempts to visit the web page. Thus, prerendering can reduce network and rendering delay. A web site can preload a web page by using `<link rel="prerender" href="[URL]">`.

Figure 13 shows JavaScript pseudocode to prerender target web sites. At Line 2, an array `URL` contains a list of web pages to be inspected. At Line 5, the script defines `PrerenderTimer` to periodically call `prerenderURL()` (at Line 7). At Lines 7–10, the script prerenders a new web page every 10 s by dynamically inserting a `link` tag into an attack web page (at Lines 12–21).

Instead of prerendering, an attacker can use an `iframe` tag to load a web page. However, a web page can prevent an `iframe` tag from including itself by using either an HTTP header field `X-Frame-Options` [35] or *frame busting code* [37]. Therefore, we do not use an `iframe` tag when conducting our browser status inference attacks.

4.2.2 Peak size of storage footprints

We use the peak size of storage footprints to identify whether a victim web browser has visited a web site rather than OSB. When a victim visits a web site that he or she has visited before, many resources of the web site are already stored in the browser cache such that the victim web browser only downloads changed or dynamic resources of the web site. Definitely, the peak size of storage footprints in such a case is smaller than that when a victim firstly visits the web site. Using the peak size can avoid the computation overhead of OSB and shows enough accuracy as explained in the following sections.

4.3 History Stealing in a Normal Network

We first consider how our browser status inference attack successfully identifies browser history in a normal network.

4.3.1 Data collection

We collected the peak size of storage footprints of the *non-cached* front pages of Alexa Top 500 web sites on three different platforms (Linux, Windows, and Android) and treated them as attack databases. A data collection procedure for each front page of Alexa Top 500

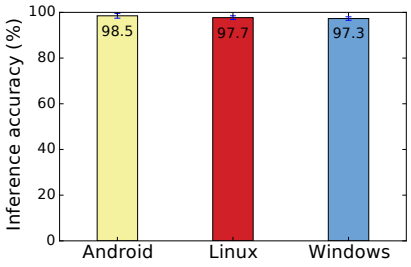


Figure 14: Inference accuracy of browser history stealing (Alexa Top 500 sites). We always observed high inference accuracy regardless of OSes.

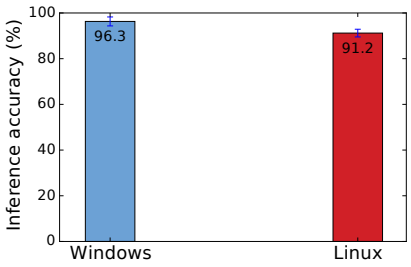


Figure 15: Inference accuracy of browser history stealing via Tor (Alexa Top 500 sites). The attack on Linux showed slightly worse accuracy due to frequent geographical changes in Tor.

web sites is as follows. First, using a Chrome web browser, we visit our attack page that uses Prerendering to visit the front page. Second, we monitor temporal changes in the size of storage footprints for 10 s. Third, we inform an attack server of the peak size of storage footprints. Lastly, we clear the browser cache. We repeated this procedure five times for each front page on each platform and regarded the 2,500 peak size values of storage footprints per each platform as attack databases.

4.3.2 *Inference accuracy*

To evaluate the inference accuracy of our history stealing attack, we visited each front page of Alexa Top 500 web sites five times on each platform and compared the peak size of storage footprints for each front page with that for each non-cached front page stored in the attack databases. Let μ_p and σ_p are mean and standard deviation of the peak size values of a web page p stored in the attack databases. Also, let s_p is the peak size of storage footprints when a victim web browser visits p. If $s_p < \mu_p - \sigma_p$, we treat that the victim web browser has visited p before. Otherwise, we treat that the victim web browser has not visited p. This treatment resembles the *distinguishability* defined in [20].

We identify that the inference accuracy of our history stealing attack is considerably high. As shown in Figure 14, the inference accuracy of our attack against the front web pages of Alexa Top 500 web sites on the three platforms is 98.5% (Android), 97.7% (Linux), and 97.3% (Windows).

4.4 History Stealing in a Tor Network

Next, we consider browser history stealing in a Tor network. Figure 15 shows results of browser history stealing for front pages of Alexa Top 500 web sites on Windows and Linux. Inference accuracy of our attack on Windows and Linux is 96.3% and 91.2%, respectively.

