The Clock is Still Ticking: Timing Attacks in the Modern Web

Tom Van Goethem[‡], Wouter Joosen[‡], Nick Nikiforakis[†]
[‡]iMinds-Distrinet, KU Leuven, 3001 Leuven, Belgium
firstname.lastname@cs.kuleuven.be

[†]Department of Computer Science, Stony Brook University
nick@cs.stonybrook.edu

Abstract

Web-based timing attacks have been known for over a decade, and it has been shown that, under optimal network conditions, an adversary can use such an attack to obtain information on the state of a user in a cross-origin website. In recent years, desktop computers have given way to laptops and mobile devices, which are mostly connected over a wireless or mobile network. These connections often do not meet the optimal conditions that are required to reliably perform cross-site timing attacks.

In this paper, we show that modern browsers expose new side-channels that can be used to acquire accurate timing measurements, regardless of network conditions. Using several real-world examples, we introduce four novel web-based timing attacks against modern browsers and describe how an attacker can use them to obtain personal information based on a user's state on a cross-origin website. We evaluate our proposed attacks and demonstrate that they significantly outperform current attacks in terms of speed, reliability, and accuracy. Furthermore, we show that the nature of our attacks renders traditional defenses, i.e., those based on randomly delaying responses, moot and discuss possible server-side defense mechanisms.

Categories and Subject Descriptors

K.4.1 [**Computers and Society**]: Public Policy Issues—*privacy*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

Keywords

Side-channel attacks; privacy; web-based attacks

1. INTRODUCTION

Ever since the first web browser, browser vendors have been eagerly adding new features to their software. This eagerness helped the web transition from a static information retrieval medium, to a ubiquitous system where billions of users each have their personalized view, and share personal data on numerous online services. While these new browser features allow web developers to create applications that were not possible using traditional HTML, such as, single-page dynamic websites, they have also brought along various types of vulnerabilities, for instance the execution of attacker-controlled code in a cross-site scripting attack. In addition, some types of attacks arise from the unexpected interplay between different browser components and can thus be very hard to eliminate.

A powerful yet underappreciated class of attacks are side-channel attacks. In these attacks, an attacker leverages the information exposed by the unintended behavior of specific mechanisms, in order to disclose secret or private information. One of the most well-known side-channel attacks in browsers is the disclosure of pages the user had previously visited by first modifying the color of visited links using the CSS `:visited` pseudo-class, and subsequently requesting the computed style in JavaScript [9]. In an empirical study, researchers discovered that this technique was actively being used by various high-profile websites to uncover a user's browsing history [17]. This finally lead to a lawsuit against an advertising company that leveraged the history hijacking attacks to infer visitors' interests [1,15]. In a related attack, Felten et al. have shown that the time required to load an external resource can leak sensitive information, compromising a user's privacy [12]. The timing attack described by the researchers leverages the reduced loading time for cached resources to uncover recently visited, and thus cached, web pages. While this attack has been known for over 15 years, the timing side-channel leak, which is inherent to a browser's design, is still present in modern browsers. Just recently, researchers showed that using exactly the same cache timing techniques on various online services, an adversary can derive the geo-location of web users [18].

In addition to cache-based timing attacks, which can only be applied to static resources, Bortz and Boneh presented cross-site timing attacks where an adversary measures the time it takes for a user to download a dynamically generated resource [7]. The resulting timing measurements often disclose information on the state of the user at the vulnerable cross-origin website, e.g., whether the user is currently logged in to that website.

In this paper, we expand upon the aforementioned prior work, and focus on various new browser features that can be exploited by adversaries to obtain substantially more accurate timing measurements. Contrary to classic timing at-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
CCS '15 Denver, Colorado USA
© 2015 ACM. ISBN 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813632.

tacks which are subject to several limitations, such as variations in latency and instability of the network, our newly introduced attacks *do not* rely on the network download time, and therefore do not suffer from these limitations. We show that by using these new attack vectors, adversaries are able to rapidly obtain timing measurements that can be analyzed in order to estimate the size of a cross-origin resource. By the means of real-world attack scenarios on five of the most popular social networks, we illustrate how adversaries can apply these novel timing techniques to obtain various types of personally identifiable information from an unwitting user. For targeted advertising purposes, an adversary could use this information to create a profile based on the user's demographics and interests. Alternatively, the attacker can leverage information acquired from online social networks to de-anonymize the user, as has been shown in previous research [42].

Motivated by the effectiveness of our proposed attacks, as well as the, seemingly, innumerable opportunities to apply these attacks on popular websites, we discuss a possible anti-CSRF-like server-side countermeasure that hides the difference in resource sizes from potential cross-site attackers.

Our main contributions are:

- We evaluate various browser features with regard to the timing information they expose.
- We propose several new timing techniques, and demonstrate that our techniques outperform existing attacks in speed and reliability, allowing an adversary to estimate the size of a cross-origin resource despite unfavorable network conditions.
- We describe how an attacker can use these timing techniques to extract personally identifiable information, exemplified by five attack scenarios on widely-used social network websites.
- We discuss possible server-side solutions that have the potential to mitigate all variations of cross-site timing attacks.

2. BACKGROUND

Timing attacks are one of the oldest types of side-channel attacks, where the time required to perform a certain operation is leveraged to deduce private information on the attacked system. For example, Kocher found that unintentional timing characteristics reveal sufficient information to extract the entire secret key from a vulnerable cryptosystem [20]. In the context of the web, Felten and Schneider [12] were the first to indicate that adversaries could use timing attacks to compromise a user's online privacy. In their research, published in 2000, they describe how different types of caching can leak information about the static web pages a user recently visited. Several years later, Bortz et al. presented two types of web-based timing attacks targeting dynamic web pages [7]. In their first attack, called *direct timing*, an adversary directly measures response times from a website in order to obtain information about the website's state, e.g., the existence of an account with a certain username. The second attack, called *cross-site timing*, enables an attacker to learn information on the state of a user at a cross-origin website. By leveraging JavaScript to time cross-origin requests, the researchers show how an adversary can detect whether a user is currently logged in to a cross-origin website.

2.1 Threat Model

In the novel timing attacks we present, we employ a similar threat model as in Bortz' cross-site timing attack, i.e., an adversary provides an unwitting user with a malicious client-side script that performs timing measurements on a cross-origin website. By analyzing these timing measurements, adversaries can estimate the file size of an external resource, which often depends on the current state of the user. This consequently allows an attacker to infer information about a user's current state at the third-party website.

Since there is a plethora of different types of personal information shared across a large and diverse set of online web services, an attacker's interests may vary. An attacker may be interested in uncovering the unique identity of a user, which, as previous research has indicated [10, 14, 42], can be obtained by combining various bits of personal information of that user. In another attack scenario, the adversary could leverage a user's private information in order to create, or extend, a profile on the user, and display targeted advertising, derived from the obtained information. In their research, Nikiforakis et al. have shown how advertising companies are already using, often questionable, advanced techniques to track a user across different sites in order to uncover a user's interests [29]. The use of timing attacks could extend their ability to discover personal information, including age, location and interests of a user, posing an imminent threat to the online privacy of users.

3. WEB-BASED TIMING ATTACKS

When requesting a certain URL, a web server will often return a different resource based on the current state of the user. For example, when requesting the page of a private group on a social network, a user who is not a member of this group will receive a short error message, whereas group members are provided the full information of that private group. If an attacker is able to differentiate between the two types of responses, it is possible that the user's group membership is uncovered. Using web-based timing attacks, an attacker can differentiate between responses on the basis of their file size, as timing measurements are related to file size.

