I Know Where You've Been: Geo-Inference Attacks via the Browser Cache

Yaoqi Jia*, Xinshu Dong†, Zhenkai Liang *, Prateek Saxena*
*School of Computing, National University of Singapore
{jiayaoqi, liangzk, prateeks}@comp.nus.edu.sg
†Advanced Digital Sciences Center
xinshu.dong@adsc.com.sg

***Abstract*—Many websites customize their services according to different geo-locations of users, to provide more relevant content and better responsiveness, including Google, Craigslist, etc. Recently, mobile devices further allow web applications to directly read users' geo-location information from GPS sensors. However, if such websites leave location-sensitive content in the browser cache, other sites can sniff users' geo-locations by utilizing timing side-channels. In this paper, we demonstrate that such geo-location leakage channels are widely open in popular web applications today, including 62% of Alexa Top 100 websites. With *geo-inference attacks* that measure the timing of browser cache queries, we can locate users' countries, cities and neighborhoods in our case studies. We also discuss whether existing defenses can effectively prevent such attacks and additional support required for a better defense deployment.**

I. Introduction

Geo-location is a type of privacy-sensitive information. Websites have strong interests in obtaining users' geo-location information to provide personalized services and advertisements. On the other hand, web attackers [1] misuse victims' geo-locations for spear phishing, personally targeted advertisements, or even social engineering attacks. Geo-location leakage can cause tremendous damage to the user's privacy.

A traditional way for websites to identify users' locations is through IP addresses [2], [3]. However, inferring location from IP addresses is unreliable. First, IP address-based geo-location tracking is not accurate for mobile networks [4]. For example, one recent study shows that more than 90% of the mobile devices in Seattle can be associated with IP addresses that are located over 600 miles away from Seattle [4]. Second, users may intentionally use anonymization services, such as VPN [5] and Tor [6], to hide their real IP addresses [7].

Recent advancement in mobile devices enables websites to obtain geo-location information from GPS sensors. Nevertheless, modern browsers disable the access to geo-location information by default to protect user privacy. Mobile browsers require users' explicit permission to access GPS data. In this work, we show that web attackers can utilize side channels to infer the user's geo-location without the user's explicit permission.

Prior research has unravelled numerous privacy leakage side-channels via the browser cache [8]–[13]. Specifically, timing attacks on browser cache were introduced to sniff browsing history more than a decade ago [8]. Bortz et al. later deployed similar timing attacks on more web applications and scenarios [9]. We demonstrate how such timing side channels caused by browser caches can be utilized to identify a user's geo-location with high accuracy, without permission to access GPS sensors. We term such attacks *geo-location inference (or geo-inference) attacks*.

Our geo-inference attacks are based on a simple assumption that users usually visit location-oriented websites provided for their locations that they live in or plan to visit. For example, when visiting Google's main page, users will be automatically redirected to their specific country page of Google, e.g., *www.google.com.sg* in Singapore. As another example, many sites are meant to be accessed by local residents, such as local advertisement websites, e.g., *sfbay.craigslist.org* for San Francisco Bay Area users. Under this assumption, we conduct experiments on three popular websites, *Google*, *Craigslist* and *Google Maps*. We demonstrate that with geo-inference attacks via the browser cache, attackers can reliably infer a user's country, city, neighborhood, and even home address.

Our geo-inference attacks affect all mainstream browsers and a large fraction of popular websites on both desktop and mobile browsers. After running experiments on five mainstream browsers, we find that Chrome, Firefox, Safari, Opera, IE and even TorBrowser (version 3.5.2.1) [6], are all vulnerable to geo-inference attacks. Aided by a multi-country proxy service (virtual private network), we've browsed Alexa Top 100 websites in five different countries (USA, UK, Australia, Japan, and Singapore), and identified the sites that contain location-sensitive resources. Besides the three popular sites, we show that 62% of these sites contain location-sensitive resources and are susceptible to geo-inference attacks.

Security researchers have proposed various defense solutions against browser cache sniffing [14]–[16]. Jackson et al. propose a defense solution by segregating browser cache based on web origins, thus limiting web attackers from observing timing differences between accessing cached and non-cached cross-origin web resources [15]. However, none of today's mainstream browsers have adopted it so far; part of the reason is the significant performance overhead incurred by it. To validate this hypothesis, we implement the same-origin caching policy [15] on Chromium. We find that the policy triggers more than 50% performance overhead among Alexa Top 100 websites. In the context of geo-inference attacks, we believe that server-aided cache control, which involves the cooperation of both the server and the browser, is a more practical solution than cache segregation policy at only the browser side.

Contributions. In summary, we make the following contributions:

- We introduce geo-inference attacks that exploit geo-location information leakage channels via the browser cache on desktop and mobile platforms without direct access to GPS or IP address. With such geo-inference attacks, we can reliably track the geo-locations of web users to their countries, cities, and neighborhoods.
- We show that all five mainstream browsers (Chrome, Firefox, Safari, Opera and IE) on both desktop and mobile platforms as well as TorBrowser are vulnerable to geo-inference attacks. Meanwhile, 62% of Alexa Top 100 websites are susceptible to geo-inference attacks.
- We discuss existing and potential defenses and propose a more balanced solution to segregate location-sensitive resources instead of caching them in the user's browser.

II. Problem Definition

A. Threat Model

The adversary in a geo-inference attack is a standard *web attacker* [1], who controls at least one web server and hosts a malicious site, e.g., *attacker.com*. The web attacker can run JavaScript on the malicious site, but is not capable of compromising the browser code, or bypassing the same-origin policy. To advertise the malicious site, the attacker can promote the popularity of the site via Search Engine Optimization (SEO), and disseminate its shortened URL through emails, social networks and advertisements.

We assume that Alice is an ordinary web user who does not clear browser cache frequently, and she usually visits geo-location-oriented sites specific to her geo-location, e.g., Google, Craigslist, and Google Maps,

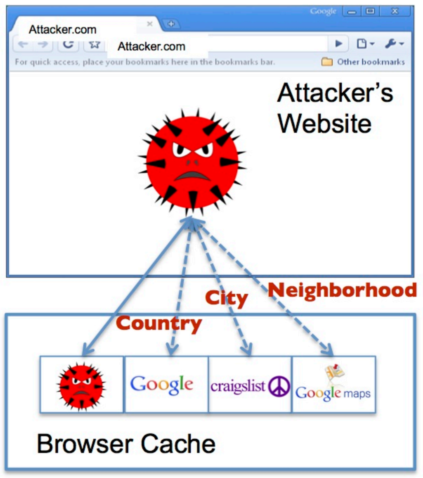

Fig. 1: Geo-inference attacks sniff location-sensitive resources left by the location-oriented sites (e.g., Google, Craigslist, and Google Maps) through timing side channels in the browser cache to infer the victim's geo-location.

so that she can take advantage of the customized local location-oriented services. When Alice visits the attacker's malicious site in her browser, she denies the unfamiliar site's request to access the device's location. However, the malicious payload in *attacker.com* can sniff the location-sensitive resources left by the location-oriented sites through side channels in the browser, as Figure 1 shows.