We analyze why the inference accuracy of our attack in a Tor network decreases and identify that it is due to changes of the geographical location of a victim web browser by Tor as explained in §3.4. When a web site provides different contents according to the country information of a web browser, the resources of the web site cached in the browser are no longer valid if the browser moves to a different country. The web browser needs to re-download the resources of the web site prepared for the country, so the peak size of storage footprint does not reduce.

```
1  function create_iframe() {
2      var webpage = document.createElement('iframe');
3      webpage.setAttribute('start', new Date().getTime());
4      webpage.src = URL[Url_count];
5
6      webpage.onload = function () {
7          var end = new Date().getTime();
8          var load = end - this.getAttribute('start');
9
10         // Send "load" to an attack server.
11
12         this.parentNode.removeChild(this);
13         ...
14         create_iframe();
15     }
16 }
```

Figure 16: Javascript pseudocode to measure load time of web pages to steal the browser history.

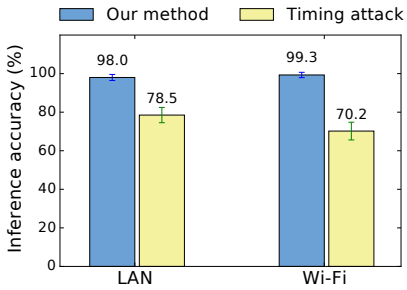


Figure 17: Inference accuracy of browser history stealing using storage footprints and using timing attacks on Linux (Alexa Top 200 sites). Our attack was better than the timing attacks.

4.5 Comparison with Timing Attack

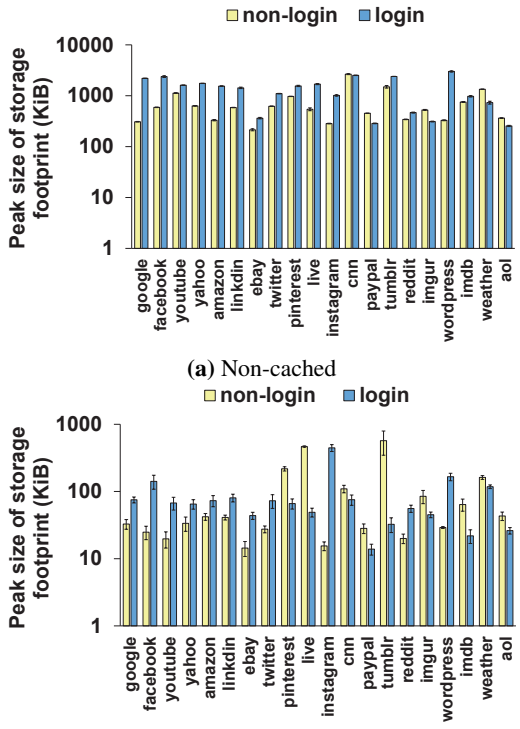
We compare history stealing using storage footprints with a conventional cache timing attack. Figure 16 shows JavaScript pseudocode to infer the browser history of a victim web browser by measuring page load time [21].

We evaluate the inference accuracy of the timing attack by using the front web pages of Alexa Top 200 web sites on Linux. The methods to collect data and to evaluate the inference accuracy are the same as those of §4.3 except that we measure page loading time for the timing attack instead of the peak size of storage footprints.

Figure 17 shows that the inference accuracy of our history stealing attack using storage footprints is 1.2× (LAN) and 1.4× (Wi-Fi) higher than that of the cache timing attack, respectively. Further, Figure 17 shows that the inference accuracy of our history stealing attack is not affected by network condition, whereas the inference accuracy of the timing attack slightly decreases when a victim web browser uses Wi-Fi.

4.6 Login Status Identification

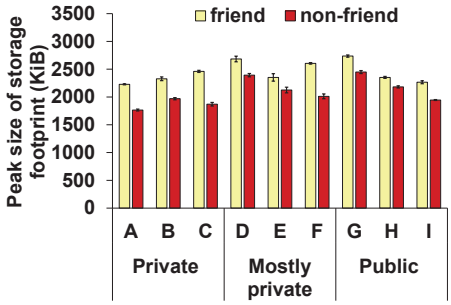
We explain an attack to infer the login status of a victim web browser by using the peak size of storage footprints. Many web sites have personalized web pages whose contents are changed or prohibited according to whether a web browser has login information. Thus, we expect that the peak size of storage footprints belonging to such web pages is also changed according to the login status.



