学术新星计划暑期专项

跨站点泄露漏洞

田正祺（George Tian）　软件 01　2020080095

指导教师：陈建军
小导师：王楚涵

摘要

浏览器是最广泛使用的软件之一，因此保证浏览器的安全性具有非常高的重要性。浏览器安全的一个方面是用户隐私的保护，即防止用户信息被泄漏给第三方。跨站点泄露漏洞是一类旁道攻击，当用户访问攻击网站后，目标网站上的某些信息可以被泄露给攻击者。此研究报告介绍了跨站点泄露漏洞的形式模型，其次分析了若干种泄露的工作原理以及防范措施。最后聚焦一类影响到 Chromium 浏览器的漏洞，并分析漏洞的工作原理以及修复方法。

关键词：浏览器安全、跨站点泄露漏洞、Chromium。

A web browser is one of the most commonly-used types of software, therefore the task of ensuring browser security is of high importance. One aspect of browser security is user privacy, that is preventing the unintended leakage of user data to third parties. Cross-site leaks are a class of vulnerabilities that leaks certain pieces of information from a target webpage when the user visits an attacker's webpage. This report introduces a formal model for XSLeaks, then analyzes multiple classes of vulnerabilities along with possible prevention methods. Finally, focus is drawn to a series of bugs affecting the Chromium browser, and their attack and mitigation strategies are analyzed.

Key words: web browser security, XSLeaks, Chromium

1 引言

在现在日益依赖技术的世界中，用户可以有意或无意地将个人信息上传到互联网。因此恶意行为者通过互联网窃取个人信息的激励以及潜在的收益也日益增加。因此网络安全以及线上的个人隐私的维护是安全研究人员与恶意攻击者之间无尽的战争。

跨站点泄露漏洞（Cross-site leaks, XSLeaks）[1] 是一类浏览器中的旁道攻击。当用户访问攻击者控制的网站或包含攻击者的恶意代码的网站（统称“攻击网站”），攻击者就可以通过使用跨站点泄露漏洞而获取用户在另一个网站（统称“目标网站”）上的信息。

跨站点泄露漏洞的诞生是由跨来源资源共享（CORS）策略而引发的。某个资源的来源是由它的方案 (协议)，主机 (域名) 和端口而组成。两个资源是同源的当且仅当以上三个元素都相等[2]。服务器向浏览器发送的响应中的 CORS 头指示浏览器应该如何设置响应中的资源的访问全。若不许跨站点的网站或脚本读取资源，则攻击者必须使用旁道攻击技巧——跨站点泄露漏洞——窃取信息。

2 形式模型

Knittel et al.[3] 定义了跨站点泄露漏洞的形式模型。根据他们的定义，跨站点泄露漏洞是输出一个比特 b' 的函数 xsl：

$$b' = xsl(sdr, i, t)$$

其输入为：

- sdr：依赖于状态的资源（state-dependent resource），而它是二元组 $(url, (s, d))$，其中 $(s, d) \in \{(s_0, d_0), (s_1, d_1)\}$：
  - url：目标资源的 URL，可以包括若干个查询参数，比如 https://www.google.com/search?q=xsleaks。
  - $S = \{s_0, s_1\}$：网站的两个状态的集合。
  - $D = \{d_0, d_1\}$：网站的行为的差异，依赖于 s_0 和 s_1。
- $i \in I$：包含方法（inclusion method），即如何将攻击网站向 sdr 发出请求。
- $t \in T$：泄露技术（leak technique），即如何判别目标网站上的差异。

(s, d) 的定义比较抽象因为对于不同的浏览器，不同的 URL，不同的一对状态，或不同的 i 可以产生不同的差异。比如，若 s_0 是用户未登录的状态，s_1 是用户登录的状态，若使用缓存探测（章节 3.2），d_0 可以是用户发布的帖子未缓存，而 d_1 是用户发布的帖子以缓存，可以设 i 为时序攻击判别 d_0 以及 d_1。或者，若登录和未登录状态下的网页有不同的 `frame` 个数，可以用计帧法判别两个状态。

