

34 | CSRF攻击：陌生链接不要随便点

2019-10-22 李兵

浏览器工作原理与实践

[进入课程 >](#)



讲述：李兵

时长 13:35 大小 10.90M



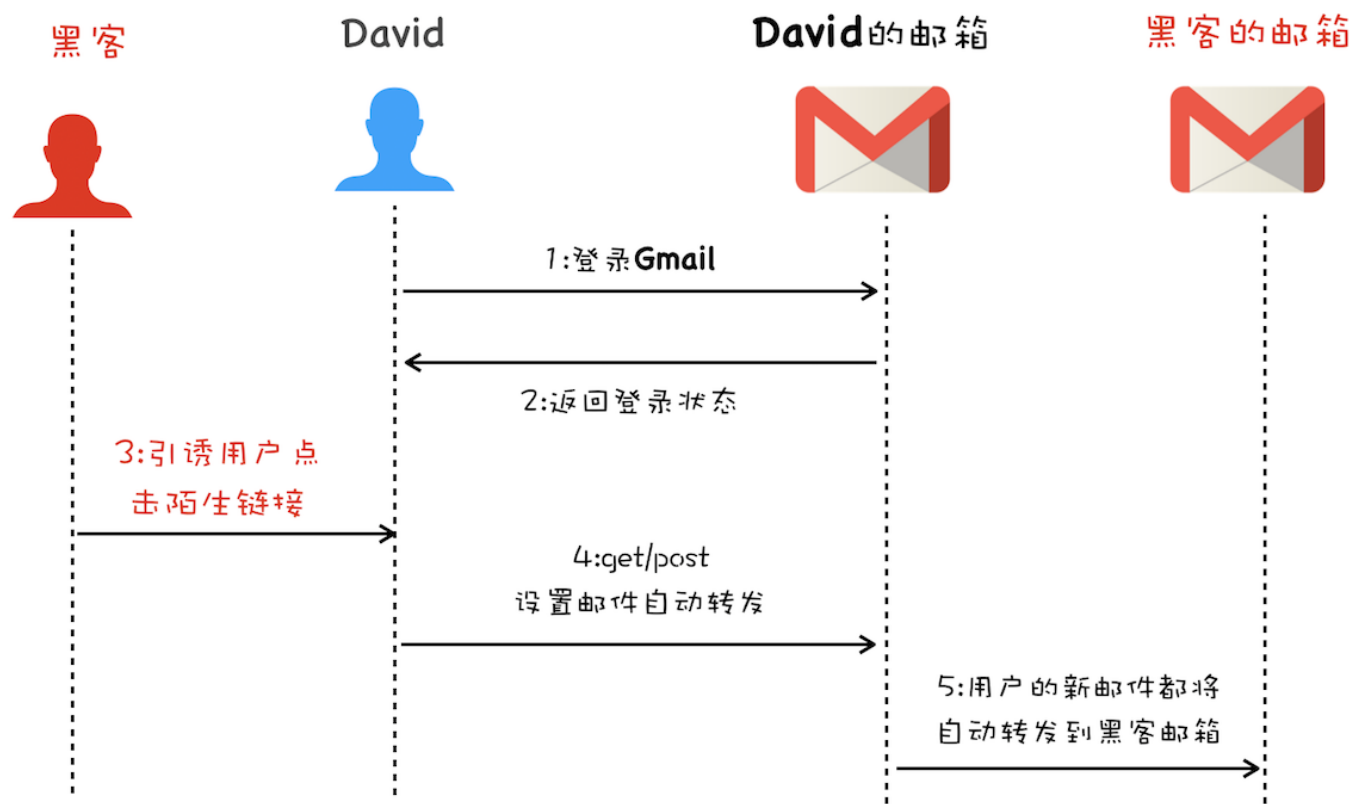
在[上一篇](#)文章中我们讲到了 XSS 攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS 攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的 CSRF 攻击。

相信你经常能听到的一句话：“别点那个链接，小心有病毒！”点击一个链接怎么就能染上病毒了呢？

我们结合一个真实的关于 CSRF 攻击的典型案例分析下，在 2007 年的某一天，David 无意间打开了 Gmail 邮箱中的一份邮件，并点击了该邮件中的一个链接。过了几天，David 就发现他的域名被盗了。不过几经周折，David 还是要回了他的域名，也弄清楚了他的域名之所以被盗，就是因为无意间点击的那个链接。