If Alice manually changes the location-oriented service to a location different from her location, and she never visits any websites provided for local people, her location will not be leaked to attackers, but she also loses the benefits of customized services for local users.

B. Browser Cache Timing Channels

In order to reduce the page load time of websites, all present mainstream browsers utilize memory cache or disk cache to save infrequently changed resources such as image files, JavaScript files, CSS files and so on [17]. Once the resources are cached locally, the browser can fetch the resources from local cached copies instead of downloading from the original website. Thus the browser cache reduces the page load time of the website by decreasing the round-trip time.

While browser cache provides efficiency in loading pages to users, it also introduces side channels to attackers. In a nutshell, by measuring the resources load time for a specific site twice in a victim's browser, the attacker can calculate the difference in the resources load time. If the difference is smaller than a threshold, it indicates that the user has visited this site previously; otherwise, the user has not. These implicit attacks are called timing attacks on browser cache [8].

C. *Geo-Inference Attacks via the Browser Cache*

As a typical case of attacks on browsing history, Wondracek et al. de-anonymize social network users by analyzing users' visited URLs [10]. In this paper, we show the implication of timing attacks in de-anonymizing the user's geo-location. Our main observation is that to improve user experience, websites usually provide geo-targeting services, e.g., Craigslist maintains city-specific sites to serve users in different cities.

In geo-inference attacks, the attacker makes guesses on the victim's geo-location, and then queries the cached resources corresponding to the geo-location. By utilizing the location-sensitive resources left by location-oriented sites in the browser, geo-inference attacks provide an oracle for attackers to verify the user's country, city, or neighborhood. For example, a paper author wants to check which one of the 20 PC members visits the supplementary website provided in an anonymous submission for review. Suppose the PC members are known to be from 20 different cities. The author can mount our geo-inference attacks on the website to locate the PC member, which queries whether the visitor of the website comes from any of the 20 cities. The author has a fare chance to successfully infer the reviewer of the paper, even if the visit is made through a web proxy.

III. GEO-INFERENCE ATTACKS: CASE STUDY

In this section, we show how to conduct geo-inference attacks at various granularity with mainstream browsers, i.e., Chrome, Firefox, Safari, IE and Opera. We set up a web site that contains scripts to explore side channels from browser cache. We use two well-known timing channels, including page load time in frames and resources (e.g., image file) load time. We also utilize two other vectors for querying, which have not been widely explored, i.e., Cross-Origin Resource Sharing (CORS) and <img>'s *complete* property. Together, these channels are practical ways to query cache states in present web browsers.

A. *Locate Your Country*

Google has 191 geography-specific domains to provide enhanced service for users. Usually, when a user opens *google.com* to search, Google will automatically redirect the page to the local Google site hosted in the user's physical location, e.g., *google.com.sg*. This makes Google a good case study for geo-inference attacks to locate the user's country.

In order to infer a user's country, we aim to find out which page from 191 Google's domains hits cache. To achieve this, we utilize Google's logo image, whose URL consists of Google's local domain, e.g., *google.com.sg*, and */images/srpr/logo11w.png* to specify Google's website. We demonstrate three ways to locate the user's country.

1) Geo-Inference Attacks with Image Load Time: Page load time [18] is the period of time from the initiation of the page load (e.g., click on a page link) to its completion in the browser. It usually consists of HTTP/HTTPS request time, response time, and the overhead of parsing and rendering the page in the browser.

Image load time is similar to page load time, which is the interval between requesting the image file and finishing rendering the image in the browser. We measured the image load time of Google logos from Google's 191 regional domains to determine the geo-location of the victim.

Measurement Techniques. We set the start time as an attribute of the image tag, and set the end time in the *onload* event handler. We make three measurements of the image load time, the first measurement without cache and the latter two with cache. If the first image load time is significantly larger than the latter two, we infer that the image is not cached in the user's browser. If the time difference is significantly small, e.g., 10 ms, we infer that the image hits cache. Since each logo stands for one local domain, and one domain is hosted in one country, which country's logo out of 191 logos is cached indicates the country the victim lives in.

Below is the piece of code to measure the image load time.

```
var image = document.createElement('img');
image.setAttribute('startTime', (new Date().
    getTime()));
image.onload = function()
{
    var endTime = new Date().getTime();
    var loadTime = endTime - parseInt(this.
        getAttribute('startTime'));
    ......
}
```

Listing 1: Measuring the image load time with JavaScript

Results. We found that the image load time of each Google's domain without cache is much larger than that with cache. We show the difference in Figure 2 as well as Figure 8 in the Appendix. Because there is only one request for the Google's logo image, when reloading the cached image file, the browser directly reads it from the cache, and the average image load time is usually 1 ms or 0 ms.

We have evaluated 191 Google's regional domains on five mainstream browsers, including Chrome, Firefox, Safari, Opera and IE, on both desktop and mobile platforms. We switch the region to US, UK, Australia, Singapore, and Japan with a multi-country proxy service (virtual private network). According to the results, there are no big differences among results from these browsers. Due to the limited space, we only show the Singapore-based results in Chrome as the representative

TABLE I: Geo-inference attacks on mainstream browsers

	I	II	III	IV	V
Chrome (Linux, Windows & OS X + Android & iOS)	✓	✓	–	✓	✓
Firefox (Linux, Windows & OS X + Android)	✓	–	✓	✓	✓
Safari (Windows & OS X + iOS)	✓	–	✓	✓	✓
Opera (Linux, Windows & OS X + Android & iOS)	✓	✓	–	✓	✓
IE (Windows)	✓	–	✓	✓	✓

I : Locate Your Country with Image Load Time
II : Locate Your Country with Cross-Origin Resource Sharing (CORS)
III: Locate Your Country with <img>'s *complete* Property
IV: Locate Your City with Craigslist
V : Locate Your Neighbourhood with Google Maps
✓ : Susceptible
– : Not Susceptible