In order to detect the group membership of a user, the attacker can first perform a timing measurement on a resource with a predictable size, e.g., the page of a private group without any members. Using this value as a baseline, the attacker can discover whether the user is member of a specific group by collecting timing measurements of that group's page. If the outcome is comparable to the baseline, the user is not a member of the group. If, however, the difference in timing results is considerable, this indicates that the user received the full group information, and is thus a member of the group.

The accuracy with which an attacker can discover information on the state of a victim, is dependent on the correctness of the timing measurements. For example, if the attacker's timing measurements are based on the time required to download a resource, there are many factors that can negatively influence the correctness of the conducted measurements. Variations in latency, network congestion, and dropped packets are just a few examples which may prevent an attacker from successfully executing a timing attack. To improve the attack's effectiveness, an attacker can obtain multiple timing measurements, and subsequently apply statistical methods. However, by using multiple measurements, an adversary incurs a performance cost since the external resources need to be downloaded numerous times. Depending on the state of the network, these costs may become significant, preventing the attacker from inferring the state of the victim within a limited time frame. Moreover, a web server may start blocking requests when it detects a large number of requests originating from the same user, preventing an attacker from executing his attack.

In this section, we show how side-channel information exposed by modern browsers can be used to perform more accurate timing attacks. We exemplify this by describing four new types of timing attacks that can be employed by an adversary to estimate the size of a resource, and are independent of a victim's network stability. In addition, we compare these newly introduced attacks to the classic timing attacks where the network response time is measured, and show how some of these novel timing techniques can be combined to further improve their performance.

3.1 Experimental setup

In order to evaluate how well the different types of timing techniques perform, we conducted the following experiments. We set up a remote web server to serve four randomly generated HTML files, each with a different size: 50kB, 60kB, 150kB and 250kB. For each type of timing attack, we wrote a JavaScript program which would obtain 100 timing measurements. In the interest of minimizing the influence of a temporary network anomaly, or network congestion, the files were requested sequentially and in random order.

All experiments were executed in the latest version of Google Chrome, on a 2.5 GHz Intel Core i5 Macbook Pro with 16 GB RAM, which was placed in our campus' wireless network. As the network speed of our campus is significantly higher than the global average of 22.1 Mbps [30], the experiments were re-evaluated in a residential network. We found that the results of the residential-network evaluation were very similar to the ones from our campus. Thus, in this paper, we only present the results based on the measurements obtained at our campus.

To measure the time in our experiments, we used the `performance.now` function of the High Resolution Time API, which is present in all modern browsers and returns timing information with up to microsecond precision [24].

3.2 Basic web-based timing attack

The most straightforward way to perform a cross-site timing attack is to attempt to load an external resource that leaks information on the state of the victim as an image, and measure the time required to download the resource. More concretely, an attacker could use the JavaScript snippet as defined in Listing 1 to estimate the size of a victim's dashboard on the `example.org` website.

In this example, the browser will start downloading the user's dashboard as soon as the `src` attribute is set on the `Image` object. Since the browser does not know in advance whether the external resource is an image, it will first download its entire contents and, subsequently, it will try to display the resource as an image, but will fail to do so since the user's dashboard is an HTML resource. As a result,

```
var img = new Image();
img.onerror = function() {
  var end = window.performance.now();
  alert('Result: ' + (end - start));
};
var start = window.performance.now();
img.src = 'http://example.org/dashboard.php';
```

Listing 1: Basic web-based timing attack

the browser will fire an `error` event, which indicates to the attacker that he can stop his timing measurement.

Figure 1 (a) shows the distribution of the time interval between assigning the `src` attribute to an image, and the firing of the `error` event for the four files of different size. From this graph, it is clear that by using this traditional type of timing attack, an adversary requires multiple measurements to differentiate between a resource of size 50kB and one of 60kB. When there is a significant difference in file size, e.g., 50kB versus 150kB, it may be sufficient for an attacker to rely on the network download time to perform the timing attack. However, it should be noted that these measurements were acquired in optimal conditions: the browser had only a single tab open, very few other connections were made during the experiment, and the network jitter between the web server and end-user was minimal. In real-world scenarios, it is likely these optimal conditions are not met.

As the performance of this type of attack is heavily influenced by the stability of the network, we performed the same experiment on a mobile device, which was connected over a 4G network. Although the mobile device was placed in a fixed position, and performed no other networking operations, it becomes nearly impossible to distinguish the distribution of the two smallest files, as can be seen in Figure 1 (b). For the HTML files of 150kB and 250kB, the standard variance becomes considerably larger, which means that an attacker will require a significant number of timing measurements to reliably differentiate between two file sizes. As the window of opportunity during which an attacker can execute his attack is limited, and the average download time can range from multiple hundreds of milliseconds to seconds, the chance of a successful timing attack is considerably reduced.

3.3 Video parsing

To reduce the impact of network performance on timing measurements, we propose various new types of web-based timing attacks in the following sections. All the newly presented timing attacks make use of different timing side-channels that are present in most modern browsers. In this first attack, the side-channel leak is the time it takes the browser to parse a cross-origin document as a multimedia resource.

To support built-in media, HTML5 introduced two new elements: `<audio>` and `<video>` [27]. Using these elements, a website developer can directly include sound and video content in a way that is very similar to including an image, namely by assigning a link of the external resource to the element's `src` attribute. Similar to the `<img>` element, the new media elements also fire various events to indicate the progress of loading and playing a media file. More precisely, to indicate that a resource is currently being downloaded, a `progress` event is periodically fired. Similarly, a `suspend`

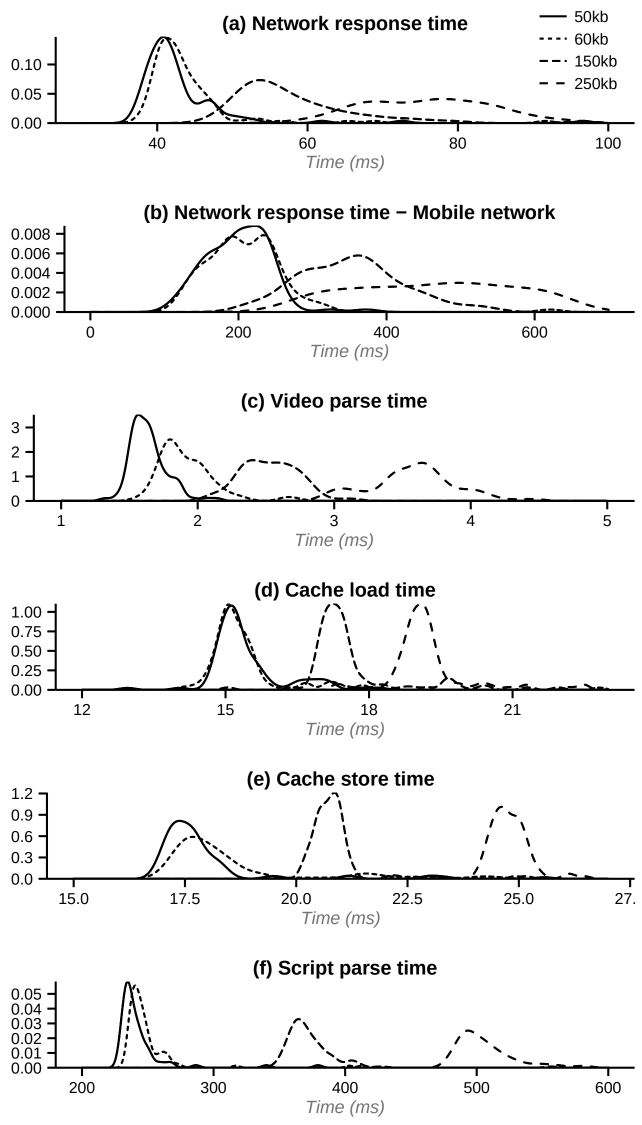

Figure 1: Distribution of load time, or time required to parse documents of four difference sizes

event is fired when the fetch is completed, to indicate the network state returns to the idle state.