那 David 的域名是怎么被盗的呢？

我们结合下图来分析下 David 域名的被盗流程：



David 域名被盗流程

首先 David 发起登录 Gmail 邮箱请求，然后 Gmail 服务器返回一些登录状态给 David 的浏览器，这些信息包括了 Cookie、Session 等，这样在 David 的浏览器中，Gmail 邮箱就处于登录状态了。

接着黑客通过各种手段引诱 David 去打开他的链接，比如 hacker.com，然后在 hacker.com 页面中，黑客编写好了一个邮件过滤器，并通过 Gmail 提供的 HTTP 设置接口设置好了新的邮件过滤功能，该过滤器会将 David 所有的邮件都转发到黑客的邮箱中。

最后的事情就很简单了，因为有了 David 的邮件内容，所以黑客就可以去域名服务商那边重置 David 域名账户的密码，重置好密码之后，就可以将其转出到黑客的账户了。

以上就是 David 的域名被盗的完整过程，其中前两步就是我们今天要聊的 CSRF 攻击。David 在要回了他的域名之后，也将整个攻击过程分享到他的站点上了，如果你感兴趣的话，可以参考 [该链接](#)（放心这个链接是安全的）。

什么是 CSRF 攻击

CSRF 英文全称是 Cross-site request forgery，所以又称为“跨站请求伪造”，是指黑客引诱用户打开黑客的网站，在黑客的网站中，利用用户的登录状态发起的跨站请求。简单来讲，**CSRF 攻击就是黑客利用了用户的登录状态，并通过第三方的站点来做一些坏事。**

通常当用户打开了黑客的页面后，黑客有三种方式去实施 CSRF 攻击。

下面我们以极客时间官网为例子，来分析这三种攻击方式都是怎么实施的。这里假设极客时间具有转账功能，可以通过 POST 或 Get 来实现转账，转账接口如下所示：

 复制代码

```
1 # 同时支持 POST 和 Get
2 # 接口
3 https://time.geekbang.org/sendcoin
4 # 参数
5 ## 目标用户
6 user
7 ## 目标金额
8 number
```

有了上面的转账接口，我们就可以来模拟 CSRF 攻击了。

1. 自动发起 Get 请求

黑客最容易实施的攻击方式是自动发起 Get 请求，具体攻击方式你可以参考下面这段代码：

 复制代码


```
1 <!DOCTYPE html>
2 <html>
3   <body>
4     <h1> 黑客的站点: CSRF 攻击演示 </h1>
5     
6   </body>
7 </html>
```

这是黑客页面的 HTML 代码，在这段代码中，黑客将转账的请求接口隐藏在 img 标签内，欺骗浏览器这是一张图片资源。当该页面被加载时，浏览器会自动发起 img 的资源请求，

如果服务器没有对该请求做判断的话，那么服务器就会认为该请求是一个转账请求，于是用户账户上的 100 极客币就被转移到黑客的账户上去了。

2. 自动发起 POST 请求

除了自动发送 Get 请求之外，有些服务器的接口是使用 POST 方法的，所以黑客还需要在他的站点上伪造 POST 请求，当用户打开黑客的站点时，是自动提交 POST 请求，具体的方式你可以参考下面示例代码：

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4   <h1> 黑客的站点: CSRF 攻击演示 </h1>
5   <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=PO:
6     <input type="hidden" name="user" value="hacker" />
7     <input type="hidden" name="number" value="100" />
8   </form>
9   <script> document.getElementById('hacker-form').submit(); </script>
10 </body>
11 </html>
```

在这段代码中，我们可以看到黑客在他的页面中构建了一个隐藏的表单，该表单的内容就是极客时间的转账接口。当用户打开该站点之后，这个表单会被自动执行提交；当表单被提交之后，服务器就会执行转账操作。因此使用构建自动提交表单这种方式，就可以自动实现跨站点 POST 数据提交。

3. 引诱用户点击链接

除了自动发起 Get 和 Post 请求之外，还有一种方式是诱惑用户点击黑客站点上的链接，这种方式通常出现在论坛或者恶意邮件上。黑客会采用很多方式去诱惑用户点击链接，示例代码如下所示：

 复制代码