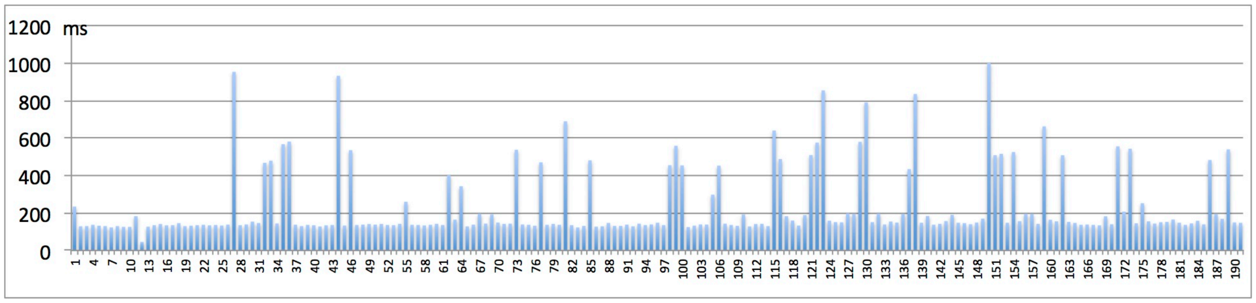

Fig. 2: *Difference in image load time (in millisecond):* Without Cache (> 129 ms) v.s. With Cache (0 ∼ 1 ms), for 191 Google's regional domains in Chrome on Mac OS X.

in this paper. Table I shows the details of other browsers and platforms. We present the experimental results on Android in the Appendix.

2) Geo-Inference Attacks with Cross-Origin Resource Sharing (CORS): Cross-origin resource sharing (CORS) [19] allows JavaScript to make XMLHttpRequests to another origin. Although Google does not allow JavaScript in other origins to read XMLHttpRequests responses from Google, we can still measure the time interval between sending XMLHttpRequests to Google and receiving the rejection responses.

Measurement Techniques. We set the start time in the *onloadstart* event handler, and set the end time in the *onloadend* event handler. We measured three rounds of the request load time of Google's logo from Google's 191 regional domains. The big difference between the first round and last two rounds indicates a cache miss for the image. If the request load time is approximately same for three rounds, the image is cached in the user's browser.

Below is the piece of code to measure the load time of XMLHttpRequests.

```
var starTime, endTime, loadTime;
var xmlhttp = new XMLHttpRequest();
xmlhttp.onloadstart = function()
{
    startTime = (new Date()).getTime();
}
xmlhttp.onloadend = function()
{
    endTime = (new Date()).getTime();
    loadTime = endTime - startTime;
    ......
}
```

Listing 2: Measuring the load time for XMLHttpRequests

Results. We found that the request load time without cache is larger than that with cache. Similar to Figure 2, Figure 3 shows the difference, which indicates that it is practical to locate users' countries in this way. Figure 9 in the Appendix shows that this method is also applicable to mobile browsers.

3) Geo-Inference Attacks with <img>'s complete Property: As for the <img> tag in HTML, it has *complete* property to indicate that the image file finishes loading. We found that this attribute also indicates whether the image file is cached or not in some browsers.

Measurement Techniques. We created <img> elements for each URL of Google's regional domains, and check the complete property of each element. If the return value is true, the image is cached in the user's browser; otherwise, the image is not cached.

Below is the piece of code to distinguish whether the image file is cached or not.

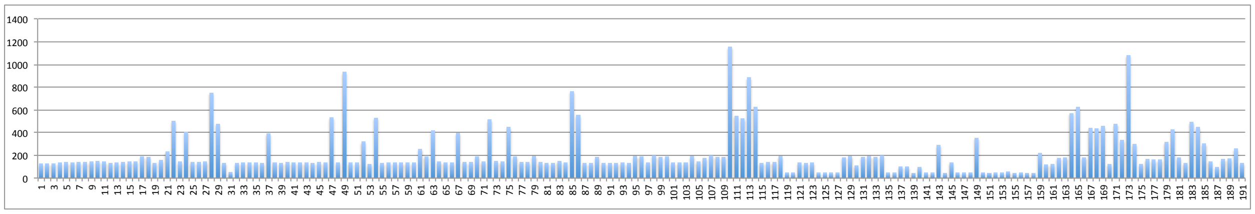

Fig. 3: *Difference in load time (in millisecond) for XMLHttpRequests:* Without Cache (> 100 ms) v.s. That With Cache (0 ∼ 1 ms), for 191 Google's regional domains in Chrome on Mac OS X.

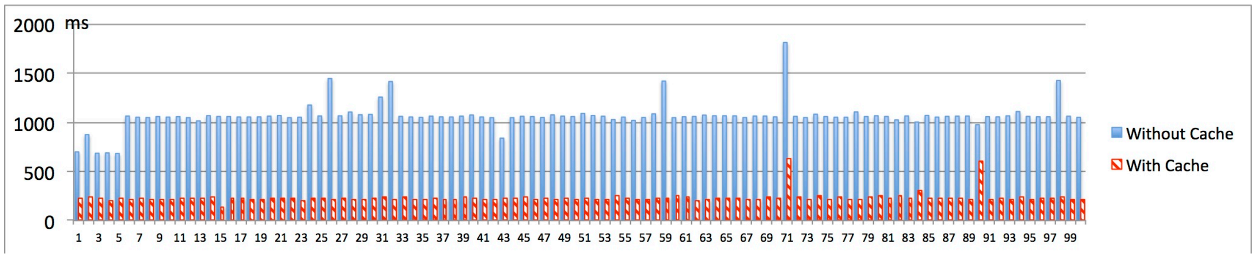

Fig. 4: The big difference between the page load time (in millisecond) of 100 Craigslist sites without cache (> 1000 ms) and with cache (≈ 220 ms) indicates geo-inference attacks with Craigslist in Chrome on Mac OS X.

```
function cached(url)
{
    var image = document.createElement('img');
    image.src = url;
    return image.complete || image.width+image.
        height > 0;
}
```

Listing 3: Distinguishing whether an image file is cached or not

Results. We have evaluated Google's 191 regional domains, and this method can reliably distinguish cache hit from cache miss. This method works on Firefox, Safari and IE, but is not applicable for Chrome and Opera.

B. *Locate Your City*

Several websites offer content specific to cities, e.g., Craigslist [20] is a large classified advertisements website that covers 712 cities on the world. Since all the sub-sites are city-oriented and can be loaded in frames, we measured the page load time to locate the user's city.

