跨站点泄露漏洞

George Tian（田正祺）

2022 年 9 月 17 日

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，最后聚焦一个曾经影响到 Chromium 浏览器的漏洞。

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。因此网络安全以及在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）[2] 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。服务器向浏览器发送的响应中的 CORS 头指示浏览器如何响应中的资源。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

2.0.1 基础模型[3]

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
  - $S = \{s_0, s_1\}$：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
- $i \in I$：包含技术，即如何从攻击网站向 sdr 发出请求
- $t \in T$：泄露技术，即如何观察目标网站上的差异

在以上的例子中：

- $url =$ `https://google.com/search?q=[query]`
- $S =$ { `query` 未被查询, `query` 已被查询 }
- $D =$ { 时间间隔较长, 时间间隔较短 }
- i 可以使用多种方式，比如将 url 嵌入到攻击网站中的 `iframe`
- t 为计时攻击

2.0.2 扩展模型[4]

Goethem et al. 用 Knittel et al. 的模型为基础，作出了扩展：

2.1 COSI

跨源状态推断（Cross-Origin State Inference）攻击[5]：

- 考虑两个网站
  - 攻击网站：用于发出跨站点请求，是攻击者（部分）控制的网站
  - 目标网站：用户在此网站上有不同的状态，不被攻击者控制的网站
- 攻击网站中含有依赖于状态的网址（state-dependent URL, SD-URL），比如一个只有用户登录后才能访问的网页
- 被包含的 SD-URL 使用户的浏览器发出跨源请求，但同源策略防止攻击网站直接阅读响应
- 攻击者可以通过跨站点泄露漏洞间接地读取响应

State scripts: In this work, we capture states at a target site using state scripts that can be executed to automatically log into the target site using a configurable browser and the credentials of an account with a specific configuration. For example, we may create multiple user accounts with different configurations, e.g., premium and free accounts, two users that own different blogs, or authors that have submitted different papers to a conference management system. We also create a state script for the logged out state.

两个阶段：

1. 准备
   - The goal of the preparation phase is to create an attack page that when visited by a victim will leak the victim’s state at the target web site
   - An attack page implements at least one, possibly more, attack vectors
   - Attack vector = (SD-URL, inclusion method (embedding SD-URL into attack page), attack class (defines, among others, a leak method (or XS-Leak) that interacts with the victim’s browser to disclose a victim’s state at the target site))
   - URL is state-dependent if, when requested through HTTP(S), it returns different responses depending on the state it is visited from
   - Note that it is not needed that each state returns a different response. For example, if there are 6 states and two different responses, each for three states, the URL is still state-dependent
   - When the attack page is visited by the victim, the inclusion method forces the victim’s browser to automatically request the SD-URL from the target site.
   - However, there exist XS-Leaks that allow bypassing a browser’s SOP to disclose information about cross-origin responses
   - In this work, we introduce the concept of a COSI attack class, which defines the two different responses to a SD-URL that can be distinguished using a XS-Leak, the possible inclusion methods that can be used in conjunction with the XS-Leak, and the browsers affected

- Attacks classes are independent of the target site states and thus can be used to mount attacks against different targets

2. 攻击
   - the attacker convinces the victim into visiting the attack page
   - **more than 2 states possible**
3. threat model
   - attacker: can trick victims into loading the attack page on their web browsers, during prep can create/manage diff acc on target site, controls attack site, can identify victim's browser version
   - victim: uses updated browser, can be lured to attack site, uses target website using same browser as attack website
   - target site: contains at least one SD-URL for which the attacker knows an attack class, does not suffer from any known vulns

2.1.1 COSI attack classes

6-tuple

- class name
- 2 signatures for 2 groups of responses that can be distinguished using the attack class
- xsleak
- list of inclusion methods
- list of affected browsers