Once a resource is fetched, the browser will parse its contents in an attempt to make it available for playing. As the external resources in web-based timing attacks usually consist of HTML content, parsing the content will obviously fail. Interestingly, the time required to parse a resource is dependent on the size of this resource. Consequently, browsers expose side-channel information that can be used by an attacker to perform a timing attack. It should be noted that Internet Explorer and Firefox only allow multimedia files to be played when these are served with the correct `Content-Type` header. Because these browsers immediately abort video processing as soon as the headers are received, it is not possible to perform this type of timing technique.

To analyze the variance in parsing time, we measured the time required to parse a resource, by measuring the time between the `suspend` and the `error` events. The latter event, just as with images, indicates a failure in the attempt to parse the file as a media resource. Similar to the previous experiments, we collected the parsing time for the four files, where each remote resource was parsed 100 times. The distribution of this timing information is depicted in Figure 1 (c). This graph shows that, especially for the 50kB and 60kB files, the timing measurements for each file are less distributed (have a smaller standard deviation) than when the resource download time is used as a timing measurement.

In summary, this type of timing technique exploits side-channel information caused by a difference in parsing time for multimedia elements. This type of attack is particularly useful for an adversary when attacking a user whose internet connection is unstable. As the attacker starts his timing measurement *after* the resource has been downloaded, the network connection has no influence on the timing process. This also means that delays imposed by the web server, as a countermeasure for classic timing attacks, are rendered obsolete. Coupled with the ability to perform measurements simultaneously, as parsing a resource only requires a few milliseconds, an adversary can use this attack to rapidly collect accurate timing measurements on different cross-origin resources.

3.4 ApplicationCache

To make websites available offline, web developers can make use of a recent browser feature named ApplicationCache [41]. By defining a manifest, the web author can define, among other things, which files should be permanently cached, making them available even when the user is no longer connected to the Internet. Normally the web server on which a resource is located, determines the caching policy, for instance by sending out a `Cache-Control` directive by the means of an HTTP header [13]. However, in the case of ApplicationCache, the server-side directives are overridden[1], allowing an attacker to force an external resource to be cached in the context of his attack page.

When a cached resource is requested, the web browser will read it from the hard disk, and make it available to the web page. Although reading out a small file may take less than a millisecond, we found that the size of a file still has a measurable influence on the time required to read it from the cache. As a result, this exposes side-channel information that allows an attacker to estimate the size of a file.

In our experiment, we defined a manifest as shown in Listing 2, which forces the four HTML files to be cached. When all files are cached, the AppCache mechanism fires an event named `cached`, after which we start our timing measurements. To reduce the impact of small measuring inaccuracies, we measured the time required to sequentially load the same file five times. This resulted in the four distributions as depicted in Figure 1 (d). The graph shows that, contrary to the previously discussed attacks, the relative standard deviation is small for all four files, including larger files. A major benefit of using this attack technique is that files only need to be downloaded once. As soon as the remote resources are placed in cache, the attacker can rapidly perform multiple timing measurements, with each taking only a few milliseconds.

[1]The ApplicationCache manifest will override all caching directives, except for the `no-store` directive of the `Cache-Control` header.

```
CACHE MANIFEST
CACHE:
http://example.com/50kb.html
http://example.com/60kb.html
http://example.com/150kb.html
http://example.com/250kb.html

NETWORK:
*
```

Listing 2: Example ApplicationCache manifest

To further improve his attack, an adversary could combine this attack with the video parsing technique. For smaller files, the latter gives more accurate timing information and because resources do not have to be downloaded for each measurement, the attacker can determine the size of a cross-origin resource with precision in a very short time frame.

3.5 Service Workers

Due to the increasing interest in developing web applications that can gracefully handle an offline environment, more and more developers have been complaining about the limitations of the AppCache mechanism [3]. To remove most of these limitations and to give the web developer programmatic control over the browser's cache, a new feature named Service Workers was developed [39]. Service Workers are defined as event-driven scripts which have a lifetime that is independent of the web page that created them. This means that even when a user closes the browser tab that started the Service Worker, a process could still be running in the background. This daemon-like quality of Service Workers can, unfortunately, become particularly useful for an adversary. Whereas an attacker traditionally had a very limited time frame in which he had to collect his timing measurements, this time frame can now be considerably extended by using Service Workers.

Since the main purpose of Service Workers is to make websites faster and available offline by intercepting network requests and controlling the cache, they do not have DOM access and can only use a limited API. Because of this limited environment, the video parsing attack defined in Section 3.3 can no longer be used when a victim closes the attacker's web page. One of the APIs that is available in a Service Worker environment, is the Fetch API [37], which allows a script to perform network requests. Unlike the XMLHttpRequest API, which also can be used to perform network requests, the Fetch API can make authenticated cross-origin requests without using the Cross-Origin Resource Sharing (CORS) mechanism. For security reasons, it's not possible to read out the response of this authenticated cross-origin request, but the time required to download the resource can still be used in a web-based timing attack. However, as we have shown in Section 3.2, a user's network conditions can heavily influence the performance of a timing attack.

By using another API in Service Workers, namely the Cache API, we show that it is possible to extract accurate timing measurements which are independent of the network stability, and give an indication of a resource's file size. As was previously mentioned, Service Workers enable a web developer to programmatically control the cache. This means that, for instance, a script could first download a specific resource, hold it in memory, and subsequently place it in the cache[2]. In the presented attack, we exploit side-channel information that is exposed by the time required to place a resource in the cache, and afterwards remove it.

To evaluate the performance of this timing attack, we calculated, for the four HTML files, the distribution of 100 timing measurements where each file was first placed in the cache and then removed, ten times in a row. The number of sequential additions and removals from the cache was picked to accommodate the speed of the hard disk, but could be fine-tuned by an adversary based on a brief benchmark on the victim's disk speed. The results of this experiment are displayed in Figure 1 (e), and show that the performance of this timing attack is slightly better than the ApplicationCache attack, as the relative standard deviation is small, and the distributions of different files show less overlap. This comes as no surprise, as both attacks exploit the side-channel information exposed by the disk activity, i.e., read operations for the ApplicationCache attack, and write operations for the Service Workers attack.

At the time of this writing, Service Workers are incorporated in the stable versions of Chrome and Opera, which covers more than 50% of the user-base according to StatCounter [33]. Implementations in other browsers will likely follow soon: Service Workers are already shipped in stable versions of Firefox [26] but require manual activation, and Internet Explorer has also shown interest in providing them [25]. This means that in the near future, all users who operate a modern browser, can be victimized by web-based timing attacks occurring in background processes.