```
1 <div>
2   <img width=150 src=http://images.xuejuzi.cn/1612/1_161230185104_1.jpg> </img>
3   <a href="https://time.geekbang.org/sendcoin?user=hacker&number=100" taget="_l
4     点击下载美女照片
5   </a>
6 </div>
```

这段黑客站点代码，页面上放了一张美女图片，下面放了图片下载地址，而这个下载地址实际上是黑客用来转账的接口，一旦用户点击了这个链接，那么他的极客币就被转到黑客账户上了。

以上三种就是黑客经常采用的攻击方式。如果当用户登录了极客时间，以上三种 CSRF 攻击方式中的任何一种发生时，那么服务器都会将一定金额的极客币发送到黑客账户。

到这里，相信你已经知道什么是 CSRF 攻击了。**和 XSS 不同的是，CSRF 攻击不需要将恶意代码注入用户的页面，仅仅是利用服务器的漏洞和用户的登录状态来实施攻击。**

如何防止 CSRF 攻击

了解了 CSRF 攻击的一些手段之后，我们再来看看 CSRF 攻击的一些“特征”，然后根据这些“特征”分析下如何防止 CSRF 攻击。下面是我总结的发起 CSRF 攻击的三个必要条件：

第一个，目标站点一定要有 CSRF 漏洞；

第二个，用户要登录过目标站点，并且在浏览器上保持有该站点的登录状态；

第三个，需要用户打开一个第三方站点，可以是黑客的站点，也可以是一些论坛。

满足以上三个条件之后，黑客就可以对用户进行 CSRF 攻击了。这里还需要额外注意一点，与 XSS 攻击不同，CSRF 攻击不会往页面注入恶意脚本，因此黑客是无法通过 CSRF 攻击来获取用户页面数据的；其最关键的一点是要能找到服务器的漏洞，所以说对于 CSRF 攻击我们主要的防护手段是提升服务器的安全性。

要让服务器避免遭受到 CSRF 攻击，通常有以下几种途径。

1. 充分利用好 Cookie 的 SameSite 属性

通过上面的介绍，相信你已经知道了黑客会利用用户的登录状态来发起 CSRF 攻击，而 **Cookie 正是浏览器和服务器之间维护登录状态的一个关键数据**，因此要阻止 CSRF 攻击，我们首先就要考虑在 Cookie 上来做文章。

通常 CSRF 攻击都是从第三方站点发起的，要防止 CSRF 攻击，我们最好能实现从第三方站点发送请求时禁止 Cookie 的发送，因此在浏览器通过不同来源发送 HTTP 请求时，有如下区别：

如果是从第三方站点发起的请求，那么需要浏览器禁止发送某些关键 Cookie 数据到服务器；

如果是同一个站点发起的请求，那么就需要保证 Cookie 数据正常发送。

而我们要聊的 Cookie 中的 SameSite 属性正是为了解决这个问题的，通过使用 SameSite 可以有效地降低 CSRF 攻击的风险。

那 SameSite 是怎么防止 CSRF 攻击的呢？

在 HTTP 响应头中，通过 set-cookie 字段设置 Cookie 时，可以带上 SameSite 选项，如下：

 复制代码