其次，虽然一个网页可以有多个状态，但可以简化到二元、是否问题。比如一个邮件可以有无限多种标题，但通过判别“第一个字母是否是‘a’”、“第一个字母是否是‘b’”、……、“第二个字母是否是‘a’”等等逐渐获得完整的标题。

3 漏洞类型

3.1 时序攻击

时序攻击通过测量两个事件之间的时间间隔而推断出信息。时序攻击的最重要的要素之一是时间间隔的测量[4]。Performance API 中的 `performance.now()` 返回网页加载到调用函数之间的时间间隔，可以达到微秒级别的精确度[5]。为了避免使用此函数的攻击，有些浏览器降低了`performance.now()` 的精确度[6-8]。与较老的 Date API，它们是两个显式时钟。此外，隐试的时钟包括[9]：

- CSS 动画
- setTimeout
- setImmediate
- postMessage
- Sub worker
- Broadcast Channel
- MessageChannel
- SharedArrayBuffer

一个使用时序攻击挖掘的信息包括用户的网速，通过测量一个较大文件的下载时间[10]。

在一些情况下，比如成功以及失败的请求所返回的响应的大小相似，时序攻击准确性会降低。膨胀技巧（inflation technique）可以增加两种响应时间的差异。一种办法是将服务器需要传输的数据增加，从而增加传输时间（见章节 3.3 中的案例）。另一种方式是增加服务器的计算工作量。此方法适用于让用户发出含有复杂的查询参数的请求[11]。膨胀以及统计技巧[12] 的结合可以将时序攻击成为一种精确的跨站点泄露漏洞。

3.2 缓存探测

当用于访问一个网页，由于用户再次访问同一个网页的概率较大，浏览器会将某些资源，比如图像、脚本、HTML 代码，缓存在用户的机器上。当用户再次访问用一个网页，浏览器不必从服务器再次下载，而可以更快速地从本地存储读取，从而加快了网页加载的速度。缓存探测漏洞基于检测某个资源是否被缓存，从而攻击者可以判断被攻击者是否曾经访问有个网页。

此类漏洞有多种实现方法。一种简单的方法是使用时序攻击技巧。这种攻击来自于缓存本质的用途，即若某个资源被缓存了，则它的访问时间较短，反之亦然。以下讨论其他实现方法。

3.2.1 错误事件[13]

1. 使被缓存的资源无效：
   - 使用 `cache:'reload'` 发出请求，在收到响应之前使用 `AbortController.abort()` 终止
   - 使用 `cache:'reload'` 以及 overlong referer header
   - 将请求失败的 Content-Type, Accept, Accept-Language 等等请求头，必须针对由一个网站
2. 发出请求，使得某一个资源被缓存
3. 再对同一个资源发出请求，但需要将服务器拒绝此请求（比如使用 overlong referer header）。若此资源在第二步被缓存了，则此请求会成功，否则抛出错误。

3.2.2 CORS 错误[14]

若响应包含 Access-Control-Allow-Orign (ACAO)，发出请求的来源以及被请求的资源一起被缓存在本地。若 `attacker.com` 访问此资源：

- 若此资源未被缓存，此资源以及 `Access-Control-Allow-Orign (ACAO): attacker.com` 将被缓存。
- 若此资源被缓存，由于 `attacker.com` 与以缓存的 `target.com` 不匹配，会产生 CORS 错误，从而可以判定此资源被缓存过。

此攻击容易避免：在资源上设置 `Access-Control-Allow-Origin: *`

3.2.3 防范措施[15]

- 通过设置 `Cache-Control: no-store` 禁用缓存，非常简单、高效地防止此类攻击，并被大多数浏览器支持，但对网页的加载速度有负面的影响。
- 在资源的 URL 中加随机记号，比如 `users/john.jpg` 变成 `users/john.jpg?cache_buster=<RANDOM_TOKEN>`。依然可以使用缓存机制，从而不会影响到加载速度，但是必须由每个网站的管理人员实现，而不是由浏览器实现，因此不是可以保护所有网站的防范措施。
- Fetch metadata: 可以让服务器判定请求来自于相同还是不同的来源。比如如果资源的 URL 为 `cdn.example.com/image.png` 并且设置了 `Vary: Sec-Fetch-Site`（SFS），则：