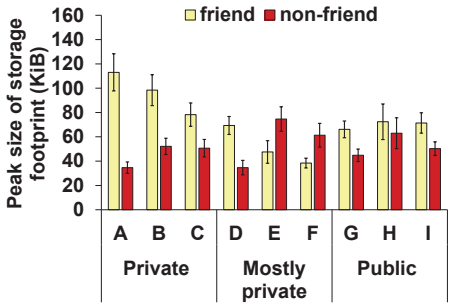
(a) Non-cached

(b) Cached

Figure 18: Differences in the peak size of storage footprints for 20 web pages according to the login status. The 20 web pages are the front web pages of web sites highly ranked at Alexa. In all cases, we were able to recognize login status.



(a) Non-cache



(b) Cached

Figure 19: Differences in the peak size of storage footprints when we visit the timeline of nine Facebook users (three private, three mostly private, and three public users) as a friend of the users or as not a friend of the users. Except H, we were able to recognize whether a victim was a friend of the Facebook users or not.

Inspecting whether a victim is logged in to web sites for specific people (*e.g.*, company, university, and society web sites) allows an attacker to infer the identity and preference of the victim.

We aim to identify the login status of a victim web browser for 20 popular web sites highly ranked in Alexa Top web sites that we already have accounts on them (details: Appendix §A).

We measured the peak size of storage footprints when visiting non-cached and cached web pages without and with login information 10 times on Linux, respectively. The results show that the difference between the peak size values of the same web pages without and with login information is high; their 95% confidence intervals do not overlap (Figure 18). Accordingly, we can distinguish whether a victim browser has login information of *all web sites checked*.

4.7 Friendship and Group Membership Identification on Facebook

We explain an attack to identify the friendship and group membership of a victim on Facebook, using the changed peak size of storage footprints for the same web page according to user permission. Facebook allows users to create private web pages that only permitted users can access. For example, if a Facebook user makes her timeline private, only her friends can view all contents on her timeline. Moreover, if a Facebook group manager makes the group secret or closed [9], only the group members can view the contents posted to the group page. When a Facebook user visits secret or closed group pages that the user does not belong to, the user receives an error page (secret) or a page that lists the group members with their profile photos (closed). In other words, according to user permission, a web browser that visits such web pages would store different resources in the local storage. Consequently, the peak size of storage footprints is also changed according to user permission.

4.7.1 Friendship

First, we attempt to identify the Facebook friendship of a victim web browser's user. We choose nine Facebook users who make their timelines private, mostly private, and public (three users for each type), and visit their timelines as a friend and not a friend 10 times on Linux, respectively. As shown in Figure 19, we can distinguish the peak size values of storage footprints for the same timelines visited as a friend and as not a friend except a cached public timeline H.

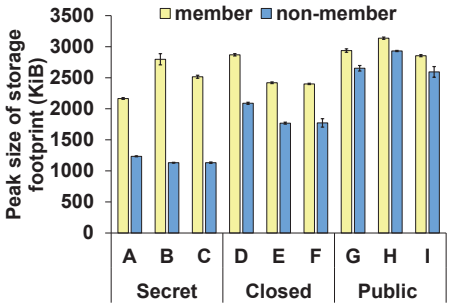
Interestingly, we observe difference between the peak size values of storage footprints for public Facebook timelines visited as a friend and as not a friend. When visiting a public Facebook timeline, a friend receives resources belonging a "Write something" box to post on the timeline whereas a non-friend receives resources belonging to an "Add Friend" box to be a friend. This makes the difference in the peak size values of storage footprints.

4.7.2 Group membership

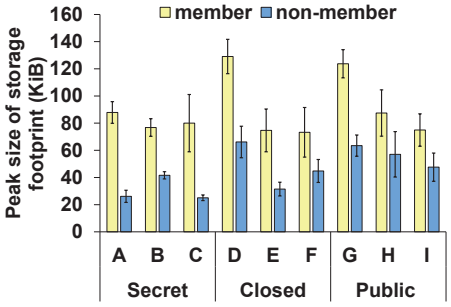
Next, we attempt to identify the Facebook group membership of a victim web browser's user. We choose nine Facebook groups: three secret, three closed, and three public groups, and visited the group pages as a member and as not a member 10 times on Linux, respectively. Figure 20 shows that we can distinguish the peak size values of storage footprints for the same groups visited as a member and not a member except a cached public group H.

