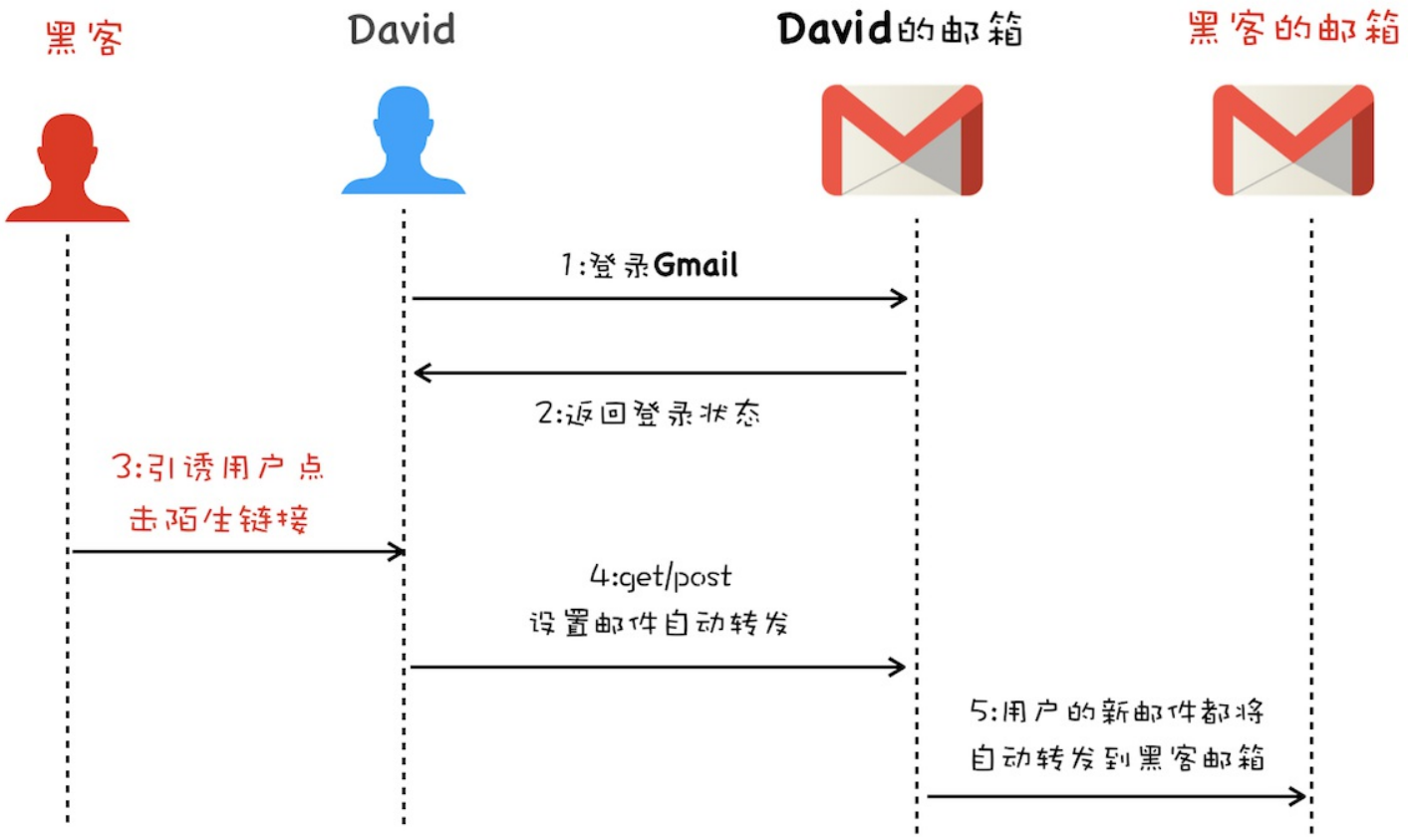
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
```

```
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