请求来源	SFS
example.com	same-site
cdn.example.com	same-origin
evil.com	cross-site

依然可以使用缓存机制但是弊端包括：

  - 并非所有浏览器支持 fetch metadata
  - 跨站点资源无法被保护
  - 资源若被第三方访问，也无法被保护

3.3 XS-Search

XS-Search 是攻击基于查询的搜索系统的一个重要技巧。此技巧通常需要做多个查询，蛮力获取关于查询对象的信息。以下由案例[16] 介绍 XS-Search 的运行方式。

Monorail 是 Chromium 以及其他相关项目使用的错误追踪系统。由于查询结果可以以 CSV 格式下载，并且没有针对 CSRF 的保护措施，所以攻击者可以跨源地下载（但不能读取）包含有关漏洞的信息的 CSV 文件。若攻击者将具有浏览未公开的漏洞的人员访问含有 `Restrict-View-SecurityTeam` 标签的 URL，则包含关于此类漏洞的信息将会被下载。

此外也存在膨胀 CSV 文件大小的方法。CSV 中的每一列可以在请求中定义，比如 `https://bugs.chromium.org/p/chromium/issues/csv?can=1&q=id:51337&colspec=ID+Summary+Summary+Summary` 会将一个编号列以及三个相同的“概要”列存放在 CSV 中，而每一行是符合搜索参数的漏洞。如果再插入更多概要列，因漏洞的概要在文件里多次重复，则含有漏洞和不含漏洞的 CSV 文件大小会有明显的差距。

判断文件的大小就需要使用与之前讨论到类似的漏洞。作者使用了 Cache API，因为只需要向服务器发出一个请求，并可以快速、重复地测量 CSV 文件存入缓存所需要的时间，从而去除了网络的抖动对时间测量带来的变动。虽然这个技巧无法测出文件的绝对大小，但与以知不含有漏洞的文件缓存所需要的时间做比较，就可以判定出任何文件是否含有漏洞。

最后需要选择合适的搜索参数。Monorail 不接受搜索单个字母，但可以搜索单词。作者也发现许多旧的漏洞报告中有存在问题的文件的路径以及行数。Chromium 的源代码库是公开的，因此完整的攻击过程如下：

1. 使用 `OR` 运算查询根目录下的一半的文件夹。
2. 若查询成功（即 CSV 文件大小较大），则对相同一半的文件夹重复第一步，直到寻找到了包含漏洞的文件夹。类似地，如果查询失败，则查询另一半文件夹。
3. 找到了包含漏洞的文件夹后，将这个文件夹设为根目录，重复第一步，直到找到含有漏洞的文件。

3.4 postMessage

`window.postMessage()` 实现了不同 `Window` 对象之间的跨站点通讯，比如一个网页和它创建的弹窗或它之中的 `iframe`[17]。`postMessage()` 所发出的 `MessageEvent` 这个事件可以泄漏信息，比如如果一个网站在用户名存在的情况下发出 `MessageEvent`，则可以判定一个用户的用户名。此外，`MessageEvent` 中也可以含有敏感信息。

防止攻击者滥用此漏洞目前是网页开发者的责任。开发者需要保证网页在不同的（登录、授权等等）状态下，`postMessage` 的行为是一致的。此外，应该合理设定 `postMessage` 的 `targetOrigin` 参数，使得只有目标站点可以读取 `MessageEvent` 中的信息。

3.5 Error Events

网页资源的加载可以成果或失败，并调用元素的一个函数比如 onload、onerror 等等[18]。用户未登录，服务器返回的数据无法被浏览器解析都可能 onerror 的触发。比如，如果用户的头像 target.com/profile_picture 只能被登录后的用户访问，则以下的代码可以泄漏用户的登录状态：

```
function is_logged_in(url) {
    let img = document.createElement('img');
    img.src = url;
    img.onload = () => console.log('Logged in');
    img.onerror = () => console.log('Not logged in');
}
is_logged_in('https://target.com/profile_picture');
```