5. DISCUSSION

5.1 Countermeasure: Round Down



(a) Non-cached



(b) Cached

Figure 20: Differences in the peak size of storage footprints when we visit nine Facebook groups (three secret, three closed, and three public groups) as a member of the groups or as not a member of the groups. Except H, we were able to recognize whether a victim was a member of the Facebook groups or not.

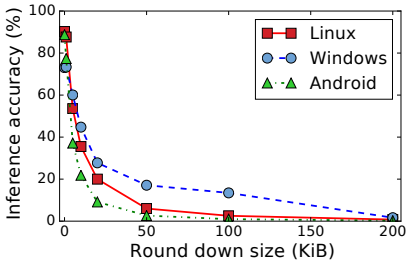


Figure 21: Accuracy of cross-tab activity inference attacks according to the size of round down (Alexa Top 500). The round down size of 200 KiB made the attack almost meaningless.

In this section, we explain a *round-down method* that can effectively mitigate our attacks. For example, when the remaining storage space of a victim web browser is 99.9 MiB and a web application queries the quota of temporary storage, the current Quota Management API informs the web application that the quota is 9.99 MiB. Instead of such an exact number, our proposal rounds the quota value down to the nearest multiple of a unit. For example, when we use 100 KiB as a unit of round down, the returned quota would be 9.9 MiB. Although our proposal can waste storage space according to the size of round down, it can effectively decrease the inference accuracy of our attacks as shown in the following evaluation results.

Cross-tab activity inference attack. Figure 21 shows that the accuracy of our cross-tab activity inference attack decreases as the round-down size increases. For example, when we round quota values down the nearest multiple of 20 KiB, the inference accuracy of our attack on Linux, Windows, and Android decreases to 20%, 27.75%, and 9.15%, respectively. Furthermore, when we round quota values down to the nearest multiple of 200 KiB, the inference accuracy of our attack on Linux, Windows, and Android decreases to 0.75%, 1.65%, and 0.25%, respectively.

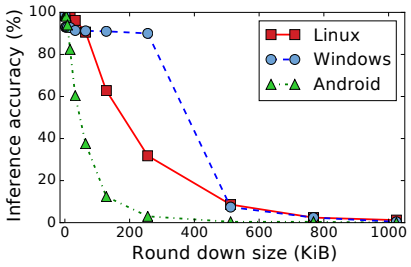


Figure 22: Inference accuracy of browser history stealing according to the size of round down (Alexa Top 500). The round down size of 1 MiB made the attack almost meaningless.

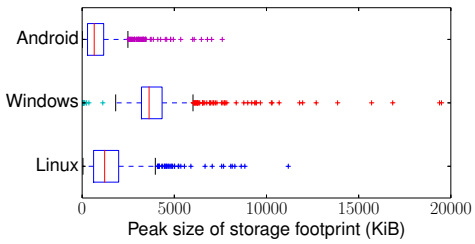


Figure 23: Statistics of the peak size of storage footprints belonging to the front pages of Alexa Top 1,000 sites.

Browser status inference attack. Figure 22 shows that the inference accuracy of our history stealing attack also decreases as the round-down size increases, but the size of round down is larger than that for our cross-tab activity inference attack (Figure 21). Unlike the cross-tab activity inference attack, the history stealing attack only uses the peak size of storage footprints, so minor changes in quota values cannot effectively decrease its inference accuracy. When we round quota values down to the nearest multiple of 256 KiB, the inference accuracy of our attack on Linux, Windows, and Android decreases to 31.8%, 90.0%, and 3.0%, respectively. Here, Windows still maintains good accuracy, since the average peak size of storage footprints on Windows is larger than 2.5 MiB unlike Linux and Android (Figure 23). However, when we round quota values down to the nearest multiple of 1 MiB, the inference accuracy of our attack on Linux, Windows, and Android decreases to 1.2%, 0.4%, and 0.0%, respectively.

Consequently, we suggest that the Quota Management API needs to round quota values down to the nearest multiple of 1 MiB to completely spoil our attack (the round-down size for login status identification is below 1 MiB, so we skip to explain it.) On average, our countermeasure will waste approximately 0.5 MiB of storage space for each web application that uses temporary storage.