3.6 Script parsing

Whereas the previous timing attacks originate from abusing relatively new HTML5 APIs, the script parsing attack serves as an example that timing side-channels may also be present in long-established browser technologies. As the name already suggests, the timing side-channel in this attack is introduced by tricking the browser into parsing a remote resource as a JavaScript file. This can be easily done by creating a `script` element, and assigning the `src` attribute to the location of the remote resource. When this element is added to the DOM, the browser downloads the resource and attempts to parse and execute it as JavaScript[3]. An example of how an attacker would measure the time it takes to parse a script is shown in Listing 3. In most attack scenarios, the external resources of which the attacker wants to estimate the size, are not valid JavaScript files. For instance, trying to execute a file that starts with `<html>`, will throw a `SyntaxError` on the first line, preventing the rest of the "script" from executing. Nevertheless, the resource still needs to be read into memory and undergo several operations in order to be parsed. We found that the time it takes for this process to complete, is dependent on the size of the resource that needs to be parsed, thus exposing a timing side-channel.

In comparison to parsing a resource as a video, the speed by which scripts are parsed, is significantly higher. As a result, it becomes impractical to measure this for smaller files, even with the High Resolution Time API. However, most modern browsers (with the exception of Firefox) use an op-

[2]The Cache API allows any resource to be stored, even if the `no-store` directive is present in the `Cache-Control` header

[3]When the value of the `X-Content-Type-Options` header is set to `nosniff`, Chrome and Internet Explorer will not parse nor execute the file as a script

```
window.onerror = function() {
    var d = performance.now() - window.start;
    console.log('parsing done', d)
}
var s = document.createElement('script');
document.body.appendChild(s);
s.onload = function() {
    console.log('script downloaded');
    window.start = performance.now();
}
s.src = 'http://example.com/resource';
```

Listing 3: Script parsing example

timization that when the same resource is requested multiple times within a short time interval, only a single request is made. Consequently, this optimization can be used to force the browser to parse a script multiple times requiring only a single GET request. To obtain a measurement, we first create a number of script elements, register an event listener for the `load` event on each element, and add them all to the DOM. Next, we register for the `error` event on the `window` object (which is where the `SyntaxError` event will be fired), and finally start parsing the remote resource by assigning the `src` attribute on all script elements simultaneously. We compute the total parsing time as the interval between the first `load` event and the last `error` event.

We applied the same performance evaluation as with the other attacks, and calculated the distribution of the time it took to parse each file 50 times. This number of iterations was chosen to optimally suit the performance of the tested device. The results of this experiment are depicted in Figure 1 (f), and show that this attack performs reasonably well, especially for smaller files. While the time required to obtain a single measurement for this attack is relatively high, it should be noted that the measurements are independent of the victim's network condition.

3.7 Performance evaluation

In the previous sections, we discussed four different types of web-based timing attacks that exploit side-channel information exposed by browsers, and briefly analyzed their performance in comparison to a basic timing attack that relies on the download time of a resource. To evaluate the potential of the newly presented timing attacks in more detail, we performed an additional experiment using a similar setup as the one discussed in Section 3.1. The goal of this experiment was to evaluate, for each type of timing technique, the time required by an attacker to successfully make a distinction between two resources of different sizes.

First, HTML files were created with a file size ranging from 100kB to 200kB, in 5kB increments. We compared timing measurements of each file to the 100kB file, which was used as a baseline, by alternately extracting timing information from the baseline file, followed by a timing measurement of the larger file. This process, which we limited to 60 seconds, was then repeated for each timing technique.

In order to estimate the time required for an adversary to perform a successful attack, we first calculated the smallest number (`n`) of timing measurements required to perform a timing attack with an accuracy of 95%. The accuracy was calculated as follows: for each group of `n` timing measurements of the baseline, we compared the mean of those measurements to the mean of the corresponding group of the tested file. For example, for the basic timing attack

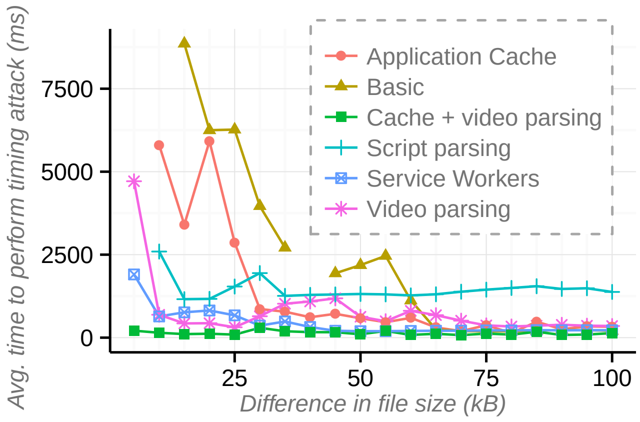

Figure 2: The average time required to perform a cross-site timing attack with 95% accuracy, for each type of web-based timing technique.

where the baseline file of 100kB was compared to a file of 155kB, we collected a total of 600 timing measurements during one minute, namely 300 for each file. For a group size of 13 measurements, we found that the mean of the measurements for the baseline file was smaller than the mean of the corresponding group of the 155kB measurements in 22 of the 23 groups, leading to an accuracy of 95.65%. For all groups with a smaller value of `n`, we found the accuracy to be less than 95%. Finally, we calculated the required time to perform a successful timing attack as the average time required to collect all measurements for the minimum group size.

The results of this experiment are shown in Figure 2. While the timing experiments were conducted in a controlled environment, on a relatively stable network, the results show that using a basic timing attack, an attacker would be unable to differentiate files with a difference in size of less than 15kB. Interestingly, for the 140kB file, the basic timing attack failed as well, which was most likely caused by a brief irregularity in the network. This again shows that performing a web-based timing attack by collecting timing measurements based on the network download time, can be a very unreliable process. Furthermore, the overall results indicate that the four newly introduced timing attacks substantially outperform basic web-based timing techniques. Especially when the difference in file size is small, the newly introduced timing techniques show a manifold increase in terms of performance.

As was mentioned earlier, our individual timing techniques can be combined to further improve the performance of a timing attack. In Figure 2, we also show the results of a timing attack where we first force the caching of the remote resources and then apply our video parsing attack. This results in a significant performance increase, where even the smallest difference in file size could be detected in approximately 200ms, including the initial download time. Combined with the ability to collect timing measurements in parallel without loss of accuracy, the use of our newly proposed techniques makes timing attacks much more viable in real-world situations.

Other devices.

In order to validate that our proposed attacks work on multiple systems, we performed the same experiment on a variety

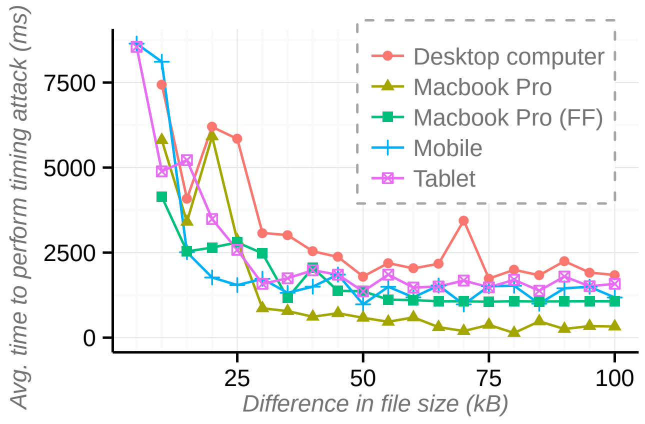

Figure 3: The average time required to perform an AppCache-based timing attack with 95% accuracy, for different platforms.

of devices: a mid-level desktop computer running Ubuntu, the Macbook Pro from previous experiments, once using Chrome and once using Firefox, a Motorola Moto G smartphone, and a Samsung Galaxy Tab 3 tablet. Due to space limitations, in Figure 3, we only show the results of the AppCache-based timing attack. All other attacks behave in a similar fashion.