不同的浏览器、HTML 标签、不同的头都可能影响到被触发的事件。与之前的一些攻击一样，为了防止此类攻击，网页开发者可以将不同状态的资源有一致的行为。此外，资源的 URL 后可以加随机记号，比如 target.com/profile_picture&token=<secret>。这样攻击者就无法容易地向资源发请求。

4 基于 Service Worker 的 Chromium 漏洞

以下的实验代码都存放在 https://github.com/georgetian3/xsleaks/tree/main/src/ 下。以下所关涉到的文件以及文件夹默认包含以上 URL 为前缀。库中大多数是模仿攻击的 HTML 以及 Javascript 代码，其中某些功能需要将代码布置到服务器上并使用 HTTPS 访问。以下测试都在 Chrome 106.0.5249.91 上运行的。

4.1 概念验证

Luan Herrera 在 2019 年提出了基于 service worker 以及 Performance API，影响到 Chromium 的攻击[19]。Service worker 可以作为网页、浏览器以及互联网之间的代理[20] 。它可以拦截一个网页发出的请求，并为此返回响应。Performance API 让开发者精确地测量网页在用户的设备上的性能[21]。其中的 `performance.getEntries()` 返回所有此网页及其资源的加载所产生的 `PerformanceEntry` 对象。

攻击过程如下：

1. 安装拦截 range header（字节范围头）为 `bytes=0-` 的请求的 Service Worker。
2. 使用 `audio` 或 `video` 元素向目标资源发出请求，其请求中的字节范围头为 `bytes=0-`。
3. Service Worker 拦截以上的请求，并返回任意内容，长度为 `n` 的响应。Chromium 会再发出类似的请求，类以区别是 `bytes=0-` 变成了 `bytes=n-`。分两种情况：
   (a) 目标资源的大小小于等于 `n`，则请求失败，服务器返回 `416` 状态码，此事件不产生 `PerformanceEntry`
   (b) 目标资源的大小大于 `n`，则请求成功，服务器返回 `206` 状态码，此事件产生 `PerformanceEntry`
4. 使用 `performance.getEntries().length` 可以得知当前的请求是否产生了 `PerformanceEntry`，从而可以判断目标资源的大小是否小于 `n`。

通过改变 `n`，比如用二分查找，可以较快地判定目标资源的大小。此攻击的源代码来源于 https://lbherrera.github.io/lab/chrome-8b024c22/sizeleak.html，并复制到了 Github 库的 /poc-original 中。

4.2 修复方法

由于概念验证中检测资源大小是基于`PerformanceEntry` 的个数从而判别成功与失败的请求，修复方法是将两种请求都产生 `PerformanceEntry`[22]：

> Currently we don't report performance entries with failing status codes. From the spec's perspective, reporting aborts is a MAY, but failing status code responses should not be considered aborts. [1] Chromium is the only engine which doesn't report those entries. This CL fixes that to report them similarly to successful status codes.

此修复的具体实现比较简单。源代码的版本库[23] 将

```
if (resource->GetResponse().IsHTTP() &&
    resource->GetResponse().HttpStatusCode() < 400)
```

改成

```
if (resource->GetResponse().IsHTTP())
```

即任何响应，无论状态码表示成功或失败，都会产生一个 PerformanceEntry。

通过运行 Herrera 的概念验证，可以确认此修复成功地防止此攻击。

4.3 PerformanceEntry 的创建条件

修复了以上的攻击之后，Chromium 的开发者再次更改了创建 PerformanceEntry 的条件，变成了：

```
if (resource->GetResponse().ShouldPopulateResourceTiming())
```

其中 ShouldPopulateResourceTiming 的定义为：

```
bool ResourceResponse::ShouldPopulateResourceTiming() const {
    return IsHTTP() || WebBundleURL().IsValid();
}
```