5.2 Realistic Evaluation

In this paper we assume a less realistic evaluation environment which is too friendly to attackers: a victim visits one of Alexa Top 100 web sites (*closed world*), does not use multiple tabs to visits several web sites simultaneously, and does not generate significant background traffic (*e.g.*, downloading a huge file). Definitely, this is an unrealistic assumption as criticized by other researchers [22, 34] and the inference accuracy would decrease when we assume *open world*.

However, what we want to emphasize is that even if we do not assume a strong adversary who can monitor a victim's network traffic, our attack shows high inference accuracy comparable with previous attacks under the similar attacker-friendly evaluation settings [6]. Making this kind of attacks work well with a realistic environment is out of scope of this paper and several researchers have already considered it [16, 24, 33, 44].

Furthermore, our countermeasure (§5.1) successfully prevents all the attacks we explained even under such an attacker-friendly environment. This countermeasure certainly works well in the real world without any problem. Therefore, we believe whether we evaluate our attacks with realistic or unrealistic assumptions is not a critical problem.

6. RELATED WORK

In this section we introduce previous side channel attacks to identify browser activity and status.

CSS visited style. A CSS-based attack [2] uses style difference between visited and unvisited links to infer the history of a victim web browser. This attack can directly access the history information of a victim web browser, so it can accurately identify the browser history unlike other relatively inaccurate attacks (*e.g.*, attacks using network traffic or timing). Wondracek *et al.* [46] further extend this attack to infer the real identity of a victim by using the information about visited groups in OSNs. Baron [3] proposes an effective countermeasure that pretends all links are unvisited when a script attempts to inspect link styles. All major browsers adopt the countermeasure so that CSS-based attacks are no longer effective. To circumvent the solution, Weinberg *et al.* [45] use a webcam and user interaction, but it is difficult to be realized.

Timing information. A timing attack measures how long it takes to load web pages to infer a victim's browsing history and other private information. Felten and Schneider [10] firstly propose timing attacks using the web cache and DNS cache to infer the web pages that a victim web browser has recently visited. Bortz *et al.* [4] further reveal that an attacker can infer the login status of a victim and the number of items in a victim's shopping cart by measuring the loading time of web pages. Jia *et al.* [21] propose an advanced timing attack that measures the loading time of web pages containing location-sensitive contents to infer a victim's location information. In addition, researchers propose scriptless timing attacks using `meta-refresh` tag [1] and CSS [28]. Timing attacks, however, are error prone especially when network condition is bad. To circumvent the problem, Goethem *et al.* [12] propose new web-based timing attacks that can estimate the size of cross-origin resources regardless of network conditions.

Vulnerable API. Researchers consider side-channel attacks using vulnerable APIs of HTML5 and CSS as we exploited the Quota Management API. Kotcher *et al.* [23] propose two timing attacks using CSS filters. They infer the login status of a victim by measuring the frame rate of a web page through `requestAnimationFrame`. Further, they infer rendered pixels of a web page on the victim's screen by measuring the frame rate of each pixel. However, their attacks are slow and inaccurate. Tian *et al.* [41] identify that by using the Screen Sharing API, attackers can peek at a victim's screen and perform cross-site request forgery (CSRF) and history stealing attacks. This attack assumes a strong adversary who can obtain permission to use the Screen Sharing API from a victim and extract sensitive information from a video stream. Lee *et al.* [26] discover that by using the Application Cache API, attackers can check the status of cross-origin resources even without using client-side scripts. But, this attack does not work when target web sites make all of their content dynamic, as Facebook and Twitter do.

Concurrently and independently to our work, Goethem *et al.* [13] proposed attacks to infer the size of cross-origin resources using the ServiceWorker and Quota Management API. Although the side channel found by them and us are similar, we consider not only active attacks but also passive attacks to monitor victim user's real-time behavior without generating any additional resource requests to target web sites. In contrast, Goethem *et al.* mainly focus on active attacks to probe cross-origin resource size, which are similar with §4.6 and §4.7.

Network traffic. Numerous researchers [5–7, 16, 17, 22, 24, 29, 33, 39, 43, 44] identify that network traffic analysis allows attackers to infer which web site a victim is visiting even when the victim protects web traffic by using HTTPS or Tor. Various information of protected web traffic, such as packet timing, ordering, and size, can be used for inference. But, these attacks have limitations: they assume a strong adversary who can monitor network traffic and can attack victims in such a monitored network.