In general, we found that the timing techniques demonstrated only a minor variation in performance among the different browsers, operating systems, and devices. As a result, the timing attacks presented in this paper can be leveraged to obtain sensitive information across a wide range of browsers and devices. Although the average time required to differentiate between two files is comparable between different platforms, we found that the time required to obtain a single measurement differed considerably. For instance, the average time required to load a 100kb resource five times from the cache was almost twice as high on the smartphone as it was on the laptop (27.76ms versus 15.84ms). While it takes longer to obtain a single measurement, the measurements from slower devices are generally more accurate, and thus fewer are required for a successful attack.

3.8 Discussion

In our research, we analyzed various browser functions that handle external resources for the presence of timing side-channels and discovered several cases that could be exploited to leak timing information. Another example of this, is the side-channel information exposed by the Navigation Timing API [38]. This API provides timing information for requested resources and, due to its design, can be used to determine whether a redirection chain was followed. More precisely, an attacker can compute the time between the initialization of the request, e.g., when he assigns the `src` attribute to an `Image` object, and the `fetchStart` property of the corresponding `PerformanceResourceTiming` entry. The latter contains the time when the browser started fetching the resource, which is usually a fraction of a millisecond after the request was triggered. However, when a redirection chain is followed, the value is set to the time the fetch algorithm for the last resource in the chain was initiated. As a result, the time between the initialization of the request and the `fetchStart` property will be considerably higher, allowing the attacker to determine whether a redirection chain was followed using just a single request. In their research, Lee et al. have shown how this information can be leveraged to uncover the login-status of users at a cross-origin website [22].

The wide variety of the discovered timing side-channels, all of which are resilient from network irregularities, serves as a strong indicator that modern browsers are lacking structural defense mechanisms to adequately protect against the exposure of timing information. As such, we expect that, along with the exponential growth of browser functionalities and accompanying APIs, new timing side-channels will arise.For an adversary, it is sufficient to be able to measure the time required to handle a remote resource, either by storing or retrieving it from the cache, or parsing it. Consequently, any browser feature that accommodates these requirements may expose a new timing side-channel.

4. REAL-WORLD TIMING ATTACKS

In the previous section, we showed how timing attacks can be used to estimate the size of an external resource. In this section, we discuss how these techniques can be applied in real-world scenarios, and how an adversary can use these attacks to extract personal information from users. We focus mainly on social networking websites and describe different attack scenarios which are based on the different functionalities offered by these services. Previous research [42] has shown that the information about group membership of a user can lead to the unique identification of that user. We extend this work by analyzing other types of personally identifiable information that can be exfiltrated using timing attacks. This information could then be used to either uniquely identify a user, or to create a profile of the user's age, gender, location, and interests. The latter is particularly useful for advertising companies who are always looking for opportunities to improve targeted advertising, and, as illustrated by previous research, often use questionable techniques to reach this goal [2, 15, 29].

As web developers tend to tailor the response for certain endpoints to the current state of a user, web-based timing attacks can be performed on a large and varied set of websites. To show some of the potential consequences of timing attacks in the modern web, we present various real-world attacks on several of the most popular social networks. While, due to abundance of personal information that users share on these services, we mainly focus on social networks in the presented attack scenarios, the proposed timing attacks can be applied to other types of online services as well.The list of possible attack scenarios described below should be seen as an indication of how widespread timing-related vulnerabilities are.

Ethical considerations.

To assess the presence of timing vulnerabilities in the wild, and quantify the effectiveness of our newly proposed attacks when compared to the classic timing attack, we cannot avoid searching for vulnerabilities in real world sites. Note that all vulnerabilities discussed in the following sections were discovered by manually interacting with a website and *never* resorting to the use of, potentially intrusive, automated vulnerability scanners. All cross-site requests were performed against our own accounts, thus real users where never exposed to our attacks. It is also important to point out that our attacks, as far as a server is concerned, are merely cross-site requests, thus the tested web applications are never exposed to any kind of malicious input. Given the above, we believe that our timing attacks did not have *any* adverse effects, neither on the tested services, nor on their users.

4.1 Facebook: Age, Gender, and Location

Facebook has approximately 1.4 billion active users [11] making it one of the largest online web services. Next to user profiles, Facebook also offers the possibility to represent companies and brands by the means of so-called “pages”. These pages are similar to a user’s profile in the sense that, just like a user, a page can update its status, and interact with others. A page’s status update will be broadcasted through the social network to everyone connected to the page, i.e., to every user that “follows” the page. For branding purposes, status updates can be limited to a particular target audience, for instance users between the age of 20 and 30, or only female users from a specific location. When visiting the permanent link (also known as permalink) of the status update, users who are not part of the target audience are presented with a static page which states the content is not available. As the size of the static page is different from the size of the page containing the actual status update, this exposes side-channel information allowing an adversary to determine whether a user belongs to a specific audience. We found that the file size of a visible post (240kB) is sufficiently different from the size of a post when the user is not part of the target audience (163kB), allowing an adversary to perform a successful timing attack in a few milliseconds, using our newly proposed timing attacks.

To verify this claim, we set up a Facebook page and made six posts, each targeted to people who fall in a specific non-overlapping age range. As a result, only a single post was visible to the victim user. For both the basic timing attack as well as our novel attack technique using Service Workers, we collected timing measurements for each post during 15 seconds. The time interval was limited because generally, an attacker only has a limited window of opportunity during which he can perform a timing attack. Furthermore, the attacker is likely to be interested in other private information on the user as well, meaning he will have to perform multiple attacks within this limited time frame. The timing measurements, displayed in Figure 4, clearly indicate that the measurements acquired using the basic timing attack are too variable to reliably determine the age of a victim. It should also be noted that these experiments were conducted in optimal network conditions, and network jitter may further decrease the basic attack. The measurements for the unauthorized posts in the Service Workers attack are, except for a few outliers, consistently lower than the post that is only visible to the age-range to which the victim belongs (23-32). These results serve as an additional indicator that an attacker can obtain sensitive information much faster by using the newly introduced timing attacks.