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

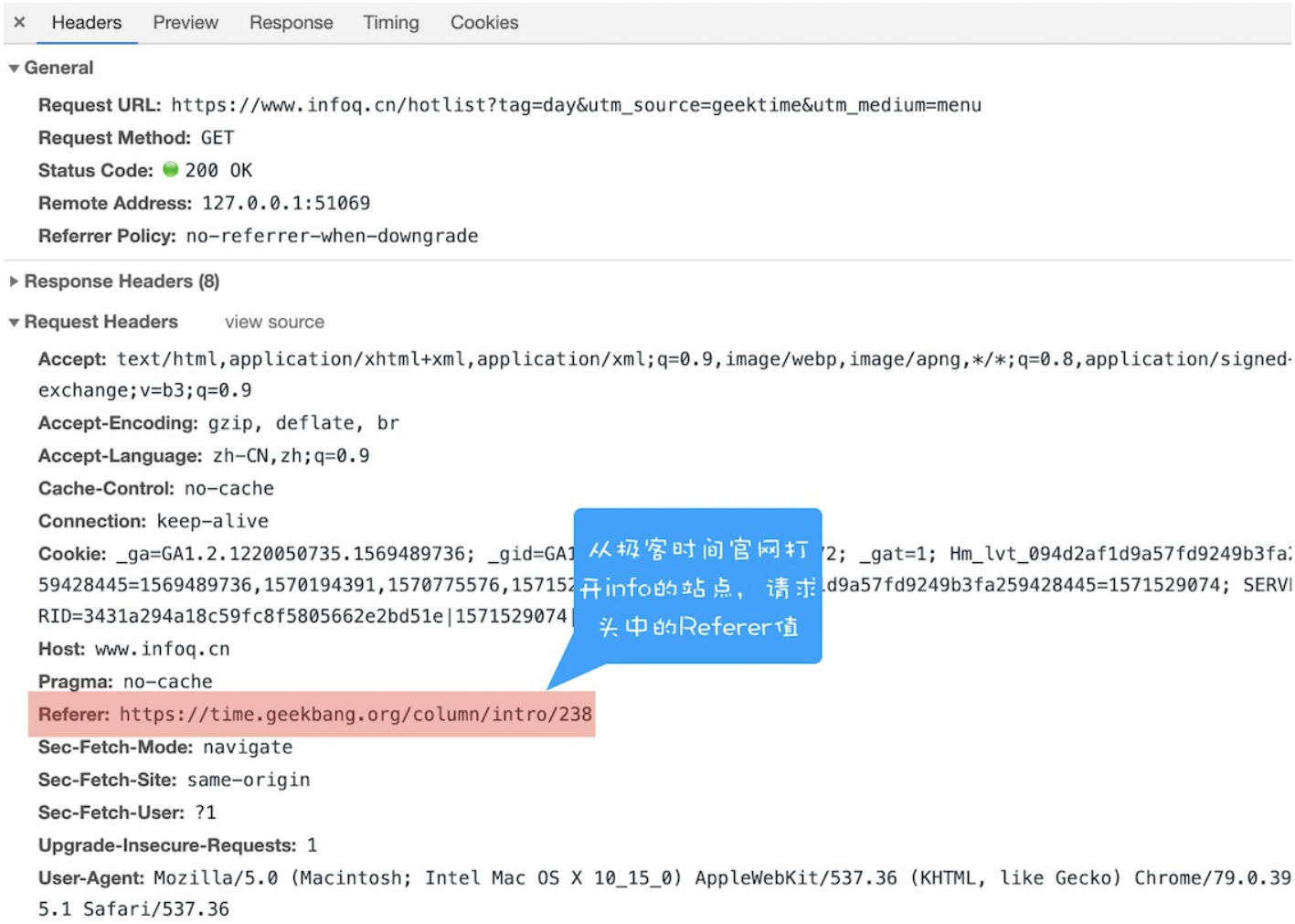
使得黑客的CSRF攻击失效。

### 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。