Local resource. Researchers consider side-channel attacks using a victim's local resources, such as memory, power-consumption, and GPU. Jana and Shmatikov [20] demonstrate that a malicious application can know which web site a victim web browser has visited by monitoring memory footprints of the browser through a `proc` file system. Clark *et al.* [8] measure the power consumption of a victim's machine to identify a visited web site. Michalevsky *et al.* [32] also analyze a smartphone's power usage to infer location of a mobile device. Lee *et al.* [27] analyze a GPU memory dump containing web page textures to recognize which web site a victim web browser has visited. The explained attacks, however, assume a strong adversary who can access the local resources of a victim web browser. An adversary needs to have a right to execute a process on the operating system on which a victim web browser is running [20, 27] or have access to the physical machine on which a victim web browser is running [8].

7. CONCLUSION

In this paper, we introduced a novel side-channel attack using the Quota Management API. Analyzing storage footprints obtained by using the Quota Management API allows a web attacker to identify which web site a victim web browser is currently visiting and the status of a victim web browser. We confirmed that the inference accuracy of our attacks is high: above 90% in many cases. To mitigate our attack, we suggested a round-down method that can substantially decrease the accuracy of our attacks while demanding minor modifications of the Quota Management API.

8. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their invaluable comments and suggestions. This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-TB1403-04.

References

[1] T. G. Abbott, K. J. Lai, M. R. Lieberman, and E. C. Price. Browser-based attacks on Tor. In *Privacy Enhancing Technologies Symposium (PETS)*, 2010.

[2] L. D. Baron. :visited support allow queries into global history. https://bugzilla.mozilla.org/show_bug.cgi?id=147777, 2002.

[3] L. D. Baron. Preventing attacks on a user's history through CSS :visited selectors. http://dbaron.org/mozilla/visited-privacy, 2010.

[4] A. Bortz, D. Boneh, and P. Nandy. Exposing private information by timing web applications. In *Proceedings of the 16th International World Wide Web Conference (WWW)*, Alberta, Canada, May 2007.

[5] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, Scottsdale, Arizona, Nov. 2014.

[6] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, Raleigh, NC, Oct. 2012.

[7] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proceedings of the 31th IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2010.

[8] S. S. Clark, H. Mustafa, B. Ransford, J. Sorber, K. Fu, and W. Xu. Current events: Identifying webpages by tapping the electrical outlet. In *European Symposium on Research in Computer Security (ESORICS)*, 2013.

[9] Facebook Help Center. What are the privacy options for groups? https://www.facebook.com/help/220336891328465.

[10] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS)*, Athens, Greece, Oct. 2000.

[11] T. Gentilcore. Chrome's 10 caches. http://gent.ilcore.com/2011/02/chromes-10-caches.html, 2011.

[12] T. V. Goethem, W. Joosen, and N. Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, Denver, Colorado, Oct. 2015.

[13] T. V. Goethem, M. Vanhoef, F. Piessens, and W. Joosen. Request and conquer: Exposing cross-origin resource size. In *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016.

[14] Google Developers. Managing HTML5 offline storage. https://developers.google.com/chrome/whitepapers/storage.

[15] Google Developers. Web developer's guide to prerendering in Chrome. https://developers.google.com/chrome/whitepapers/prerender.

[16] X. Gu, M. Yang, and J. Luo. A novel website fingerprinting attack against multi-tab browsing behavior. In *Proceedings of 19th IEEE International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2015.

[17] J. Hayes and G. Danezis. k-fingerprinting: a robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Security Symposium (Security)*, Austin, TX, Aug. 2016.

[18] I. Hickson. 7.7 offline web applications – HTML standard. http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html, 2014.

[19] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International World Wide Web Conference (WWW)*, Edinburgh, Scotland, May 2006.

[20] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2012.

[21] Y. Jia, X. Dong, Z. Liang, and P. Saxena. I know where you've been: Geo-inference attacks via the browser cache. In *Web 2.0 Security & Privacy (W2SP)*, 2014.

[22] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, Scottsdale, Arizona, Nov. 2014.

[23] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson. Cross-origin pixel stealing: Timing attacks using CSS filters. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, Oct. 2013.

[24] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *Proceedings of the 24th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2015.

[25] L. Latecki, Q. Wang, S. Koknar-Tezel, and V. Megalooikonomou. Optimal subsequence bijection. In *Proceedings of 7th IEEE International Conference on Data Mining (ICDM)*, 2007.