若希望通过 performance.getEntries() 的行为的差异实现一个漏洞，就需要分析如何 ShouldPopulateResourceTiming() 返回不同的布尔值。其中 WebBundelURL() 返回的是 KURL 类，而 IsValid() 返回的是 KURL 类的 is_valid_ 变量，而可以改变 is_valid_ 的成员函数为构造函数、拷贝构造函数、Init、以及 ReplaceComponents。IsHTTP() 返回 KURL 类的 protocol_is_in_http_family_ 变量，默认为假，而唯一将它设为真的地方在 InitProtocolMetadata 成员函数。由于代码比较繁琐，所以不再进一步分析这个方面。

4.4 其他标签的测试

Herrera 在它的概念验证中提到需要用 audio 或 video 标签实现攻击。此次测试目的是为了判定在修复 PerformanceEntry 的产生条件之前，其他哪些标签也可以发出请求。测试代码存放在 /tags 下。运行后并打开 Chrome DevTools。结果如下：

- audio、source、track、video（使用 src 属性）都有相同的行为，即：
  1. 请求头包含 Range: bytes=0- 的请求被 service worker 拦截，返回的状态码为 206
  2. 若响应头包含 Content-Range: bytes 0-9/100000，Chrome 再次发出请求，请求头包含 Range: bytes=10-，并且响应的状态码为 206
  3. 若响应头包含 Content-Range: bytes 0-999/100000，Chrome 再次发出请求，请求头包含 Range: bytes=1000-，并且响应的状态码为 416
- img、input、script 都被 service worker 拦截，响应的状态码为 206，但 Chromium 不会再次发出请求。

- `embed`、`frame`、`iframe`、`object` 不被 service worker 拦截，直接返回 `200` 状态码。

4.5 分析

以上可以被 service worker 拦截并且 Chromium 会再次发出响应的标签都适用于流数据，比如视频和声音。这是 Chromium 会再次发出请求的可能的原因。其他标签，虽然可能返回的是 `206` 状态码，但是 Chromium 不会再发出请求。

虽然此攻击可能泄漏的信息的导致此攻击的严重性较大，但是`performance.getEntries()` 的稳定性也较差。在测试概念验证的过程中，偶尔需要多次刷新后请求才会被 service worker 拦截。甚至在发出请求的时刻调整浏览器窗口的大小会改变此请求完成后产生的 `PerformanceEntry` 对象，以及将 `onsuspend` 与其他事件属性被调用。

此外，Herrera 在 2021 年提出了两个类似的攻击：

1. https://bugs.chromium.org/p/chromium/issues/detail?id=1260649
2. https://bugs.chromium.org/p/chromium/issues/detail?id=1270990

其中攻击过程与原始的过程的前两个步骤完全一样，不同之处在于如何判别响应的状态码。第一个攻击使用了 Cache API，而第二个响应也使用 `PerformanceEntry` 的创建，但只针对于 `<link preload >` 标签。这些攻击依然可以获取一个资源的大小，并且已经被修复。Herrera 发现的三个漏洞都使用 service worker 发出请求，但可以使用不同的方法探测响应的状态码，并且修复方式是防止攻击者检测到不同的状态码。这可能表明以上的修复没有修复这些漏洞的本质根源。Service worker 的权限可能过于广泛，而如何合理地限制它，避免产生更多漏洞，但依然让它有效地服务网页资源的加载是以后的研究方向。

参考文献

[1] SOUSA M, Terjanq, CLAPIS R, et al. XS-Leaks Wiki[EB/OL]. (2020-10-03) [2022-07-07]. https://xsleaks.dev/.

[2] MDN contributors. Origin[EB/OL]. (2022-07-03) [2022-07-07]. https://developer.mozilla.org/zh-CN/docs/Glossary/Origin.

[3] KNITTEL L, MAINKA C, NIEMIETZ M, et al. XSinator.Com: From a Formal Model to the Automatic Evaluation of Cross-Site Leaks in Web Browsers[C/OL]//CCS '21: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021: 1771-1788. https://doi.org/10.1145/3460120.3484739. DOI: 10.1145/3460120.3484739.

[4] SOUSA M, Terjanq, CLAPIS R, et al. Clocks[EB/OL]. (2020-12-23) [2022-09-14]. https://xsleaks.dev/docs/attacks/timing-attacks/clocks/.