Measurement Techniques. We set the start time as an attribute of the iframe, and set the end time in the *onload* event handler. We measured the page load time of Craigslist's 712 city-oriented websites in iframes three times. If the difference between the page load time at the first attempt and the latter two is significantly large, e.g., 500 ms, we infer that the page is not cached in the user's browser; otherwise, the page has been visited. From the recently visited page (cached page), we can determine the victim's city.

Below is the piece of code to measure the page load time of Craigslist sub-sites.

```
var page = document.createElement('iframe');
page.setAttribute('startTime', (new Date()).
    getTime());
page.onload = function ()
{
    var endTime = (new Date()).getTime();
    var loadTime = ( endTime - parseInt(this.
        getAttribute('startTime')));
    ......
}
```

Listing 4: Measuring the page load time of Craigslist sub-sites

Results. Our attacks can reliably identify whether a city-specific page is cached in the user's browser. We consider the average page load time of the latter two measurements as the time with cache. As Figure 4 as well as Figure 10 in the Appendix demonstrates, the difference is quite large. Due to more resources that Craigslist contains than Google's logo (only one image file), the page load time with cache is around 220 ms on desktop or 620 ms on mobile, which is much higher than 0 ms in Figure 2 and Figure 8.

Since the user buys or sells goods on Craigslist mostly in his/her city, the attacker can reveal the victim's real city with this method, even if they use VPN or Tor.

C. *Locate Your Neighborhood*

To extend our study to finer granularity than city level, we exploit the cached resources from online map service websites, e.g., Google Maps.

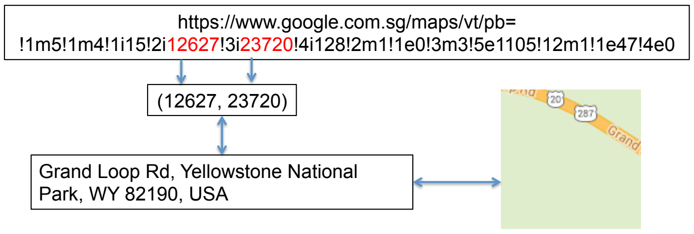
https://www.google.com.sg/maps/vt/pb=
!1m5!1m4!1i15!2i12627!3i23720!4i128!2m1!1e0!3m3!5e1105!12m1!1e47!4e0
(12627, 23720)
Grand Loop Rd, Yellowstone National Park, WY 82190, USA

Fig. 5: URLs of map tiles have static mappings to geo-locations in Google Maps.

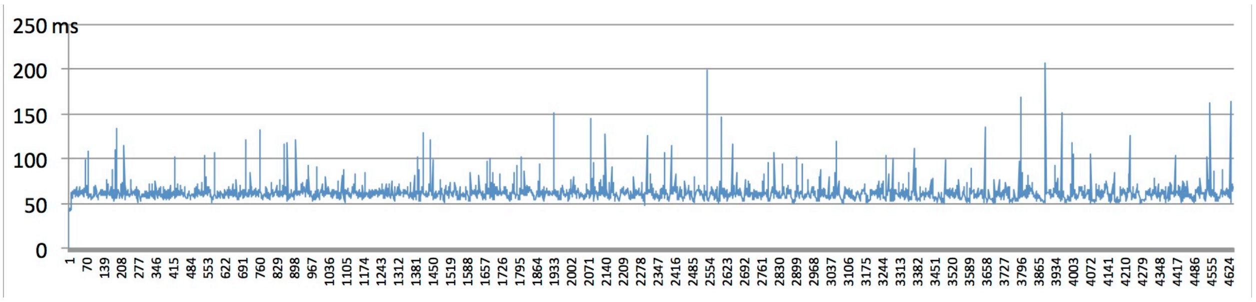

Fig. 6: *Difference in page load time (in millisecond):* Without Cache ($>$ 50 ms) v.s. With Cache ($0 \sim 1$ ms), for 4,646 map tiles of New York City from Google Maps in Chrome on Mac OS X.

Google Maps [21] is a web map service supported by Google, which provides maps and street views at each place. When a user searches one place, Google Maps presents the specific area to the user and the user can zoom in/out to control the map precision. Furthermore, the user can click on the area to see the street views mapping to the area. For all the searching and locating steps, Google Maps will request and fetch numerous map tiles from the web server.

In fact, the maps are tessellations of map tiles, and each tile localizes at a predicable URL. After manually analyzing the URLs of the requests for map tiles, we figure out that the URLs have some patterns that are a function of real geo-location coordinates[1]. For example, as Figure 5 shows, we can derive the coordinates (12627, 23720) of “Grand Loop Rd” from the requesting URL. Though the coordinates are not the real geographic coordinates, we can predict the URLs of one specific area’s map tiles corresponding with the coordinates. By recording the coordinates of all the vertices on the area, we can traverse all the possible points inside the area. After converting the coordinates to URLs, we measured the image load time of the map tiles in the victim’s city, which is verified by Craigslist. As Google Maps usually zooms in to the user’s current location when the user starts to browse Google Maps in the browser, by sniffing the victim’s recently visited map tiles, we can locate the victim’s streets and neighborhood.

[1]We have investigated 11 websites that provide map service, including Google Maps, Bing Maps, Yahoo! Maps, arcGIS, HERE, OpenStreetMap, ViaMichelin, MapQuest, WikiMapia, Navionics and Mappy. We found that the URLs of map tiles from these sites are all statically predictable.

Measurement Techniques. We set the start time as an attribute of the image, and set the end time in the *onload* event handler. In our experiments, we measured the image load time for all 4,646 map tiles in New York City on Google Maps three times. If the image load time at the first attempt is much larger than the latter two, e.g., 40 ms, it indicates a cache miss for the image. If the difference between each time is quite small, e.g., 10 ms, the image hits cache.

Results. We found that the image load time of map tiles from Google Maps without cache is much larger than that with cache. Therefore, we can reliably determine the user’s location via measuring the image load time of map tiles in the user’s city. As Figure 6 as well as Figure 11 in the Appendix shows, the huge difference makes it quite straightforward to distinguish whether the specific map tile is cached in the browser or not.

IV. Geo-Targeting on Alexa Top 100 Websites

In addition to our case studies with Google, Craigslist and Google Maps, we also show that a set of popular websites is susceptible to geo-inference attacks. We present our results from two aspects.

