跨站点泄露漏洞

George Tian（田正祺）

2022 年 8 月 14 日

摘要

摘要

1 引言

1.1 基础概念

来源（Origin）[1]: Web 内容的 URL 的方案 (协议)，主机 (域名) 和端口。两个对象是同源的当且仅当以上三个元素都相等。

旁道攻击（Side-channel attack）：通过观察系统运行过程中的物理属性从而获取信息，而非暴力破解或运用算法的理论弱点。

跨站点泄露漏洞（Cross-site leaks, XSLeaks）[2]：一类存在与浏览器中的旁道攻击，让一个站点获取另一个站点的某些信息。

例子：`evil.com` 想得知用户在 `google.com` 上的搜索的关键字。`evil.com` 可以发出请求 `https://google.com/search?q=a`, `https://google.com/search?q=b` ……并测量从发出请求和接受到响应之间的时间间隔。若请求中发出的关键字被用户被曾经访问过，响应时间与未访问过的关键字响应相比会较短，因为本地会缓存响应中一部分的数据。

1.2 形式化建模

1.2.1 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

1.2.2 扩展模型[4]

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

1.3 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

1.3.1 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

2 缓存探测漏洞

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——时序攻击——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

2.1 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

2.2 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: *`

2.3 Fetch with AbortController

见错误事件

2.4 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

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.