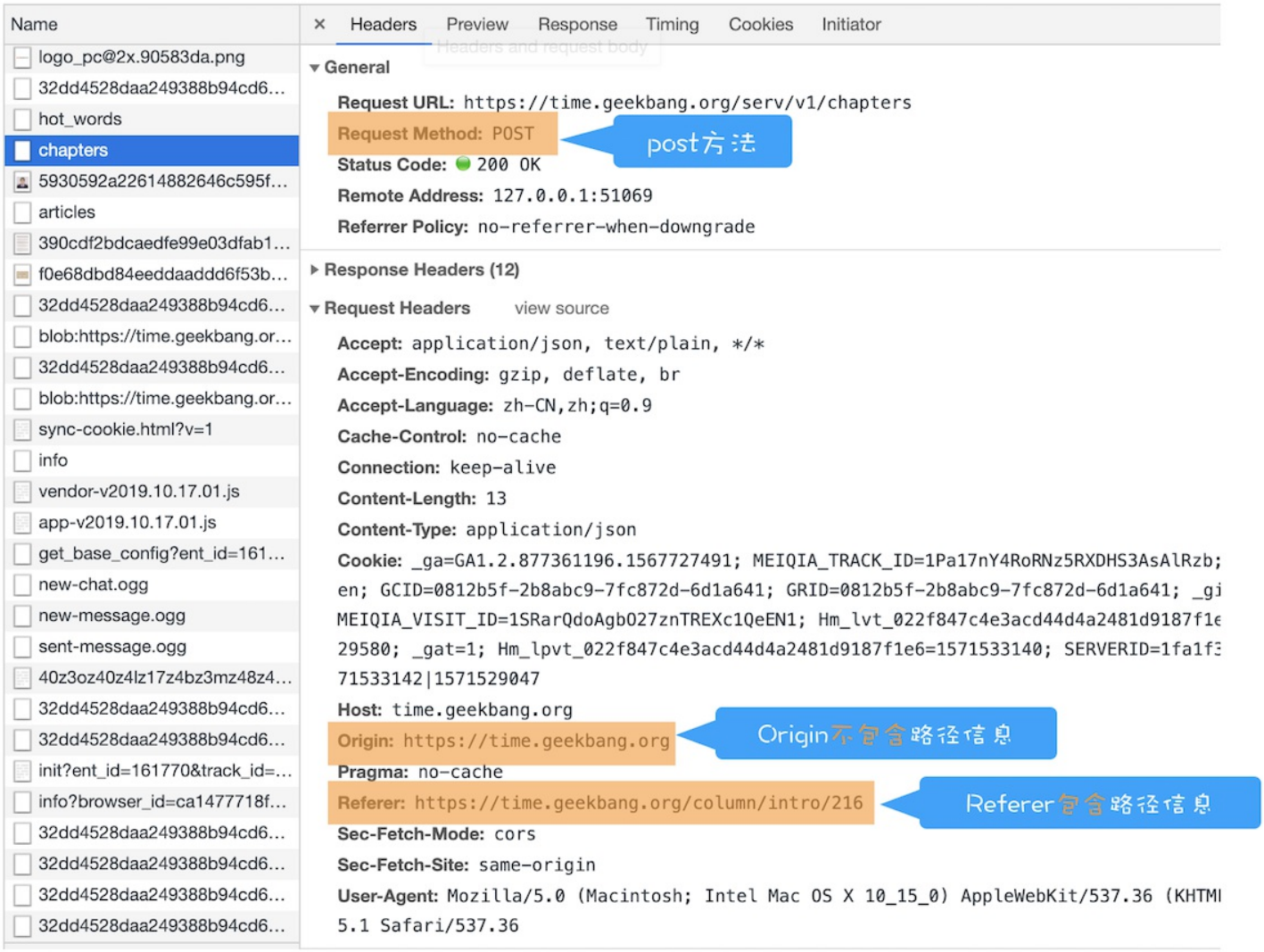
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

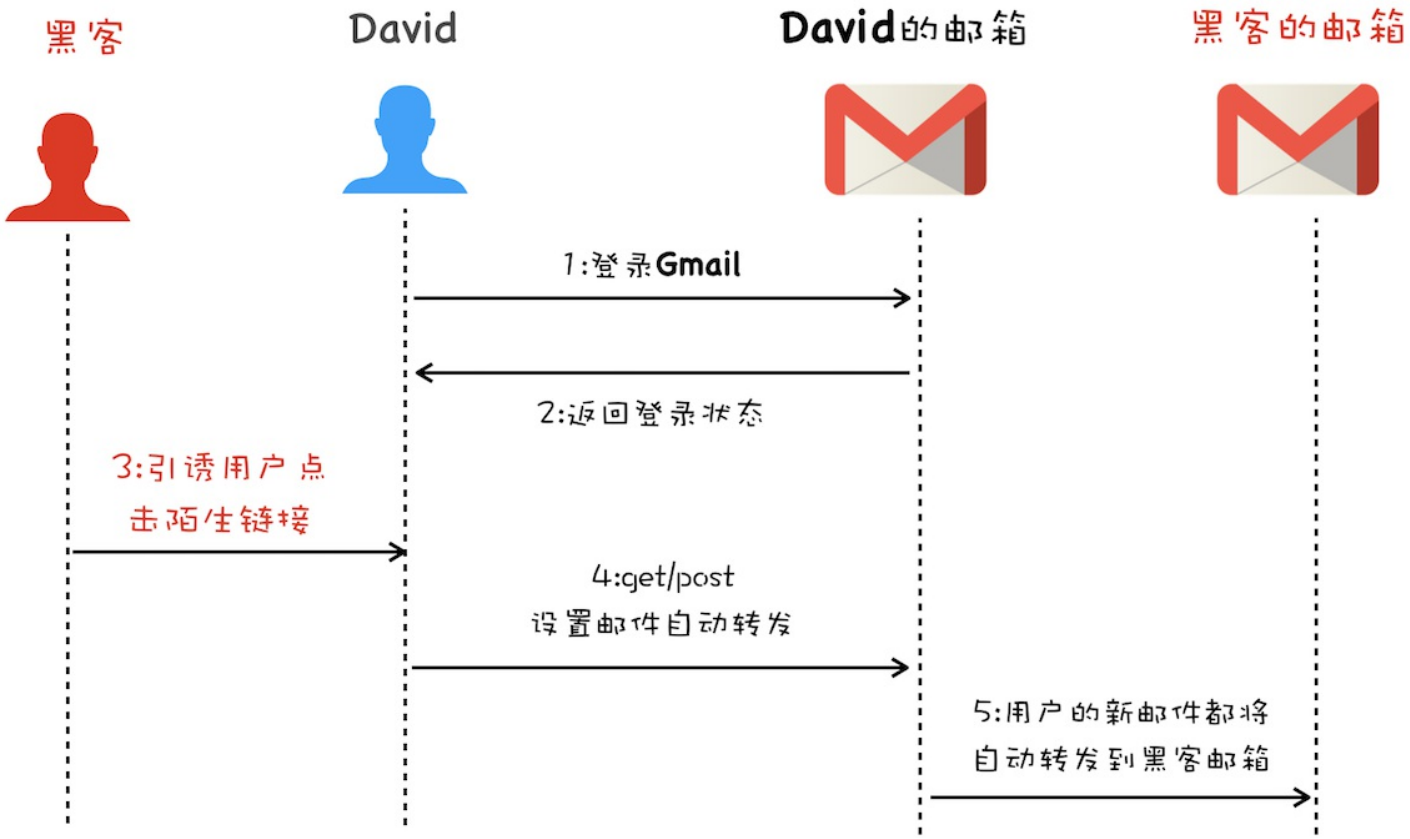
