1 漏洞类型

1.1 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[1]。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[2]。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[3-5]。与较老的 Date API，它们是两个显式时钟。此外，隐试的时钟包括[6]：

- 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

1.2 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

1.3 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

1.4 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: *`

1.5 Fetch with AbortController

见错误事件

1.6 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

参考文献

[1] 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.

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

[3] Pdr@chromium.prg. 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.

[4] 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.

[5] 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.

[6] 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.