4.2 LinkedIn: Contact Search

LinkedIn is a business-oriented social network with over 347 million users, and is mainly used for professional networking [23]. Similar to other social networking services, users on LinkedIn can create bi-directional relations with others, which are called connections. In order to browse through your contacts, LinkedIn offers the functionality to filter connections based on their name, location, employer, or

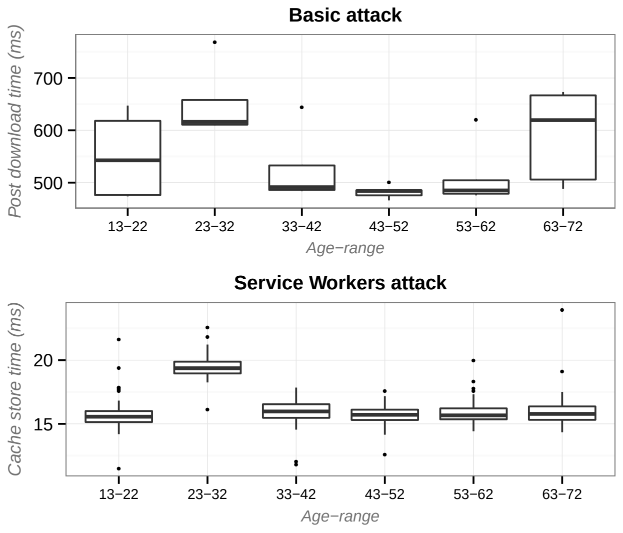

Figure 4: Timing attack against Facebook age-limited posts.

even their current job title. If the intuitive notion that users connect with colleagues, business partners, or friends in the same geographic location, holds true, then it is possible to infer a user’s location, employer, and other private aspects, on the basis of the number of connections that match certain filters. For instance, if a user has over 100 connections originating from Germany, and only a handful from other countries, it is likely that the user, just as his connections, is living in Germany.

We found that the matching contacts for a certain query are sent as a JSON stream, in response to an XMLHttpRequest request. For each connection that matches the query, we found that the response size grows approximately 0.5kB. As the JSON resource does not contain a token specific to the user, and can be accessed using a normal GET request, we found that, by using timing attacks, it is possible for an attacker to estimate the number of connections returned for a certain filter. By combining the outcome of several queries, the adversary can learn the geographic distribution of a user’s professional network, with a granularity of country or even city. In a similar fashion, the adversary could extract information on the companies the victim works with, allowing him to leverage this information to, for instance, launch a personalized phishing attack.

4.3 Twitter: Protected Accounts

With 288 million active users, and 500 million Tweets sent per day, Twitter is one of the largest online microblogging networks [36]. By default, Twitter makes all Tweets publicly available. However, Twitter also provides the possibility to make your account protected, in which case Tweets are only visible to the list of approved followers. This means that the visibility of protected Tweets depends on the state of the user: the protected Tweet is only shown if the user follows its author. By exploiting the difference in size returned for a protected account’s profile page, an adversary can detect which protected Twitter accounts the victim is following.

In their research, Wondracek et al. [42] describe how the combination of the groups a social-network user is member of, can be used in a de-anonymization attack. Using a similar approach, but replacing the notion of “groups” with "protected Twitter accounts", an adversary can use timing attacks to compose the list of protected accounts a user is following, and subsequently use that information to find the user's unique identity. Contrary to Wondracek's proposed attack, where history stealing techniques are leveraged to extract group membership information, our attack is based on the user's state on the social network. As such, our attack provides much more reliable results since it does not rely on a user's browsing behavior.

Composing the complete list of protected accounts a user is following, poses the major practical limitation in this attack. Over 10% of Twitter users, i.e., approximately 30 million users, have opted to protect their Tweets [6], and checking these one by one would be impractical. However, we argue that a motivated attacker could employ a more sophisticated algorithm, e.g., by first checking accounts with the most followers, to considerably improve this attack. By using the newly introduced timing attacks, an adversary can easily reveal a user's following information since there is a difference of more than 100kB in resource size. Interestingly, classic timing attacks will be highly inaccurate since after the `gzip` compression (typically used by web servers to reduce the size of HTTP responses), there is only a difference in file size of approximately 5kB.

4.4 Google and Amazon: Search History

So far, we discussed various attack scenarios on several social networks, which leverage cross-site timing techniques. To demonstrate that the consequences of timing attacks are not limited to social networks, in this section, we describe an attack on the most-used online service, namely the Google search engine. With over a trillion searches per year, the majority of people on the web are using Google's search results as a starting point for browsing websites related to their interests. In addition to responding to new search queries, Google provides the ability to navigate through all search queries users have made in the past, and shows which results they have clicked.

One way of navigating through the search history is by searching for a specific keyword, which will return up to 1,000 search queries and corresponding search results that match that keyword. The resource containing the results is an HTML file that can be acquired by making a GET request. Naturally, this resource grows larger as more results are returned. An adversary is able to extract information on a user's interests, based on the response size of various queries. In Figure 5, we show the measurements obtained during two timing attacks that target various keywords. We find that the timing measurements for the basic attack are distributed unequally, preventing an attacker from learning the user's interests within the allotted time frame. However, when Service Workers are leveraged to measure the time required to place a resource in the cache, an attacker can quickly estimate the number of search results that are returned for a specific query.

This particular issue is not just limited to Google's search engine, but can also be applied to other online web services that provide the functionality of viewing one's own browsing history. For example Amazon, one of the largest e-commerce services, offers users the ability to filter their own browsing history based on product category. We found that the page offering this functionality is vulnerable to timing attacks,

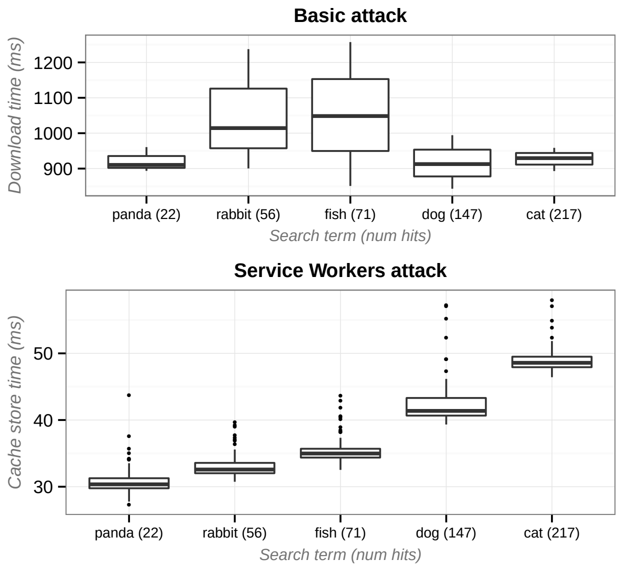

Figure 5: Timing attack against several queries in Google Search History.

which can provide an adversary with valuable information that could be used for targeted advertising.

5. DISCUSSION

In the previous sections we introduced various timing techniques, and described how these can be leveraged to discover a broad range of personal information in a variety of social networks and other online services. We found that these techniques can be applied to the majority of modern web browsers, and that a large fraction of the most popular online services fail to protect their users against cross-site timing attacks. Defense mechanisms that have been proposed to counter timing attacks include adding a random delay that requires an attacker to obtain more timing measurements [20], or implementing a fixed response time for all server responses [7]. To reduce the performance impact of these countermeasures, other researchers have proposed to just keep the execution time of sensitive processes fixed [28], or to add a fixed and unpredictable delay to the server's response time [31].

These defense mechanisms can adequately prevent both direct timing attacks, i.e., attacks where the adversary learns secrets about the state of the website, and basic timing attacks, where the adversary learns secrets about the state of a particular user on a third-party website. Unfortunately, they are fully bypassed by our newly introduced cross-site timing attacks, as our attacks exploit various browser mechanisms which expose information on the size of an external resource. In our timing attacks, a resource is downloaded just once, and consequently delays imposed by the web server do not impact the cross-site timing performance.