A. *Reliability of Timing-Based Attacks via the Browser Cache*

Geo-inference attacks are based on the techniques of timing attacks on the browser cache. We revisit such timing-based attacks by conducting a study for Alexa Top 100 websites on five mainstream browsers (Chrome, Firefox, Safari, Opera and IE) as well as TorBrowser[2]. We found that browsing history sniffing is still open to timing-based attacks on these browsers.

We measured the page load time three times per site. In Figure 12 and Figure 13 in the Appendix, we consider the average difference between the second measurement and the third measurement as the page load time with cache. The difference between page load time without cache and that with cache is quite large. As Figure 12 and Figure 13 show, our proposed geo-inference attacks can reliably measure the page load time of Alexa Top 100 websites and sniff the victim's browsing history. Table I shows that our geo-inference attacks are applicable to all five mainstream browsers.

If a website sets 'X-Frame-Options' to 'SAMEORIGIN' or 'DENY' in the response headers, the attacker cannot load the site into frames, so the page load time actually equals the request load time and is not reliable. We investigated Alexa Top 100 websites, and found that only 32 sites set 'X-Frame-Options' to 'SAMEORIGIN', and five sites set it to 'DENY'. Nevertheless, the attacker can utilize a geo-targeting image or other resources that are specific to the site, and measure the resources load time. For example, the URLs of Google's logo images are *www.google.com.sg/images/srpr/logo11w.png* and *www.google.co.jp/images/srpr/logo11w.png*, respectively for *google.com.sg* in Singapore and *google.co.jp* in Japan. As Section III-A shows, geo-inference attacks with resources load time are quite reliable.

B. Prevalence of Location-Sensitive Resources

To demonstrate the susceptibility of geo-inference attacks, we systematically analyzed Alexa Top 100 websites, and identified the location-sensitive sites and their location-sensitive resources. We exclude 45 domains that are one of the following categories.

- Sites are highly related to specific countries, e.g., google.de and yahoo.co.jp.
- Sites are known to contain pornographic content, e.g., xvideos.com and xhamster.com.
- Sites are unreachable, e.g., akamaihd.net, and googleusercontent.com.

Using Hotspot Shield, we visited the 55 websites in five different countries (USA, UK, Australia, Japan and Singapore) and recorded all the URLs of cached resources for each website. As for each site, we automatically compared the collected URLs from different countries with our analysis tool, sorted out the URLs that vary in different countries, and then manually verified the resource that has the same functionality but the URL changes from country to country.

Our study demonstrates that the majority of websites contain location-sensitive resources, which are vulnerable to geo-inference attacks. 62% of these websites have location-sensitive resources, such as domain and logo. The user who recently visited any of these websites is vulnerable to expose his/her geo-location to the attackers who conduct geo-inference attacks.

V. Discussion & Solution

Beyond its techniques, the geo-inference attack we demonstrate can be an effective attack vector in practice.

A. Effectiveness in Locating Victims

The geo-inference attacks discussed in this paper do not directly pinpoint the geo-locations of targeted users. Instead, they allow attackers to verify whether the user has been to a given location. Consider one scenario where a paper author wants to check which one of the 20 PC members visits the supplementary website provided in an anonymous submission for review via an IP-anonymization VPN service. Suppose the PC members are known to be from 20 different cities. The author can mount our geo-inference attacks on the website, which queries whether the visitor of the website comes from any of the 20 cities.

In the previous case, it only needs to issue 20 queries. However, the number of queries may grow significantly depending on different scenarios. Now consider another example where the attacker wants to locate a visitor's neighborhood within the New York City. In this case, Google Maps has 4,646 map tiles for the entire city, i.e., the attacker needs to make 4,646 queries in worst case to locate the visitor. According to our experiments, one web attacker can verify around 5 to 10 geo-locations every second (Figure 4 and Figure 10). This amounts to eight minutes' time in a single-threaded execution; of course, the attacker can speed up the process with parallel queries.

B. Pros & Cons of Potential Defenses

Technically speaking, geo-inference attacks can be prevented by existing defenses against privacy leakage via the browser cache. However, as we discuss next, there are trade-offs we need to consider when deploying such defenses.

Private browsing mode is not the cure: The private browsing mode (Private Browsing in Safari and Firefox, Incognito Mode in Chrome, Private Window in Opera, and inPrivate Browsing in IE) prevents browsers from permanently storing any history, cookies or other client-side states for websites. However, contrary to what the name naively suggests, the private browsing mode does not prevent caching of web resources *during*

[2]We run the experiments on TorBrowser 3.5.2.1, which is based on Firefox 24.3.0. We discuss TorBrowser in Section V.

users' private browsing [22]. Instead, all cache incurred during the private browsing mode is automatically cleared *after* the user closes its window. Therefore, users of the private browsing mode are still susceptible to geo-inference attacks via the browser cache.

Rectifying inconsistent access policy with XMLHttpRequest: Although Cross-Origin Resource Sharing (CORS) has been integrated into modern web browsers, it only allows websites to control which web origins can access the content of XMLHttpRequest responses. It does not, on the other hand, specify whether other origins can get the notification when the response whose content is inaccessible arrives. These error notifications can be a source of timing leakage as shown in Section III-A2. Such an inconsistent policy is one of the root causes that make geo-inference attacks possible. It is also quite different from other timing exposure with image loading (Section III-A1 and III-A3), where no explicit access control policy exists.

It thus appears as a necessary rectification step to block the XMLHttpRequest status notifications if the website has denied the request to access it. However, additional analysis and experiments are necessary to figure out whether any valid case that requires access to the status notification of an access-denied XMLHttpRequest. Besides, this is just one of the many existing side-channels that leak users' privacy; it is not a holistic solution to defend geo-inference attacks via the browser cache.

Segregating browser cache works but is expensive: In current browsers, cache is like a huge repository; there are no explicit boundaries or restrictions among different domains. Jackson et al. [15] deploy *Same-Origin Policy* on the browser cache to prevent cross-origin sites from loading the resources stored by the original sites. In principal, such a solution would defeat geo-inference attacks. To the authors' knowledge, this solution has not been adopted by modern web browsers in the default setting. We speculate that performance could be one of the concerns. Thus, we re-implement their cache segregation policy in Chromium 34 (latest release), and measure the estimated performance overhead incurred from it.

Since the most obvious performance overhead comes from loading cross-origin resources, we measure the page load time for Alexa Top 100 websites[3]. As shown in Figure 7, the same-origin caching policy triggers more than 50% performance overhead. This indicates that the same-origin caching policy may affect the page load time significantly, although such performance overhead might be mitigated to a certain extent by whitelisting domains as same-origin, such as Content Delivery Network (CDN) domains. We try to whitelist the resources from CDN, and show that the performance overhead is still non-negligible after whitelisting for five sites in Table II[4]. We will discuss alternatives to this policy in Section V-C.