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

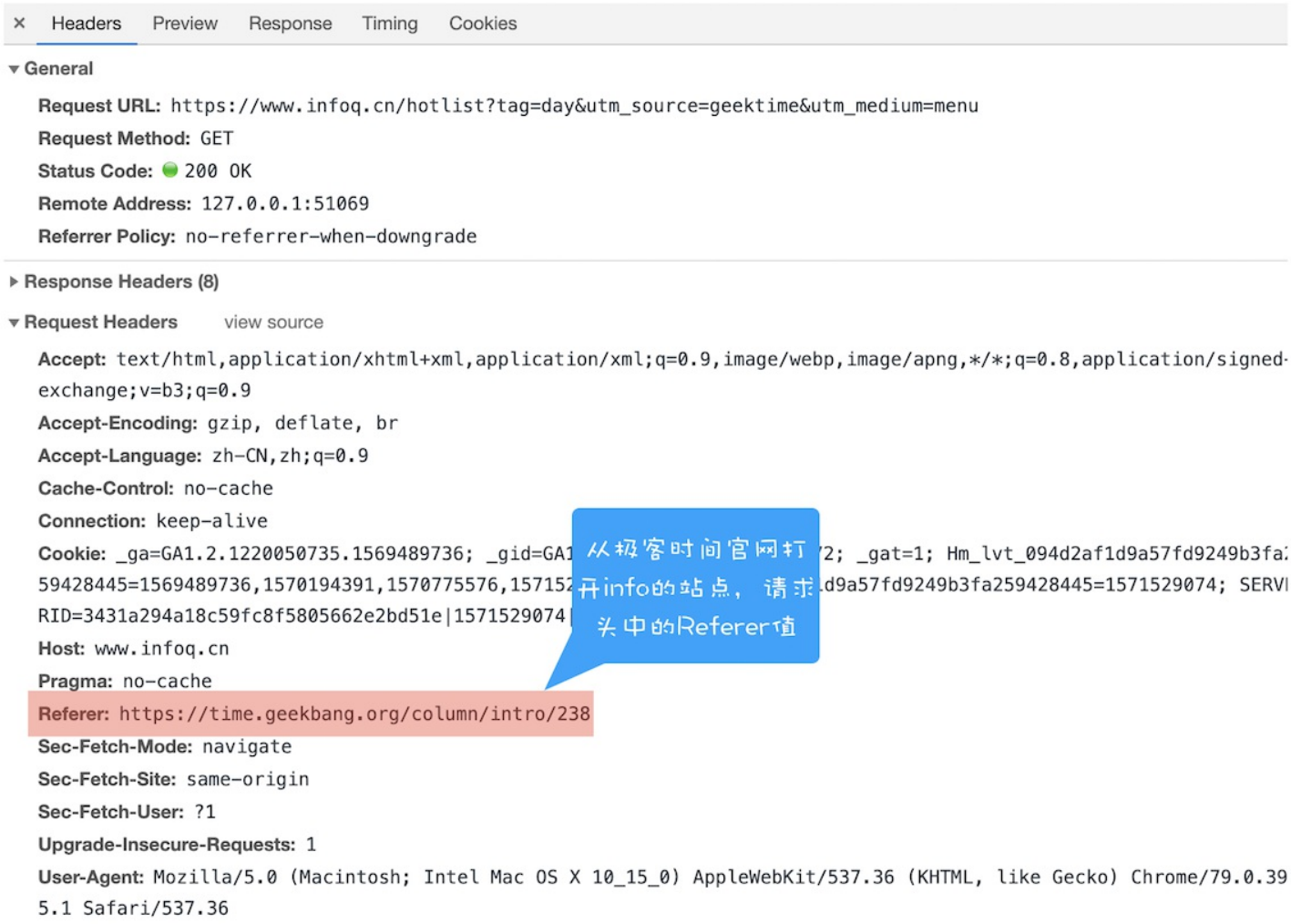
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

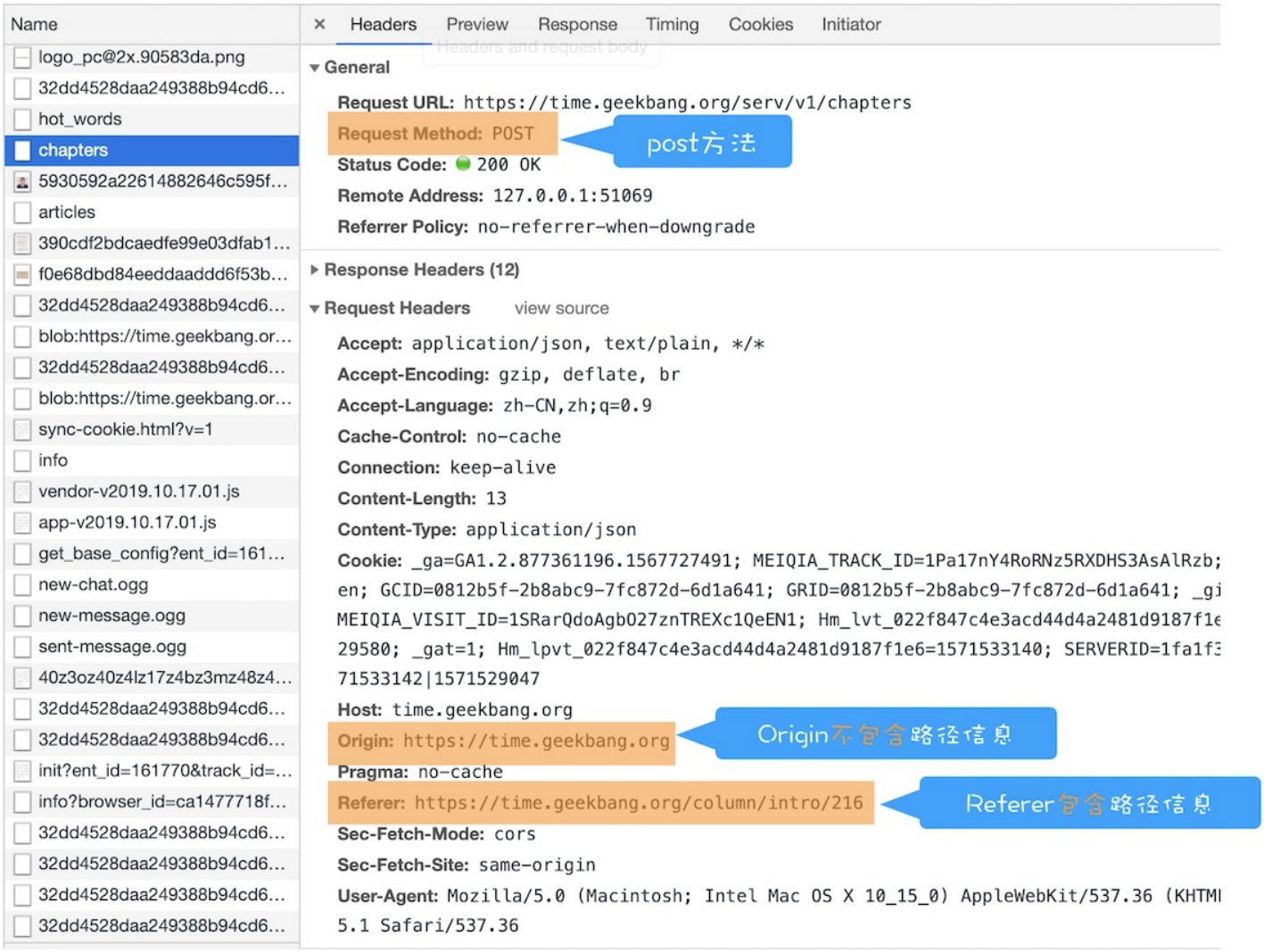