It could be argued that the side-channel information, which is used in the new timing techniques, is exposed by modern browsers, and therefore these browsers should be patched in order to prevent this from happening. A straightforward solution would be to add a random delay to the firing of events. As a result, an attacker would have to collect more measurements and apply statistical methods in order to accurately determine the size of a resource. However, since hundreds of measurements can be obtained in just a few seconds, and because this solution would negatively impact the performance of many websites, we do not consider this a viable solution. Alternatively, browser vendors could opt to fix the time when events are fired to the worst-case execution time (WCET). However, in this case an attacker could still use performance information, e.g., by continuously monitoring the writing speed, to infer the time it took to perform a certain action, e.g., writing a resource to the cache.

In light of the arduous task of eradicating timing side-channels from modern browsers, we argue that a server-side solution is more appropriate. In essence, the timing side-channels exist because the browser allows web pages to include cross-origin resources that were not meant to be included by an untrusted party. As such, mitigating the timing attacks presented in Section 3 has various similarities with protecting against CSRF attacks. For both CSRF and timing attacks, an adversary will typically need to trick the browser in sending out specific requests to a vulnerable website. In contrast to CSRF attacks, where the requests result in a state-change of the logged in user, the requests that are sent when a timing side-channel is exploited, are aimed for resources that simply contain state-specific content. Consequently, blocking illicit cross-origin requests allows a website administrator to prevent an adversary from leveraging timing attacks against his website.

A well-known defense strategy against CSRF attacks, proposed by Barth et al. [5], is to analyze the `Referer` and `Origin` headers for endpoints that may trigger a modification of the user's state. Unfortunately, it is not possible to straightforwardly apply the same technique to prevent timing attacks, because landing pages, i.e. web pages the user lands on after clicking a link, may also contain state-specific content. When navigating to such a landing page, the `Referer` header in the request will be set to the URL of a remote, and possibly untrusted, web page. As a result, it becomes impossible for a website to differentiate between legitimate requests to a landing page, and requests that were triggered from a malicious web page.

We propose to employ a placeholder web page as an additional mechanism to address this obstacle. More precisely, when the `Referer` header is missing, or originates from an untrusted domain, the placeholder is served instead of the actual content. When loaded, this placeholder page initiates an authenticated XHR request to the URL that was initially requested. Since the `Origin` header in this request is set to the same domain, the web server can verify that this is a same-origin request and will send the actual resource. When the XHR request completes, the placeholder writes the content to the DOM, and the actual page content is loaded. All requests that originate from this page load, e.g. images that are included by `img` elements, will have the `Referer` header set to the current URL, and thus will be permitted by the web server. Because only same-origin requests are allowed, an attacker will not be able to trick a victim into loading a state-dependent resource from the protected website.

Limitations.

As Barth et al. indicate, requests that lack a `Referer` header, something that may happen out of privacy concerns, pose a conundrum: either the website accepts the request, rendering the defense ineffective, or rejects it, which may prevent legitimate users from accessing the website. In our defense scenario, the check of the `Referer` header can only pass if the domain matches the protected domain. As such, the `Referer` header only needs to be present in same-origin requests. Additionally, because no sensitive information is leaked in these same-origin requests, there is no reason for browsers, extensions or network configurations to omit the `Referer` header for these requests.

By design, this defense mechanism prohibits cross-origin web pages from including resources of the protected website. However, several legitimate cases exist where a website may want to allow certain resources to be included by other websites. This poses a problem when a non-HTML resource, e.g. a dynamically generate image, is included cross-origin, and lacks the `Referer` header. In this case, the defense mechanism will be unable to uncover which domain triggered the request, and will not be able to serve the placeholder, as no HTML content will be rendered. As a result, the including website will need to resort to an alternative way of loading the image, for instance by using XHR in combination with the CORS mechanism.

6. RELATED WORK

Timing attacks are one of the oldest types of side-channel attacks in computer systems, first introduced almost two decades ago. In the context of the web, previous work focused mainly on the network download time [7], or the presence of certain resources in the cache [12] as timing side-channels. We believe this paper is the first to present web-based timing attacks that leverage timing side-channels in various browser features to estimate the size of a cross-origin resource, regardless of a victim's network condition. In addition, we extend prior research on the potential privacy-intrusive consequences of timing attacks. Previous research has indicated that cross-site timing attacks can be used to obtain the number of products in a victim's shopping basket, or to uncover the websites a victim recently visited, or is currently logged in at. In our research, we extended prior work by describing how an adversary can employ various attacks in the modern web to obtain private and personal information, such as age, gender, location, and personal interests, based on the state of an unwitting visitor. In the rest of this section, we review related work on all types of timing attacks in the context of the web, and describe other side-channel leaks in browsers that threaten a user's security and privacy.

6.1 Timing attacks in the web

In their research, Bortz et al. introduced the notion of two types of timing attacks: direct timing attacks, and cross-site timing attacks [7]. Other researchers mainly focused on the former type, where an adversary tries to extract secret information from the web server, e.g., the existence of a specific username. Crosby et al. showed that a timing difference as low as 20µs on a server-side process can be reliably distinguished over the Internet. The ability to obtain highly accurate timing information has given rise to numerous attacks that rely on remote timing information. The goals of these attacks range from breaking cryptographic systems, e.g. by extracting private keys from an OpenSSL-based web server [8], to fingerprinting the rules of Web Application Firewalls [32].

Moreover, researchers have shown that cross-site timing attacks can be employed to list network-enabled devices on the victim's local network [19]. An adversary could subsequently use this information to fingerprint the user, or to penetrate vulnerable devices, for instance by using CSRF attacks. Contrary to these attacks, where the main focus is to breach the security of machines that are generally only available over the local network, Felten and Schneider proposed various cross-site timing attacks that can be used by adversaries to obtain information on a victim's browsing history [12]. Based on the reduced loading time of cached resources, the researchers found that it is possible for an attacker to uncover whether a certain resource is present in a victim's cache. As cached resources originate from the websites a user recently visited, the adversary is able to discover a victim's browsing history. Although this type of attack has been known for over 15 years, relatively few changes were made to the browser environment to mitigate this issue. Just recently, Jia et al. showed that by using exactly the same techniques, adversaries can launch geo-inference attacks to discover a victim's geographical location without his consent [18]. The geo-inference attack exploits the fact that various web services that are trusted by the victim and know his location, cache location-specific resources. As a result, an adversary can discover the victim's location by analyzing which of these resources are cached.

Next to the network response time and server-side processing time, researchers discovered a variety of attacks that leverage the time required by the browser to complete certain computations. In 2013, Kotcher et al. found that after applying CSS filters on framed documents, the time required to render the document becomes related to its visual content [21]. As a result, this attack allowed adversaries to read out pixels from cross-origin documents in case framing was not explicitly forbidden. Similarly, Paul Stone found that applying SVG filters instead of CSS filters yielded the same results [35].

6.2 Browser side-channel leaks

Due to the complex design and intricate implementations of browsers, it comes as no surprise that researchers frequently discover unintended behavior that often leads to a leakage of users' private information, or that can be used to bypass the building block of security in modern web browsers, namely the Same-Origin Policy.