```
1 set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=,
```

SameSite 选项通常有 Strict、Lax 和 None 三个值。

Strict 最为严格。如果 SameSite 的值是 Strict，那么浏览器会完全禁止第三方 Cookie。简言之，如果你从极客时间的页面中访问 InfoQ 的资源，而 InfoQ 的某些 Cookie 设置了 SameSite = Strict 的话，那么这些 Cookie 是不会被发送到 InfoQ 的服务器上的。只有你从 InfoQ 的站点去请求 InfoQ 的资源时，才会带上这些 Cookie。

Lax 相对宽松一点。在跨站点的情况下，从第三方站点的链接打开和从第三方站点提交 Get 方式的表单这两种方式都会携带 Cookie。但如果在第三方站点中使用 Post 方法，或者通过 img、iframe 等标签加载的 URL，这些场景都不会携带 Cookie。

而如果使用 None 的话，在任何情况下都会发送 Cookie 数据。

关于 SameSite 的具体使用方式，你可以参考这个链接：[🔗 https://web.dev/samesite-cookies-explained](https://web.dev/samesite-cookies-explained)。

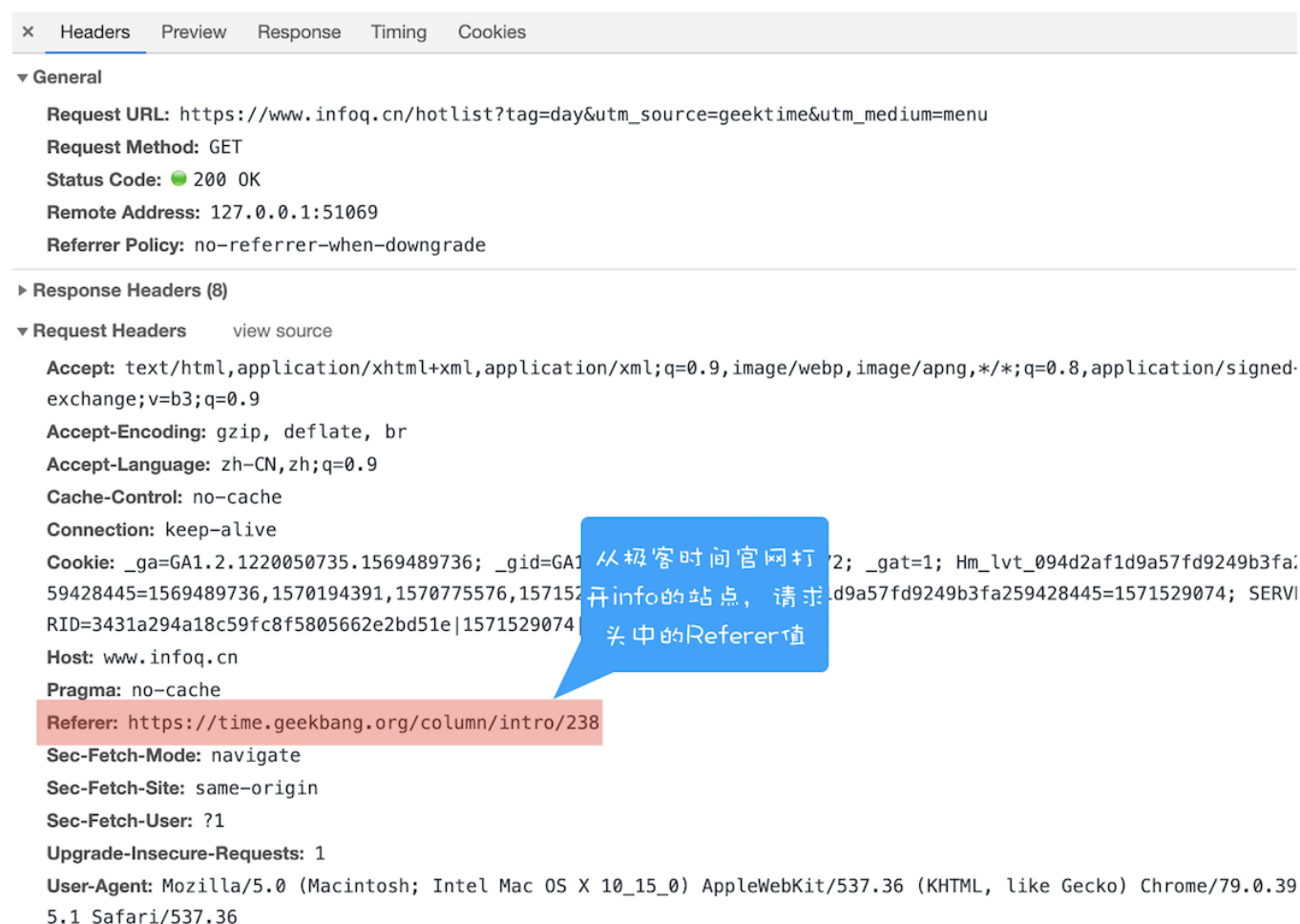
对于防范 CSRF 攻击，我们可以针对实际情况将一些关键的 Cookie 设置为 Strict 或者 Lax 模式，这样在跨站点请求时，这些关键的 Cookie 就不会被发送到服务器，从而使得黑客的 CSRF 攻击失效。

2. 验证请求的来源站点

接着我们再来了解另外一种防止 CSRF 攻击的策略，那就是**在服务器端验证请求来源的站点**。由于 CSRF 攻击大多来自于第三方站点，因此服务器可以禁止来自第三方站点的请求。那么该怎么判断请求是否来自第三方站点呢？

这就需要介绍 HTTP 请求头中的 Referer 和 Origin 属性了。

Referer 是 HTTP 请求头中的一个字段，记录了该 HTTP 请求的来源地址。比如我从极客时间的官网打开了 InfoQ 的站点，那么请求头中的 Referer 值是极客时间的 URL，如下图：



HTTP 请求头中的 Referer 引用

虽然可以通过 Referer 告诉服务器 HTTP 请求的来源，但是有一些场景是不适合将来源 URL 暴露给服务器的，因此浏览器提供给开发者一个选项，可以不用上传 Referer 值，具体可参考**Referrer Policy**。

但在服务器端验证请求头中的 Referer 并不是太可靠，因此标准委员会又制定了**Origin 属性**，在一些重要的场合，比如通过 XMLHttpRequest、Fetch 发起跨站请求或者通过 Post 方法发送请求时，都会带上 Origin 属性，如下图：

Name	Value
logo_pc@2x.90583da.png	
32dd4528daa249388b94cd6...	
hot_words	
chapters	
5930592a22614882646c595f...	
articles	
390cdf2bdcaedfe99e03dfab1...	
f0e68dbd84eeddaadd6f53b...	
32dd4528daa249388b94cd6...	
blob:https://time.geekbang.or...	
32dd4528daa249388b94cd6...	
blob:https://time.geekbang.or...	
sync-cookie.html?v=1	
info	
vendor-v2019.10.17.01.js	
app-v2019.10.17.01.js	
get_base_config?ent_id=161...	
new-chat.ogg	
new-message.ogg	
sent-message.ogg	
40z3oz40z4lz17z4bz3mz48z4...	
32dd4528daa249388b94cd6...	
32dd4528daa249388b94cd6...	
init?ent_id=161770&track_id=...	
info?browser_id=ca1477718f...	
32dd4528daa249388b94cd6...	
32dd4528daa249388b94cd6...	