[5] MDN contributors. performance.now()[EB/OL]. (2022-09-13) [2022-09-14]. https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.

[6] Pdr@chromium.org. Issue 506723: Reduce resolution of performance.now to prevent timing attacks[EB/OL]. (2015-07-03) [2022-09-14]. https://bugs.chromium.org/p/chromium/issues/detail?id=506723.

[7] CHRISTENSEN A. Bug 146531 - Reduce resolution of performance.now[EB/OL]. (2015-07-01) [2022-09-14]. https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.

[8] VEDITZ D. Reduce precision of performance.now() to 20us[EB/OL]. (2018-01-03) [2022-09-14]. https://bugzilla.mozilla.org/show_bug.cgi?id=1427870.

[9] SCHWARZ M, MAURICE C, GRUSS D, et al. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript[EB/OL]. 2020 [2022-09-14]. https://gruss.cc/files/fantastictimers.pdf.

[10] SOUSA M, Terjanq, CLAPIS R, et al. Connection speed[EB/OL]. (2021-02-11) [2022-09-07]. https://xsleaks.dev/docs/attacks/timing-attacks/performance-api/#connection-speed.

[11] SOUSA M, Terjanq, CLAPIS R, et al. Inflation Techniques[EB/OL]. (2020-10-03) [2022-09-27]. https://xsleaks.dev/docs/attacks/xs-search/#inflation-techniques.

[12] GELERNTER N, HERZBERG A. Cross-Site Search Attacks[EB/OL]. (2022-09-13) [2015]. https://web.archive.org/web/20200614162731/http://u.cs.biu.ac.il/~herzbea/security/15-01-XSSearch.pdf.

[13] SOUSA M, Terjanq, CLAPIS R, et al. Cache Probing with Error Events[EB/OL]. (2021-09-30) [2022-09-14]. https://xsleaks.dev/docs/attacks/cache-probing/#cache-probing-with-error-events.

[14] SOUSA M, Terjanq, CLAPIS R, et al. CORS error on Origin Reflection misconfiguration[EB/OL]. (2021-09-30) [2022-09-14]. https://xsleaks.dev/docs/attacks/cache-probing/#cors-error-on-origi

n-reflection-misconfiguration.

[15] SOUSA M, Terjanq, CLAPIS R, et al. Defense[EB/OL]. (2021-09-30) [2022-09-14]. https://xsleaks.dev/docs/attacks/cache-probing/#defense.

[16] HERRERA L. XS-Searching Google’ s bug tracker to find out vulnerable source code[EB/OL]. (2018-11-20) [2022-09-14]. https://medium.com/@luanherrera/xs-searching-googles-bug-tracker-to-find-out-vulnerable-source-code-50d8135b7549.

[17] MDN contributors. Window.postMessage()[EB/OL]. 2022 [2022-09-16]. https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage.

[18] W3schools. HTML Event Attributes[EB/OL]. (2022-09-27) [2022-09-16]. https://www.w3schools.com/tags/ref_eventattributes.asp.

[19] HERRERA L. Issue 990849: Leaking size of cross-origin resource by using Range Requests and Service Workers[EB/OL]. (2019-08-06) [2022-08-15]. https://bugs.chromium.org/p/chromium/issues/detail?id=990849.

[20] MDN contributors. Service Worker API[EB/OL]. (2022-09-27) [2022-09-16]. https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

[21] BUCKLER C. How to Evaluate Site Speed with the Performance API[EB/OL]. (2021-05-12) [2022-09-07]. https://blog.openreplay.com/how-to-evaluate-site-speed-with-the-performance-api.

[22] WEISS Y. [resource-timing] Report performance entries with failing status codes[EB/OL]. (2019-11-11) [2022-09-07]. https://chromium.googlesource.com/chromium/src/+/5e556dd80e03b7a217e10990d71be25d07e1ece7.

[23] WEISS Y. [resource-timing] Report performance entries with failing status codes[EB/OL]. (2019-11-11) [2022-09-07]. https://github.com/chromium/chromium/commit/5e556dd80e03b7a217e10990d71be25d07e1ece7.