TABLE II: How does whitelisting resources from CDN affect performance overhead?

	I	II
google.com	140%	9.09%
facebook.com	80.81%	5.05%
youtube.com	45.45%	12.59%
yahoo.com	123.51%	31.03%
baidu.com	107.27%	6.67%

I: Performance Overhead Without Whitelisting Resources From CDN
II: Performance Overhead With Whitelisting Resources From CDN

Can VPN or Tor prevent the attacks?: Geo-inference attacks are based on timing attacks against browser cache, so they are not affected by VPN services that replace the original IP addresses of users. Although the current version of Tor Browser Bundle disables disk cache in browsers by default, memory cache is still active. As a result, it is similar to the private browsing mode. Browser cache is available until the user closes the browser, where the cache stored in memory is invalidated. For browser caches, we also find that Tor-Browser adds an additional "domain=string" property to label every cache entry with the top-level window's domain [23]. Therefore, TorBrowser can protect users from geo-inference attacks when the malicious page's top-level URL is different from the targeted site. However, for mashup websites, all the embedded sites in frames share the same top-level window's domain, i.e., the mashup's domain. The malicious embedded site can query the cache status of other embedded sites' cached location-sensitive resources to infer the user's geo-location. Thus only adding "domain=string" property on the cache entry is insufficient. To prevent timing attacks, Jackson et al consider two observers, i.e., the site embedding the content and the host of the content to label the cache entry. Instead, TorBrowser only uses the top-level window's domain as the "domain" field to enforce the caching mechanism. Therefore, Jackson's solution defeats geo-inference attacks in the mashup scenario, but TorBrowser does not. We have run our experiments on TorBrowser v3.5.2.1 as shown in the Appendix, and find that TorBrowser can still leak users' geo-location information for mashup websites.

Randomizing timing measurements: As with other timing-based attacks, we can add noise into the timing measurement mechanisms related to browser cache queries. However, introduction of such noise needs to be carefully designed and verified to have mini-

[3]Except three unreachable sites, googleusercontent.com, akamaihd.net, thepiratebay.sx.

[4]Table II shows the performance overhead without or with whitelisting resources from CDN comparing with the vanilla version without same-origin caching policy.

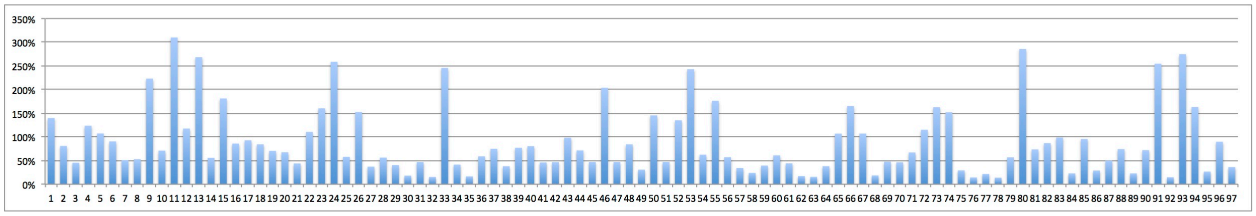

Fig. 7: Enforcing same-origin caching policy cause 20% to 300% performance overhead to load each site comparing with the vanilla version without policy.

mal impact on browser performance and responsiveness. This could be an intricate engineering effort in future.

Server-side protection of users' geo-locations: Websites may voluntarily switch to a non-geo-targeting mode upon request of users. As part of the reason for deploying geo-targeting is to improve the responsiveness of websites, this will probably degrade website performance. As a middle ground, websites may randomize the URLs for geo-targeting resources periodically. This way, they can still benefit the performance improvement from geo-targeting and cache, at least for a period of time. At the same time, periodical changes of URLs can mitigate geo-inference attacks via the browser cache, as the additional randomness makes it difficult for attackers to predict the URLs to query.

C. A Better Defense Solution?

The cache segregation proposed by Jackson et al. appears as a promising direction to resolve the geo-location leakage via browser cache. In addition, with collaborative effort from both the browser vendors and web servers, the performance overhead incurred could be significantly reduced for most of use cases. We envision an endeavor similar to Do Not Track [24]. Browsers should provide an option for end users to opt out of caching location-sensitive web resources, such as map tiles in Google Maps and geo-targeting country pages. This way, users can voluntarily protect their geo-location information from geo-inference attacks. In solutions aimed to protect users' privacy, such as Tor or private browsing mode, this option should be turned on by default. Such an option can be communicated to web servers as an HTTP request header. At the server side, web developers need to respect this option, and set the `Cache-Control: no-cache` HTTP response header for location-sensitive resources. However, discerning such resources might be a challenging task for today's web applications. We have experimented with a prototype tool during our study, which can aid web developers in identifying location-sensitive resources.

More specifically, the tool collects the URLs of cached resources during our manual testing of the web applications from different geo-locations (behind VPN services). Then it compares such collections and automatically labels the URLs that are potentially location-sensitive. Web developers can further verify resources corresponding to this scaled-down set of potentially location-sensitive URLs, and confirm the location-sensitive resources. In the future, we can integrate the no caching support for location-sensitive resources into web application generation frameworks.

VI. Related Work

There has been extensive research on both privacy leakage attacks and defense solutions. We discuss the most closely related work in this section.

A. Privacy Leakage

Jang et al. study popular privacy violation practices in websites and third-party JavaScript libraries [25]. Researchers further present another way to sniff browsing history with the *CSS visited pseudoclass* [26], [27]. In addition to traditional leakage channels via DOM or CSS primitives, user interactions can also be misused to reveal users' browsing history [11]. More recently, new primitives introduced into browsers also expose new attack vectors of privacy leakage, including CSS filters [12] and SVG filters [13].

On the other hand, side channels, such as timing attacks that were previously used to crack crypto systems or protocol implementations [28]–[30] have also been explored for privacy leakage in browsers. Felten et al. [8] first introduced timing-based attacks to detect browser cache. Bortz et al. [9] apply timing attacks to more web applications and scenarios.

With the popularity of social networks, user privacy becomes a serious concern. Wondracek et al. show that by analyzing the group memberships of social network users, attackers can potentially identify the identities of users [10].