One of the oldest, and most well-known side-channel leaks in browsers, is the history sniffing attack first introduced in 2002 [9]. By applying CSS styles to visited links and subsequently querying the computed style in JavaScript, an adversary could easily determine whether a victim had previously visited a certain link. By means of an empirical study on the 50,000 most popular websites, Jang et al. discovered the clandestine usage of these history sniffing attacks on 46 websites [17]. This pressured browser vendors into adopting an effective countermeasure that restricted the CSS directives that could be used in the `:visited` pseudo-class [4]. Shortly thereafter, researchers discovered that even with this mitigation in place, history detection techniques were still possible, either by using the aforementioned timing attacks that leverage SVG filters [34], or by user-interaction [40].

Next to attacks targeting a user's browser behavior, researchers have found that the inherent behavior of certain browser features can allow an adversary to uncover a user's private information at a cross-origin website. For instance, Heiderich et al. discovered that by leveraging various CSS and HTML features, adversaries can exfiltrate sensitive information, such as CSRF tokens [16]. Moreover, Lee et al. found that the intrinsic behavior of the ApplicationCache mechanism can be used to uncover the status code that is returned for a cross-origin resource [22]. Consequently, this allows an adversary to obtain sensitive information when the resulting status code for certain endpoint is based on the user's state. The authors showed how these attacks could be used to discover web servers on the local network, and to detect the login-status of a user at various websites. Interestingly, our proposed countermeasure, which aims to prevent illicit cross-origin requests, can also be used to deflect the ApplicationCache attacks proposed by Lee et al.Correspondingly, their defense mechanism, i.e., providing more control to website administrators over the cache-ability of a resource, can be applied to restrict the two cache-based timing techniques. As there is a variety of browser features, including features unrelated to the browser cache, that may leak timing information, we conjecture that a more systematic approach is required to thwart this type of side-channel attacks.

7. CONCLUSION

In this paper, we propose several new timing techniques for estimating the size of cross-origin resources. These attacks exploit the side-channel information that is exposed by the time required by a browser to process a resource, either by parsing it, or by involving it in caching operations, i.e. storage or retrieval. Because the timing measurements start *after* the resource has been downloaded, the side-channel attacks do not suffer from the limitations of traditional timing techniques, and can thus be used by adversaries to obtain more accurate timing measurements, regardless of the victim's potentially unfavorable network conditions. We show that these attacks can be applied on various platforms, posing an imminent threat to an extensive amount of web users. Using five real-world attack scenarios, we illustrate how attackers can leverage our novel timing techniques against a variety of online web services, allowing them to extract private data that a victim shared with trusted services.

Overall, our findings indicate that cross-site timing attacks pose an imminent threat to the privacy of online users. As the side-channel leaks exploited in the novel timing techniques are inherent to the design of browsers and the web in general, we conjecture that a systematic client-side countermeasure would require structural changes to the browser architecture. Due to the complexity of modern browsers, a complete mitigation against all side-channels leaks appears unlikely, pointing towards the need for CSRF-like countermeasures at the server-side that hide the size of a resource from cross-site attackers.

Acknowledgments

We thank the anonymous reviewers for the valuable comments. For KU Leuven, this research was performed with the financial support of the Prevention against Crime Programme of the European Union (B-CCENTRE), the Research Fund KU Leuven, the IWT project SPION and the EU FP7 project NESSoS. For Stony Brook University, this work was supported by the National Science Foundation (NSF) under grant CNS-1527086.

8. REFERENCES

[1] Bose v. interclick, inc., 2011.

[2] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689. ACM, 2014.

[3] J. Archibald. Application Cache is a douchebag. `http://alistapart.com/article/application-cache-is-a-douchebag`, May 2012.

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

[5] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM, 2008.

[6] Beevolve. An exhaustive study of Twitter users across the world. `http://www.beevolve.com/twitter-statistics/`, October 2012.

[7] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 621–628. ACM, 2007.

[8] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[9] A. Clover. CSS visited pages disclosure, 2002.

[10] X. Ding, L. Zhang, Z. Wan, and M. Gu. A brief survey on de-anonymization attacks in online social networks. In *CASoN*, pages 611–615, 2010.

[11] Facebook. Company info. `http://newsroom.fb.com/company-info/`.

[12] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 25–32. ACM, 2000.

[13] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol–HTTP/1.1, 1999. *RFC2616*, 2006.

[14] H. Gao, J. Hu, T. Huang, J. Wang, and Y. Chen. Security issues in online social networks. *Internet Computing, IEEE*, 15(4):56–63, 2011.

[15] D. Goodin. Marketer taps browser flaw to see if you're pregnant. `http://www.theregister.co.uk/2011/07/22/marketer_sniffs_browser_history/`, July 2011.

[16] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk. Scriptless attacks: Stealing the pie without touching the sill. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 760–771. ACM, 2012.

[17] 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*, pages 270–283. ACM, 2010.

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

[19] M. Johns. Exploiting the intranet with a webpage. `http://web.sec.uni-passau.de/members/martin/docs/070906_HITB_Martin_Johns.pdf`, September 2007.

[20] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer, 1996.

[21] 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*, pages 1055–1062. ACM, 2013.

[22] S. Lee, H. Kim, and J. Kim. Identifying cross-origin resource status using Application Cache. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS'15)*, 2015.

[23] LinkedIn. About LinkedIn. `https://press.linkedin.com/about-linkedin`.

[24] J. Mann. High Resolution Time. *W3C recommendation*, 2012.

[25] Microsoft. modern.IE - platform status. `https://status.modern.ie/serviceworker`.

[26] Mozilla Developer Network. ServiceWorker api. `https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker_API`.

[27] Mozilla Developer Network. Using HTML5 audio and video. `https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_HTML5_audio_and_video`.

[28] Y. Nagami, D. Miyamoto, H. Hazeyama, and Y. Kadobayashi. An independent evaluation of web timing attack and its countermeasure. In *Availability, Reliability and Security (ARES)*, 2008.

[29] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and privacy (SP), 2013 IEEE symposium on*, pages 541–555. IEEE, 2013.

[30] OOKLA Net Index. Household download index. `http://www.netindex.com/download/allcountries/`, February 2015.

[31] S. Schinzel. An efficient mitigation method for timing side channels on the web. In *2nd International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE)*, 2011.

[32] I. Schmitt and S. Schinzel. WAFFle: Fingerprinting filter rules of web application firewalls. In *WOOT*, pages 34–40, 2012.

[33] StatCounter. Top 5 desktop browsers on jan 2015. `http://gs.statcounter.com/#desktop-browser-ww-monthly-201501-201501-bar`, January 2015.

[34] P. Stone. Bug 711043 - (CVE-2013-1693) SVG filter timing attack. `https://bugzilla.mozilla.org/show_bug.cgi?id=711043`, December 2011.

[35] P. Stone. Pixel perfect timing attacks with HTML5. *Context Information Security (White Paper)*, 2013.

[36] Twitter. Company info. `https://about.twitter.com/company`, February 2015.

[37] A. Van Kesteren and WHATWG. Fetch. `https://fetch.spec.whatwg.org/`, January 2015.

[38] W3C. Navigation Timing. `http://www.w3.org/TR/navigation-timing/`, December 2012.

[39] W3C. Service Workers. `http://www.w3.org/TR/service-workers/`, February 2015.

[40] 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*, pages 147–161. IEEE, 2011.

[41] WHATWG. Offline web applications. `https://html.spec.whatwg.org/multipage/browsers.html#offline`, January 2015.

[42] 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*, pages 223–238. IEEE, 2010.