Identifying attack classes

- Our process to discover COSI attack classes comprises of three main steps: (1) identify and validate previously proposed COSI attack instances; (2) generalize known COSI attack instances into COSI attack classes; and (3) discover previously unknown attack classes.
- Generalizing a COSI attack instance into a COSI attack class comprises of two steps. First, identifying the set of responses to the inclusion method used in the attack instance, that still trigger the same observable difference in the browser (e.g., onload/onerror or different object property values). Then, checking if the observable difference still manifests with other inclusion methods and browsers

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[8]。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[9]。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[10-12]。与较老的 Date API，它们是两个显式时钟。此外，隐试的时钟包括[13]：

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

比如，可以可以
https://xsleaks.dev/docs/attacks/timing-attacks/performance-api/#connection-speed
inflation
statistical analysis

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。

Related:

- https://www.cs.jhu.edu/~fabian/courses/CS600.424/course_papers/webtiming.pdf - OLD, from 2000, gives experimental results, also discusses DNS and cookie cache
- https://terjanq.github.io/Bug-Bounty/Google/cache-attack-06jd2d2mz2r0/index.html - error based cache attack on Google products

- http://sirdarckcat.blogspot.com/2019/03/http-cache-cross-site-leaks.html - addresses some basic defense strategies
- https://web.archive.org/web/20200614162731/http://u.cs.biu.ac.il/~herzbea/security/15-01-XSSearch.pdf - improvements on timing attacks, uses statistics, amplification, and DaC algs, not specific to cache probing
- https://link.springer.com/content/pdf/10.1007/978-3-319-18467-8.pdf pdf page 110, parallized cache probing

3.3 错误事件

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
   - A POST request with a fetch no-cors
   - 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

3.4 CORS error on Origin Reflection misconfiguration

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

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

3.5 Fetch with AbortController

见错误事件

3.6 防范措施

- 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，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：依然可以使用缓存机制

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

  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

3.7 XS-Search

XS-Search 是攻击基于查询的搜索系统的一个重要技巧。与之前的技巧而相比，此技巧更加。。。

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。

比如，如果

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，因为只需要想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。作者也提到可以使用 ————————————-

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。类似的，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。

3.8 Window Reference

若

3.9 postMessage

https://developer.mozilla.org/en-US/docs/Web/API/Window/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。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）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。

关键词：浏览器安全、跨站点泄露漏洞、Chromium。

3.10 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 https://www.w3schools.com/tags/ref_eventattributes.asp。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 与 Performance API 的 Chromium 漏洞

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[6]。Service worker https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API

Performance API（性能接口）让开发者精确地测量网页在用户的设备上的性能[7]。此攻击

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 `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。比如，Herrera 对 Google 的错误追踪系统的 XS-Search 攻击就可以使用此漏洞。

（测试包含 service worker 的网站后，为了注销 service worker, 可能需要重启浏览器程序）。

4.2 修复方法

由于概念验证中检测资源大小基于成功与失败创建 `PerformanceEntry` 的行为的差异 https://chromium.googlesource.com/chromium/src/+/5e556dd80e03b7a217e10990d71be25d07e1ece7

> 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。源代码的版本库将 https://chromium-review.googlesource.com/c/chromium/src/+/1796544

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

改成

```
1 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 Luan 的概念验证，可以确认此修复成功地防止此攻击。

4.3 扩展

不稳定

参考文献

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

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

[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]// Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2021. https://doi.org/10.1145%2F3460120.3484739. DOI: 10.1145/3460120.3484739.

[4] GOETHEM T V, FRANKEN G, SANCHEZ-ROLA I, et al. SoK[C/OL]// Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. ACM, 2022. https://doi.org/10.1145%2F3488932.3517416. DOI: 10.1145/3488932.3517416.

[5] SUDHODANAN A, KHODAYARI S, CABALLERO J. Cross-Origin State Inference (COSI) Attacks: Leaking Web Site States through XS-Leaks[J/OL]., 2019. https://arxiv.org/abs/1908.02204. DOI: 10.48550/ARXIV.1908.02204.

[6] 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.

[7] 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.

[8] SOUSA M, Terjanq, CLAPIS R, et al. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript[EB/OL]. (2020-12-23) [2022-09-14]. Clocks.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.