32dd4528daa249388b94cd6...	
32dd4528daa249388b94cd6...	
32dd4528daa249388b94cd6...	

× Headers Preview Response Timing Cookies Initiator

▼ General

Request URL: https://time.geekbang.org/serv/v1/chapters

Request Method: POST

Status Code: 200 OK

Remote Address: 127.0.0.1:51069

Referrer Policy: no-referrer-when-downgrade

► Response Headers (12)

▼ Request Headers view source

Accept: application/json, text/plain, */*

Accept-Encoding: gzip, deflate, br

Accept-Language: zh-CN,zh;q=0.9

Cache-Control: no-cache

Connection: keep-alive

Content-Length: 13

Content-Type: application/json

Cookie: _ga=GA1.2.877361196.1567727491; MEIQIA_TRACK_ID=1Pa17nY4RoRNz5RXDHS3AsA1Rzb; en; GCID=0812b5f-2b8abc9-7fc872d-6d1a641; GRID=0812b5f-2b8abc9-7fc872d-6d1a641; _gi MEIQIA_VISIT_ID=1SRarQdoAgb027znTREXc1QeEN1; Hm_lvt_022f847c4e3acd44d4a2481d9187f1e 29580; _gat=1; Hm_lpvt_022f847c4e3acd44d4a2481d9187f1e6=1571533140; SERVERID=1fa1f3 71533142|1571529047

Host: time.geekbang.org

Origin: https://time.geekbang.org

Pragma: no-cache

Referer: https://time.geekbang.org/column/intro/216

Sec-Fetch-Mode: cors

Sec-Fetch-Site: same-origin

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_0) AppleWebKit/537.36 (KHTML 5.1 Safari/537.36

Post 请求时的 Origin 信息

从上图可以看出，Origin 属性只包含了域名信息，并没有包含具体的 URL 路径，这是 Origin 和 Referer 的一个主要区别。在这里需要补充一点，Origin 的值之所以不包含详细路径信息，是有些站点因为安全考虑，不想把源站点的详细路径暴露给服务器。

因此，服务器的策略是优先判断 Origin，如果请求头中没有包含 Origin 属性，再根据实际情况判断是否使用 Referer 值。

3. CSRF Token

除了使用以上两种方式来防止 CSRF 攻击之外，还可以采用 CSRF Token 来验证，这个流程比较好理解，大致分为两步。

第一步，在浏览器向服务器发起请求时，服务器生成一个 CSRF Token。CSRF Token 其实就是服务器生成的字符串，然后将该字符串植入到返回的页面中。你可以参考下面示例代码：

 复制代码

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4     <form action="https://time.geekbang.org/sendcoin" method="POST">
5         <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9a;
6         <input type="text" name="user">
7         <input type="text" name="number">
8         <input type="submit">
9     </form>
10 </body>
11 </html>
```