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

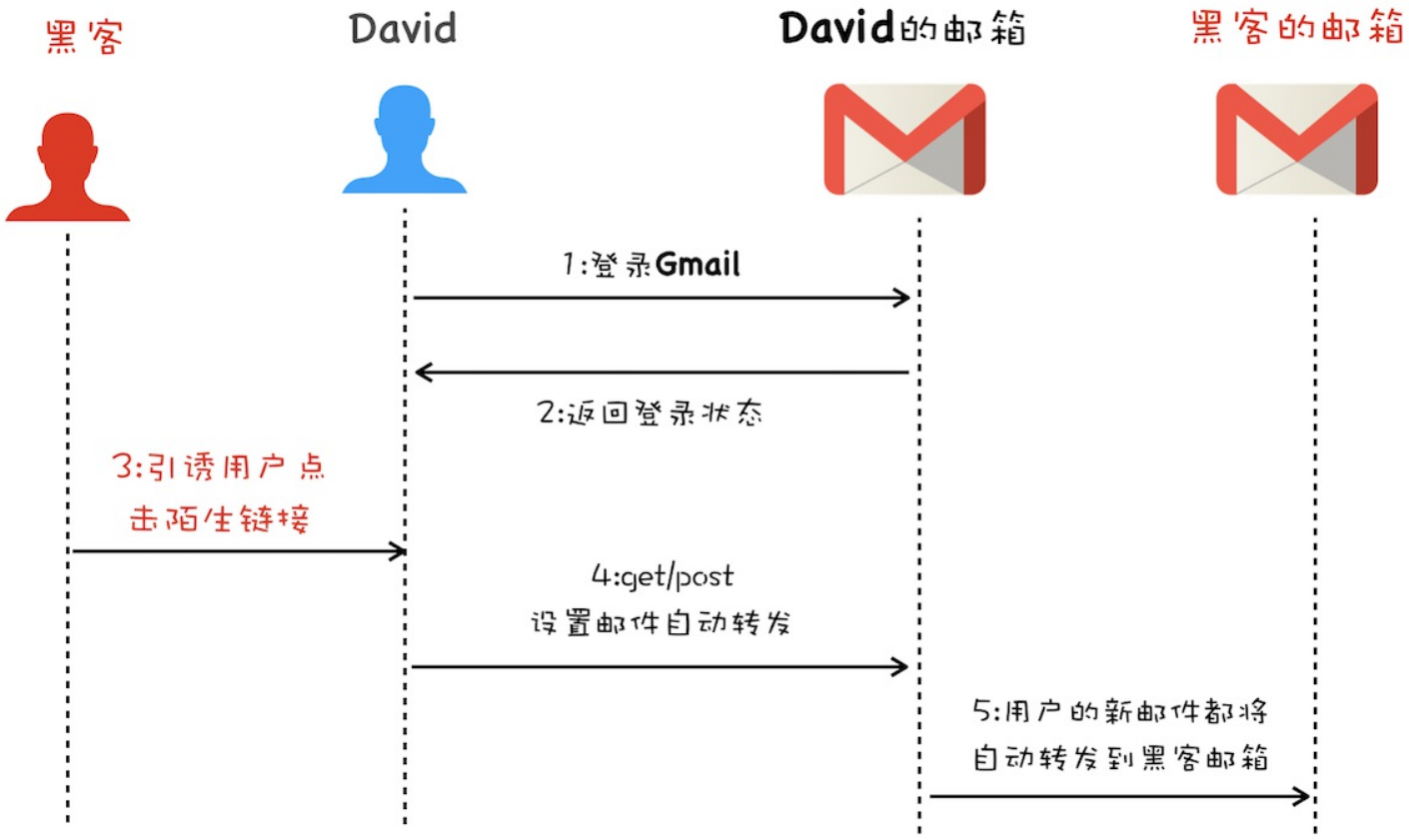
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