[26] S. Lee, H. Kim, and J. Kim. Identifying cross-origin resource status using application cache. In *Proceedings of the 2015 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2015.

[27] S. Lee, Y. Kim, J. Kim, and J. Kim. Stealing webpages rendered on your browser by exploiting GPU vulnerabilities. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2014.

[28] B. Liang, W. You, L. Liu, W. Shi, and M. Heiderich. Scriptless timing attacks on web browser privacy. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014.

[29] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS)*, Alexandria, VA, Oct.–Nov. 2006.

[30] J. Mann and A. Jain. Page visibility (second edition). http://www.w3.org/TR/page-visibility/, 2013.

[31] N. Mehta, J. Sicking, E. Graff, A. Popescu, J. Orlow, and J. Bell. Indexed database API. http://www.w3.org/TR/IndexedDB/, 2013.

[32] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly. PowerSpy: Location tracking using mobile device power analysis. In *Proceedings of the 24th USENIX Security Symposium (Security)*, Washington, DC, Aug. 2015.

[33] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel. Website fingerprinting at Internet scale. In *Proceedings of the 2016 Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2016.

[34] M. Perry. A critique of website traffic fingerprinting attacks. https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks, 2013.

[35] D. Ross and T. Gondrom. HTTP header field X-Frame-Options. RFC 7034, 2013.

[36] A. Russell, J. Song, and J. Archibald. Service Workers. http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html.

[37] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: A study of clickjacking vulnerabilities on popular sites. In *Web 2.0 Security & Privacy (W2SP)*, 2010.

[38] S. Salvador and P. Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580, 2007.

[39] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings of the 23rd IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2002.

[40] The Chromium Projects. GPU program caching. https://docs.google.com/a/chromium.org/document/d/1Vceem-nF4TCICoeGSh7OMXxfGuJEJYblGXRgN9V9hcE/edit.

[41] Y. Tian, Y.-C. Liu, A. Bhosale, L.-S. Huang, P. Tague, and C. Jackson. All your screens are belong to us: Attacks exploiting the HTML5 screen sharing API. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (Oakland)*, San Jose, CA, May 2014.

[42] Tor. Tor project. https://www.torproject.org.

[43] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium (Security)*, San Diego, CA, Aug. 2014.

[44] T. Wang and I. Goldberg. On realistically attacking Tor with website fingerprinting. Technical report, 2015.

[45] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2011.

[46] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel. A practical attack to de-anonymize social network users. In *Proceedings of the 31th IEEE Symposium on Security and Privacy (Oakland)*, Oakland, CA, May 2010.

[47] K. Yasuda. Quota management API. http://www.w3.org/TR/quota-api/, 2013.

APPENDIX

A URLs of identifying login status

Web site	URL
google.com	https://mail.google.com/mail/u/0/#inbox
facebook.com	https://www.facebook.com
youtube.com	http://www.youtube.com/feed/history
yahoo.com	https://us-mg5.mail.yahoo.com/neo/launch?reason=ignore&rs=1
amazon.com	https://www.amazon.com/gp/yourstore?ie=UTF8&ref_=gno_recs
linkdin.com	https://www.linkedin.com/profile/view?id=[ANONYMIZED]&trk=nav_responsive_tab_profile_pic
ebay.com	http://cart.payments.ebay.com/sc/view
twitter.com	https://twitter.com/following
pinterest.com	http://www.pinterest.com
live.com	https://blu184.mail.live.com/default.aspx?id=64855#fid=flin
instagram.com	https://instagram.com/accounts/edit
cnn.com	http://edition.cnn.com/profile/#mynewstop
paypal.com	https://www.paypal.com/webapps/customerprofile/summary.view
tumblr.com	https://www.tumblr.com/settings/blog
reddit.com	http://www.reddit.com/user/[ANONYMIZED]/hidden
imgur.com	http://imgur.com/account/settings/password
wordpress.com	https://[ANONYMIZED].wordpress.com/wp-admin/post.php?post=6&action=edit&message=6&postpost=v2
imdb.com	http://www.imdb.com/profile/recently-viewed?ref_=nv_usr_rvi_5
weather.com	https://profile.weather.com/#/profile/manage
aol.com	https://account.aol.com/account/settings/start

Table 2: URLs of identifying login status (20 popular web sites in Alexa).