第二步，在浏览器端如果要发起转账的请求，那么需要带上页面中的 CSRF Token，然后服务器会验证该 Token 是否合法。如果是从第三方站点发出的请求，那么将无法获取到 CSRF Token 的值，所以即使发出了请求，服务器也会因为 CSRF Token 不正确而拒绝请求。

总结

好了，今天我们就介绍到这里，下面我来总结下本文的主要内容。

我们结合一个实际案例介绍了 CSRF 攻击，要发起 CSRF 攻击需要具备三个条件：目标站点存在漏洞、用户要登录过目标站点和黑客需要通过第三方站点发起攻击。

根据这三个必要条件，我们又介绍了该如何防止 CSRF 攻击，具体来讲主要有三种方式：充分利用好 Cookie 的 SameSite 属性、验证请求的来源站点和使用 CSRF Token。这三种方式需要合理搭配使用，这样才可以有效地防止 CSRF 攻击。

再结合前面两篇文章，我们可以得出页面安全问题的主要原因就是浏览器为同源策略开的两个“后门”：一个是在页面中可以任意引用第三方资源，另外一个是通过 CORS 策略让 XMLHttpRequest 和 Fetch 去跨域请求资源。

为了解决这些问题，我们引入了 CSP 来限制页面任意引入外部资源，引入了 HttpOnly 机制来禁止 XMLHttpRequest 或者 Fetch 发送一些关键 Cookie，引入了 SameSite 和 Origin 来防止 CSRF 攻击。

通过这三篇文章的分析，相信你应该已经能搭建**Web 页面安全**的知识体系网络了。有了这张网络，你就可以将 HTTP 请求头和响应头中各种安全相关的字段关联起来，比如 Cookie 中的一些字段，还有 X-Frame-Options、X-Content-Type-Options、X-XSS-Protection 等字段，也可以将 CSP、CORS 这些知识点关联起来。当然这些并不是浏览器安全的全部，后面两篇文章我们还会介绍**浏览器系统安全**和**浏览器网络安全**两大块的内容，这对于你学习浏览器安全来说也是至关重要的。

思考题

今天留给你的思考题：什么是 CSRF 攻击？在开发项目过程中应该如何防御 CSRF 攻击？

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

浏览器工作原理与实践

>>> 透过浏览器看懂前端本质

李兵

前盛大创新院高级研究员



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 33 | 跨站脚本攻击 (XSS)：为什么Cookie中有HttpOnly属性？

下一篇 35 | 安全沙箱：页面和系统之间的隔离墙

精选留言 (6)

写留言



李小白

2019-10-22

老师，我想请问一下，在浏览器打开第三方站点是如何拿到极客时间站点cookie的？第三方站点和极客时间的站点不同，存在同源策略，所以转账请求验证cookie也是不通过的，那么CSRF是如何攻击的呢？

3

3



淡

2019-10-22

“简言之，如果你从极客时间的页面中访问 InfoQ 的资源，而 InfoQ 的某些 Cookie 设置了 SameSite = Strict 的话，那么这些 Cookie 是不会被发送到 InfoQ 的服务器上的”，这里是不是我理解错了还是写错了。应该是不会发送到极客时间的服务器上，或者说极客时间的某些Cookie设置了SameSite = Strict吧。