This paper presents a different view into privacy leakage in browsers. Instead of proposing new leakage channels, we provide a more in-depth evaluation of the implications of existing leakage channels. We specifically demonstrate that many of the timing-based attack vectors still exist today, and they are open to attackers to sniff users' geo-locations.

B. Defense

To prevent direct privacy leakage channels, L.D. Baron proposes a fix to misused CSS visited styles that leak browsing history, which has been incorporated into all popular web browsers [14]. As a more fundamental endeavor, Jackson et al. [15] analyze various degrees of cooperation between sites to track users and apply a refined same-origin policy on browser cookie and cache to protect browser states. Jakobsson et al. [16] neutralize browser sniffing by performing URL personalization on the fly at the server side. As more social networking gadgets are also used to track users, Roesner et al. propose a browser extension to allow users to prevent such gadgets to track them without their explicit interactions [31]. As we discuss in Section V-B, we expect more practical solutions with a better balance between performance and privacy preservation.

C. Location Privacy

The privacy of geo-locations has attained significant attention from security researchers. Gedik et al. propose a personalized k-anonymity model to protect location privacy with mobile clients [32]. Nevertheless, recent boom in location-based web and mobile services also demonstrates that users are generally willing to reveal partial geo-location information for better services. Duckham et al. thus propose a formal model that allows efficient balancing between the two competing needs, the need for high quality location-based services and that for location privacy [33]. Similarly, Beresford et al. develop a refined method, called the mixed zone, to achieve enhanced location privacy in location-based services [34]. In this paper, we primarily consider empirical geo-location leakage on the present web, where location information is explicitly available without anonymized protection.

VII. Conclusion

In this paper, we present geo-inference attacks via the browser cache, which utilize the cached location-sensitive resources left by location-based websites. Our geo-inference attacks provide an oracle for web attackers to verify guessed geo-locations of victims. In our case studies, we demonstrate the reliability and the power of our geo-inference attacks to track the victim's country, city, and neighborhood. Furthermore, we analyze Alexa Top 100 websites for their susceptibility of geo-inference attacks. We discuss the effectiveness in locating victims and potential defenses for geo-inference attacks, and suggest collaborative solutions between browser vendors and web applications to prevent such geo-location leakage channels.

Acknowledgements

We thank the anonymous reviewers of this paper for their insightful feedback. We thank Chunwang Zhang, Hong Hu, and Shweta Shinde for their comments on an early presentation of this work. We also thank Roger Dingledine and Georg Koppen for their feedback on our experiments on TorBrowser. This work is supported by the Ministry of Education, Singapore under Grant No. R-252-000-495-133. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the Ministry of Education, Singapore.

References

[1] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security," in *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*, 2010.

[2] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards ip geolocation using delay and topology measurements," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006.

[3] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, "Ip geolocation databases: unreliable?" *ACM SIGCOMM Computer Communication Review*, 2011.

[4] M. Balakrishnan, I. Mohomed, and V. Ramasubramanian, "Where's that phone?: geolocating ip addresses on 3g networks," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, 2009.

[5] "Virtual private network," http://en.wikipedia.org/wiki/Virtual_private_network.

[6] "Tor," https://www.torproject.org/.

[7] "'revolution 2.0': How social media toppled a dictator," http://www.npr.org/2012/02/08/145470844/revolution-2-0-how-social-media-toppled-a-dictator.

[8] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *Proceedings of the 7th ACM conference on Computer and communications security*, 2000.

[9] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proceedings of the 16th international conference on World Wide Web*, 2007.

[10] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *Security and Privacy (SP), 2010 IEEE Symposium on*, 2010.

[11] 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 *Security and Privacy (SP), 2011 IEEE Symposium on*, 2011.

[12] R. Kotcher, Y. Pei, P. Jumde, and C. Jackson, "Cross-origin pixel stealing: timing attacks using css filters," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013.

[13] P. Stone, "Pixel perfect timing attacks with html5," *Online at http://www. contextis. com/files/Browser_ Timing_ Attacks. pdf*, 2013.

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

[15] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in *Proceedings of the 15th international conference on World Wide Web*, 2006.

[16] M. Jakobsson and S. Stamm, "Invasive browser sniffing and countermeasures," in *Proceedings of the 15th international conference on World Wide Web*, 2006.

[17] "Optimize caching," https://developers.google.com/speed/docs/best-practices/caching.

[18] "Interpret site speed," https://support.google.com/analytics/answer/2383341?hl=en.

[19] "Cross-origin resource sharing," http://en.wikipedia.org/wiki/Cross-origin_resource_sharing.

[20] "Craigslist," http://www.craigslist.org.

[21] "Google maps," https://maps.google.com/.

[22] G. Aggarwal, E. Bursztein, C. Jackson, and D. Boneh, "An analysis of private browsing modes in modern browsers," in *Proceedings of the 19th USENIX Conference on Security*, ser. USENIX Security'10, 2010.

[23] M. Perry, E. Clark, and S. Murdoch, "The design and implementation of the tor browser," https://www.torproject.org/projects/torbrowser/design/#identifier-linkability.

[24] J. Mayer and A. Narayanan, "Do not track - universal web tracking opt out," http://donottrack.us/.

[25] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010.

[26] A. Janc and L. Olejnik, "Feasibility and real-world implications of web browser history detection," *Proceedings of W2SP*, 2010.

[27] S. Stamm, "Plugging the css history leak," http://blog.mozilla.org/security/2010/03/31/plugging-the-css-history-leak/.

[28] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in CryptologyCRYPTO96*, 1996.

[29] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, 2005.

[30] O. Aciiçmez, W. Schindler, and Ç. K. Koç, "Improving brumley and boneh timing attack on unprotected ssl implementations," in *Proceedings of the 12th ACM conference on Computer and communications security*, 2005.

[31] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against third-party tracking on the web," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12, 2012.

[32] B. Gedik and L. Liu, "Protecting location privacy with personalized k-anonymity: Architecture and algorithms," *IEEE Transactions on Mobile Computing*, 2008.

[33] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," in *Proceedings of the Third International Conference on Pervasive Computing*, ser. PERVASIVE'05, 2005.

[34] A. R. Beresford and F. Stajano, "Mix zones: User privacy in location-aware services," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, ser. PERCOMW '04, 2004.

APPENDIX

A. Geo-Inference Attacks on TorBrowser

As we discussed in Section V, TorBrowser adds an additional "domain" field for each cache entry to prevent the cross-origin timing channels [23]. However, when a user uses a mashup site on TorBrowser, e.g., *igoogleportal.com*, the "domain" field for each cache entry of all the embedded sites in iframes is the mashup site's domain. Therefore, the embedded malicious site can utilize the timing channels to query the cache status of other embedded sites' cached resources to infer the user's geo-location. We demonstrate a working exploit example and provide suggestions below. We have tested this example on TorBrowser v3.5.2.1.