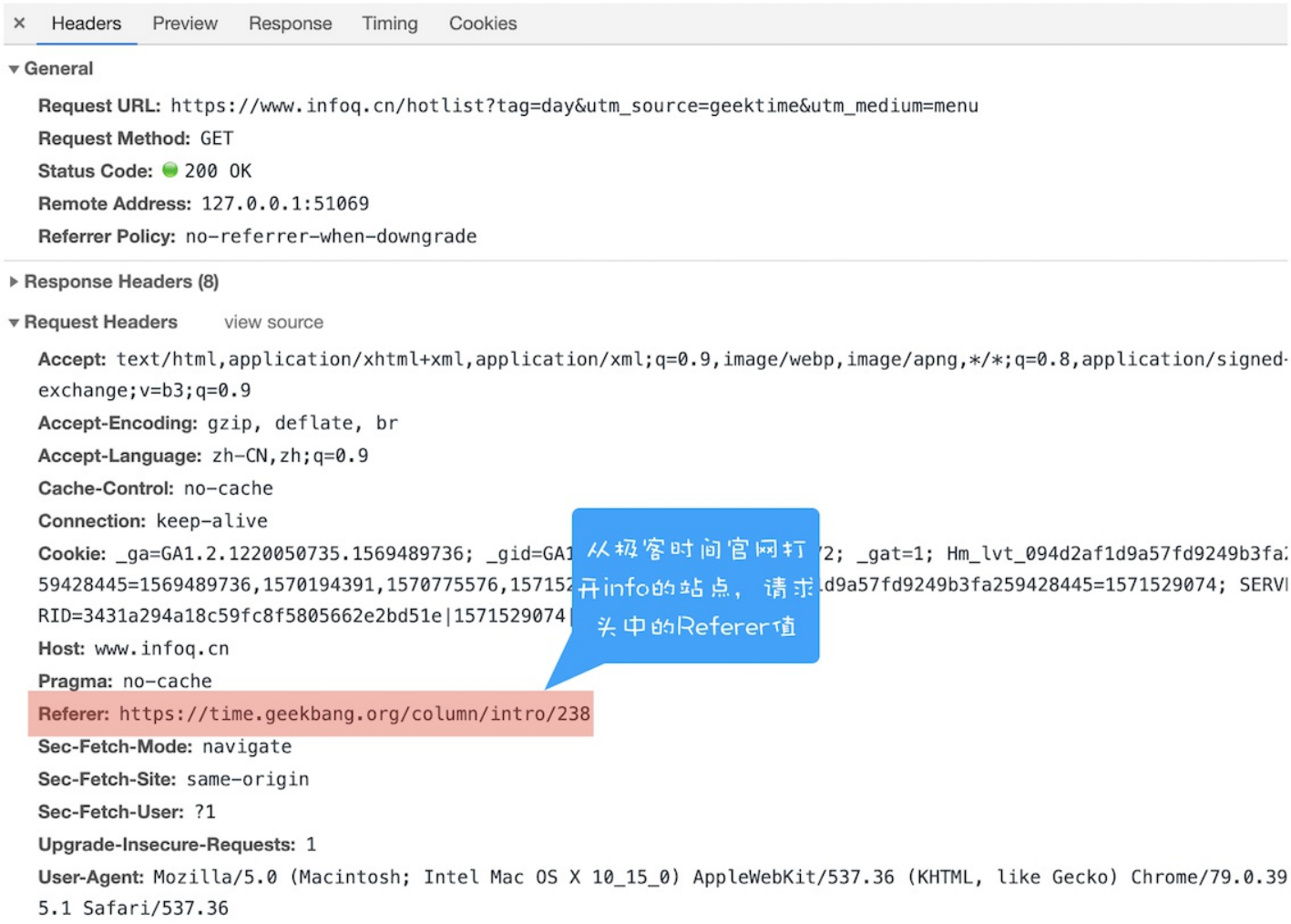
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

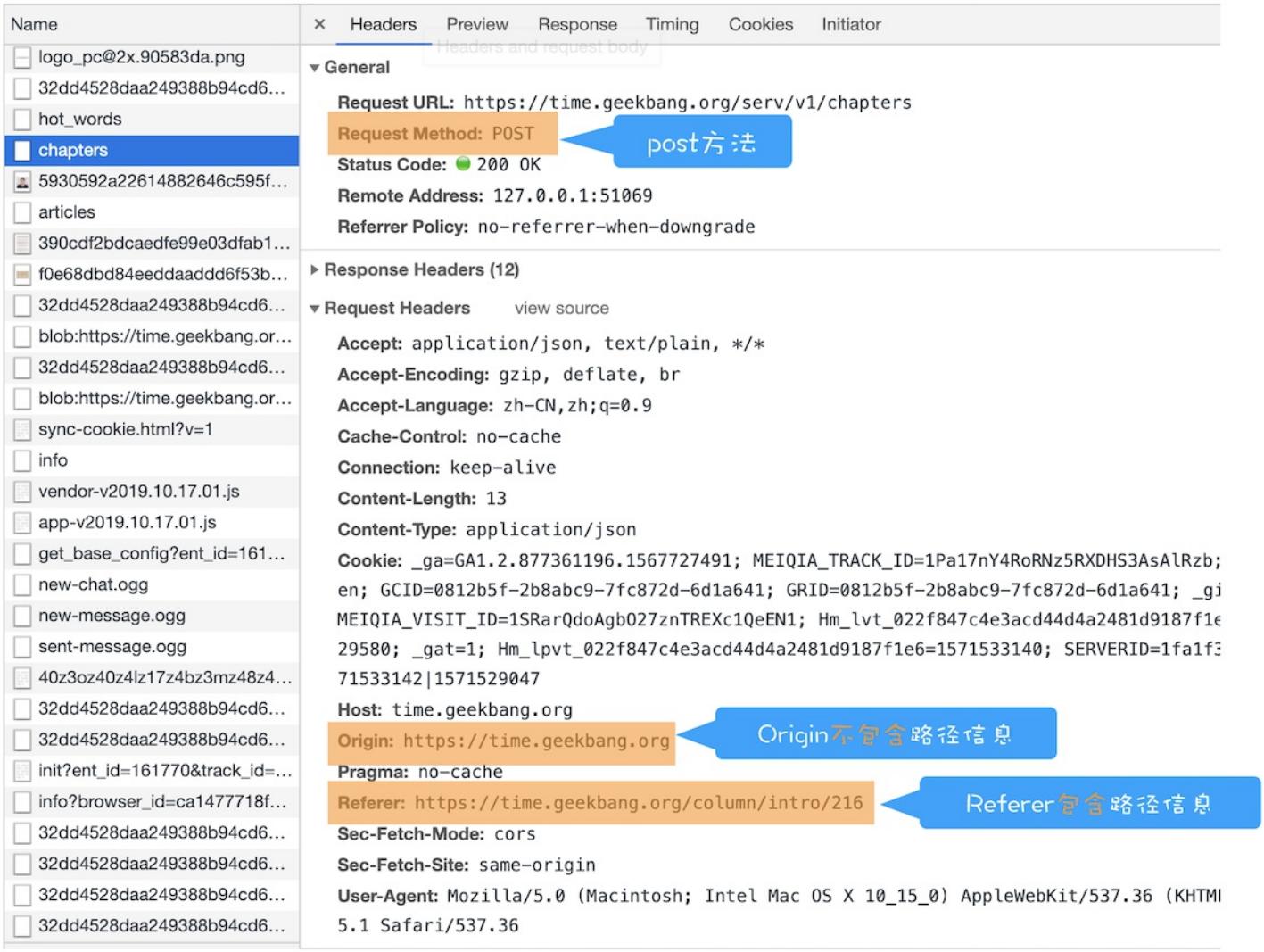