展开 ▾

作者回复: 我把整个流程写一遍:

首先假设你发出登录InfoQ的站点请求, 然后在InfoQ返回HTTP响应头给浏览器, InfoQ响应头中的某些set-cookie字段如下所示:

```
set-cookie: a_value=avalue_xxx; expires=Thu, 21-Nov-2019 03:53:16 GMT; path=/; domain=.infoq.com; SameSite=strict
```

```
set-cookie: b_value=bvalue_xxx; expires=Thu, 21-Nov-2019 03:53:16 GMT; path=/; domain=.infoq.com; SameSite=lax
```

```
set-cookie: c_value=cvaule_xxx; expires=Thu, 21-Nov-2019 03:53:16 GMT; path=/; domain=.infoq.com; SameSite=none
```

```
set-cookie: d_value=dvaule_xxxx; expires=Thu, 21-Nov-2019 03:53:16 GMT; path=/; domain=.infoq.com;
```

我们可以看出,

a_value的SameSite属性设置成了strict,

b_value的SameSite属性设置成了lax

c_value的SameSite属性值设置成了none

d_value没有设置SameSite属性值

好, 这些Cookie设置好之后, 当你再次在InfoQ的页面内部请求InfoQ的资源时, 这些Cookie信息都会被附加到HTTP的请求头中, 如下所示:

```
cookie: a_value=avalue_xxx;b_value=bvalue_xxx;c_value=cvaule_xxx;d_value=dvaule_xxxx;
```

但是, 假如你从time.geekbang.org的页面中, 通过a标签打开页面, 如下所示:

```
<a href="https://www.infoq.cn/sendcoin?user=hacker&number=100">点我下载</a>
```

当用户点击整个链接的时候, 因为InfoQ中a_vaule的SameSite的值设置成了strict, 那么a_vaule的值将不会被携带到这个请求的HTTP头中。

如果time.geekbang.org的页面中, 有通过img来加载的infoq的资源代码, 如下所示:

```

```

那么在加载infoQ资源的时候, 只会携带c_value和d_value的值。

这样写不知道你明白没有, 如果还有疑惑欢迎继续提问。

2

1



許敲敲



2019-10-22

老师，这方面有比较好的资料或是书嘛？想多了解一下

展开 ∨

作者回复: 主要是根据这几年工作经验和浏览器文档、源码总结出来的，因为维护一个日活跃几千万的浏览器，能带来大量的流量，而流量就是金钱，因此我们的浏览器会遭受到各种类型的攻击，所以我们有很大一部分时间都是在处理攻击！

当然为了写这个专栏，网上web安全相关的书籍我也买了好多，基本都看了一圈，但是有一些问题：

第一是这些书籍都有一些年限了，里面的知识比较滞后！

第二，大多数和前端关系不是太大

第三，大多数写的比较零碎

你们如果有好的web安全相关的书籍，也可以推荐下！



💬 1

👍 1



可笑的霸王

2019-10-23

老师，关于Referrer服务器验证不太稳定可以详细解释下吗，因为我看到Referrer-Policy也可以设置为origin，达到Origin类似的效果

展开 ∨

💬

👍



Chao

2019-10-22

chrome 默认启用 SameSite 了

展开 ∨

💬

👍



蓝配鸡

2019-10-22

有个疑问：

same origin policy不是确保了不同域名之间不可以访问数据的吗？那第三方站点如何拿到cookie和session？

谢谢老师🙏

展开 ∨

作者回复: 如果是CSRF攻击, 那么黑客是拿不到受害者站点数据的。

但是黑客会在他的A站点中调用受害者B站点的http接口, 这些接口可以是转账, 删帖或者设置等。

这个过程中你需要注意一点, 在黑客A站点中调用受害者B站点的http接口时, 默认情况下, 浏览器依然会把受害者的Cookie等信息数据发送到受害者的B站点, 【注意这里并不是黑客的A站点】。

如果B站点存在漏洞的话, 那么黑客就会攻击成功, 比如将受害者的金币转出去了!

这样解释不知道问你清楚了没有?