Exploit Example: Suppose a user is an Argentinean, when visiting the sub-page for Godaddy in the igoogleportal's page [5], he switches the country to Argentina. TorBrowser caches the Godaddy's logo in Argentina with setting the "domain" field in the cache entry as *http://www.igoogleportal.com*. After that, when the user visits the geo-inference page, the scrips in the page will create and measure the image load time of all Godaddy's logos for different countries. Since the geo-inference page is also under the top-level window's domain (*http://www.igoogleportal.com*), thus TorBrowser will directly load the cached logo image for the Godaddy's logo in Argentina. Therefore, the scripts can figure out the minor time difference to create the Godaddy's logo for Argentina, and infer that the user is in Argentina.

Suggested Solutions: To completely defeat such attacks, we suggest two solutions for TorBrowser. First, TorBrowser can disable memory cache by default like TorBrowser 2.3.25. Second, TorBrowser can set the top-level window's domain together with the frame's domain as the additional field in the cache entry instead of only using the top-level window's domain. Either one can help to mitigate geo-inference attacks. In fact, it is more practical for website developers to set "cache-control: no-cache" for the location-sensitive resources, e.g., logos, as we discussed in Section V-C.

[5]The user logins *http://www.igoogleportal.com/* with the account's name: *aliceisnotbob1@gmail.com* and password: *aliceisbob*.

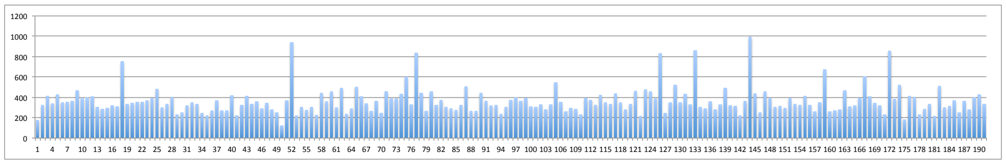

Fig. 8: *Difference in image load time (in millisecond):* Without Cache (> 200 ms) v.s. With Cache (0 ∼ 1 ms), for 191 Google's geo-targeting domains in Chrome on Android.

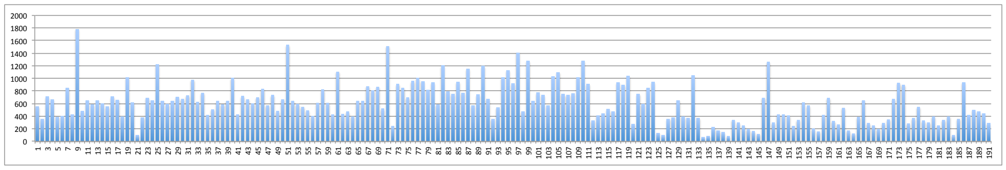

Fig. 9: *Difference in load time (in millisecond) for XMLHttpRequests:* Without Cache (> 400 ms) v.s. With Cache (20 ∼ 30 ms), for 191 Google's geo-targeting domains in Chrome on Android.

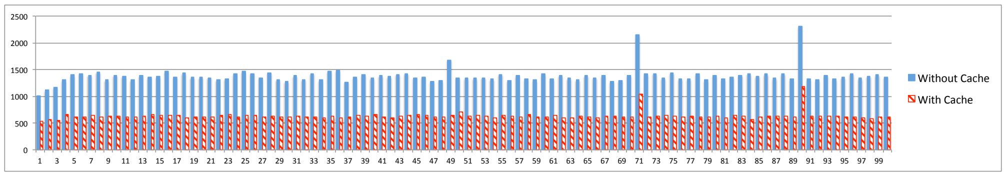

Fig. 10: The page load time (in millisecond) of 100 Craigslist sites without cache (> 1300 ms) is substantially larger than the time with cache (≈ 620 ms) in Chrome on Android.

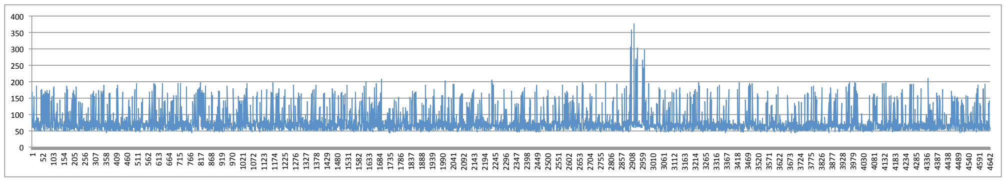

Fig. 11: *Difference in page load time (in millisecond):* Without Cache (> 50 ms) v.s. With Cache (0 ∼ 5 ms), for 4,646 map tiles of New York City from Google Maps in Chrome on Android.

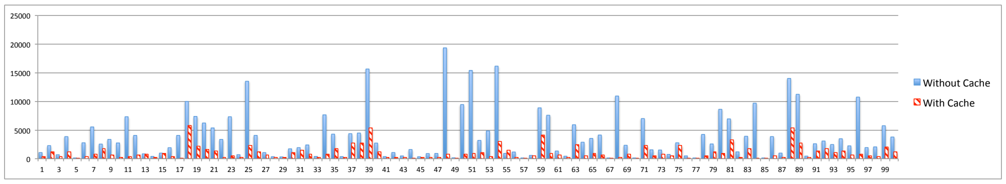

Fig. 12: The difference between the page load time (in millisecond) of Alexa Top 100 websites without cache and with cache indicates that it is effortless for timing-based attacks to distinguish whether the site is cached or not in Chrome on desktop.

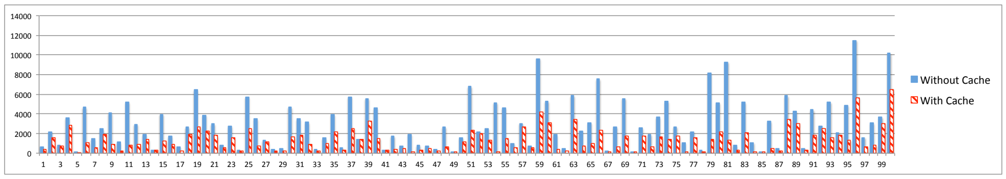

Fig. 13: The page load time (in millisecond) of Alexa Top 100 websites without cache is much larger than the time with cache in Chrome on Android.