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

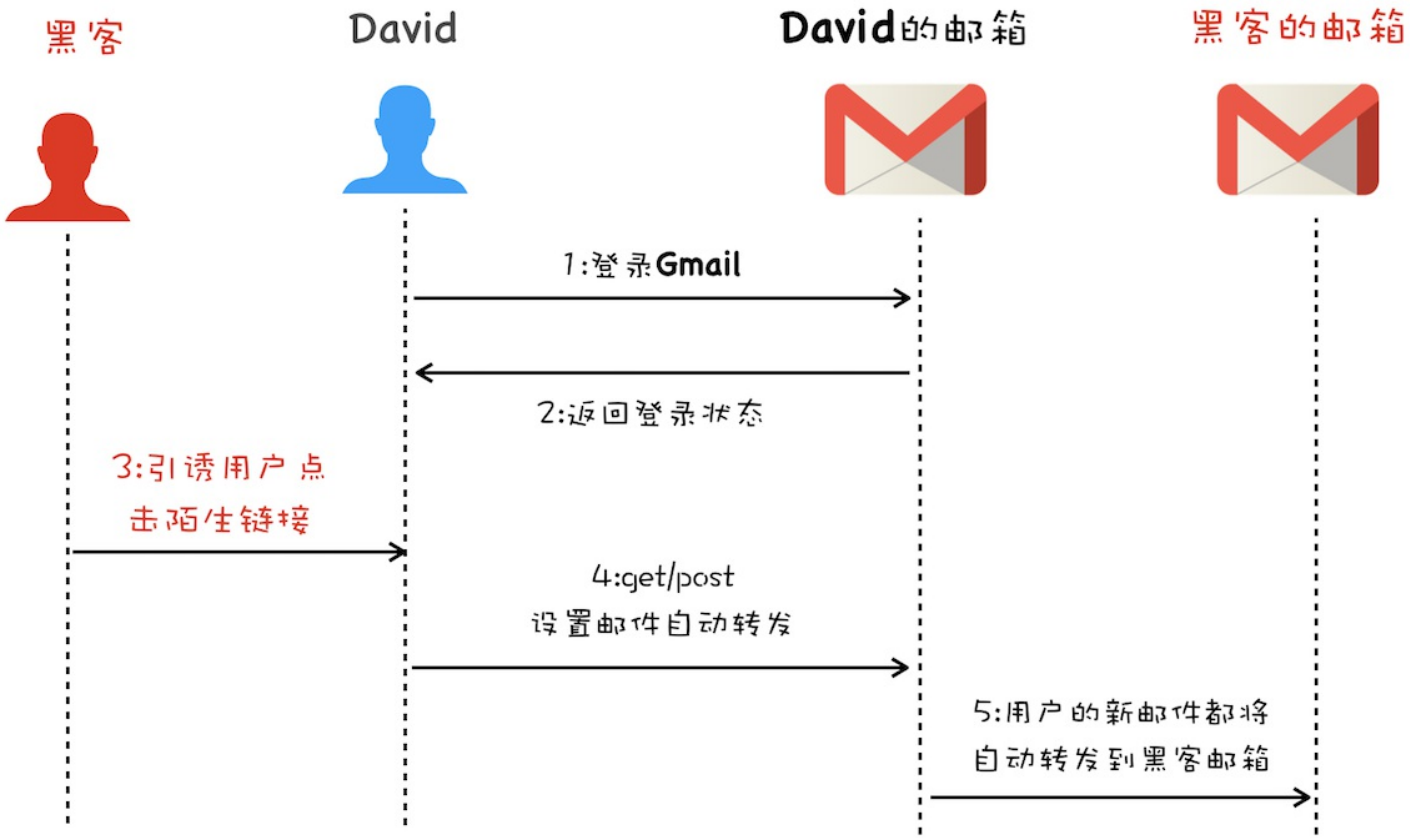
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

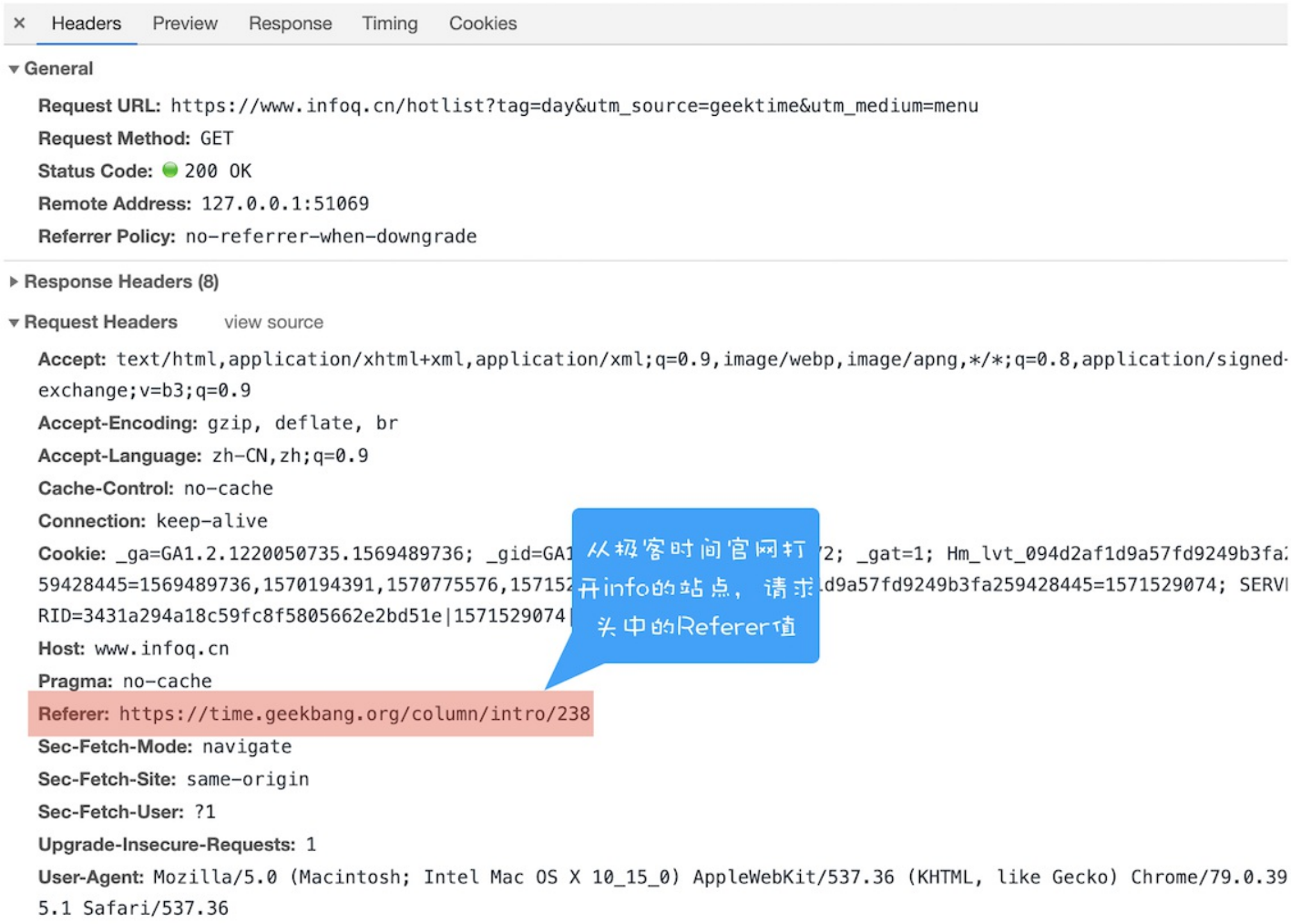
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

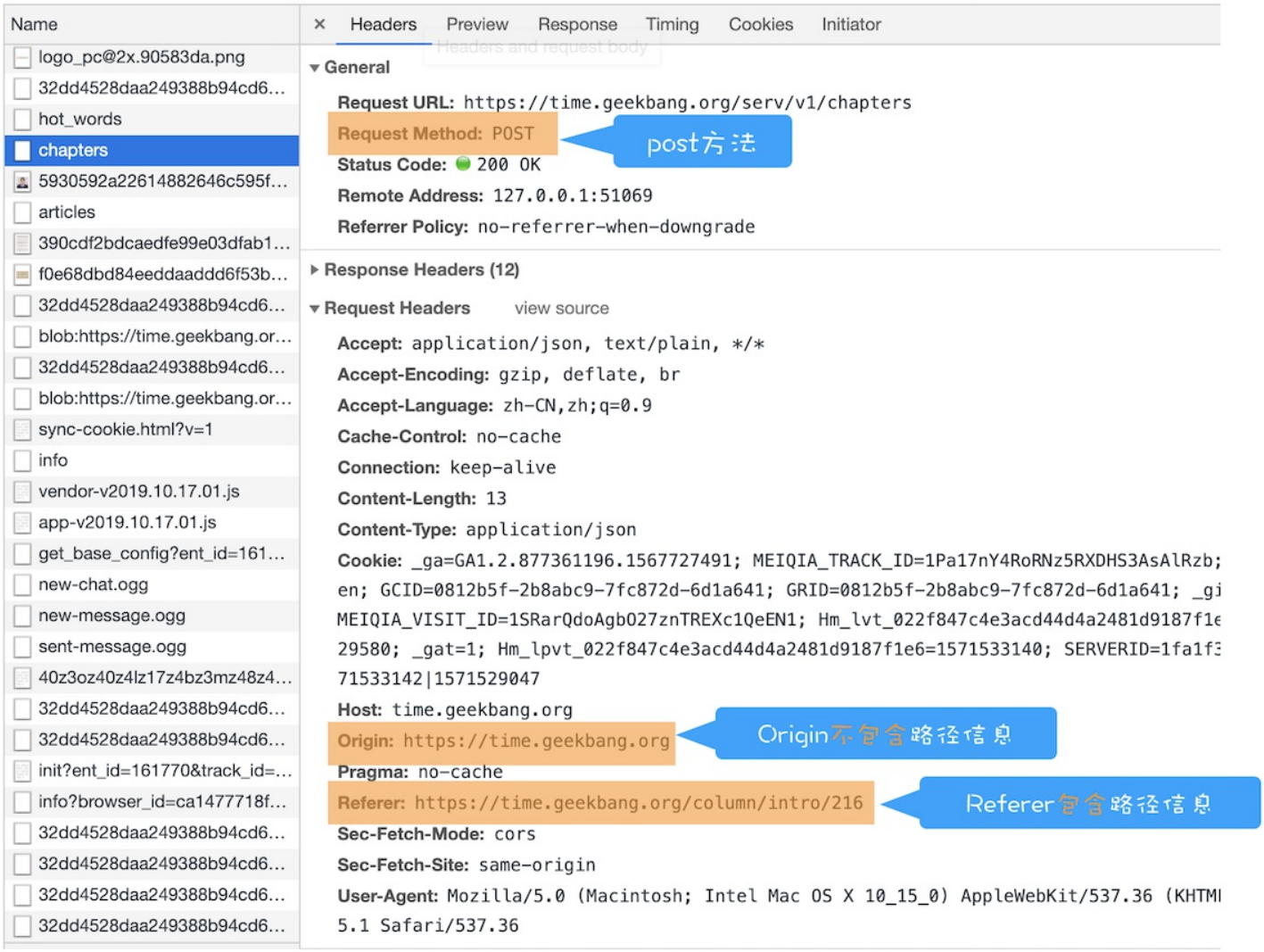
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

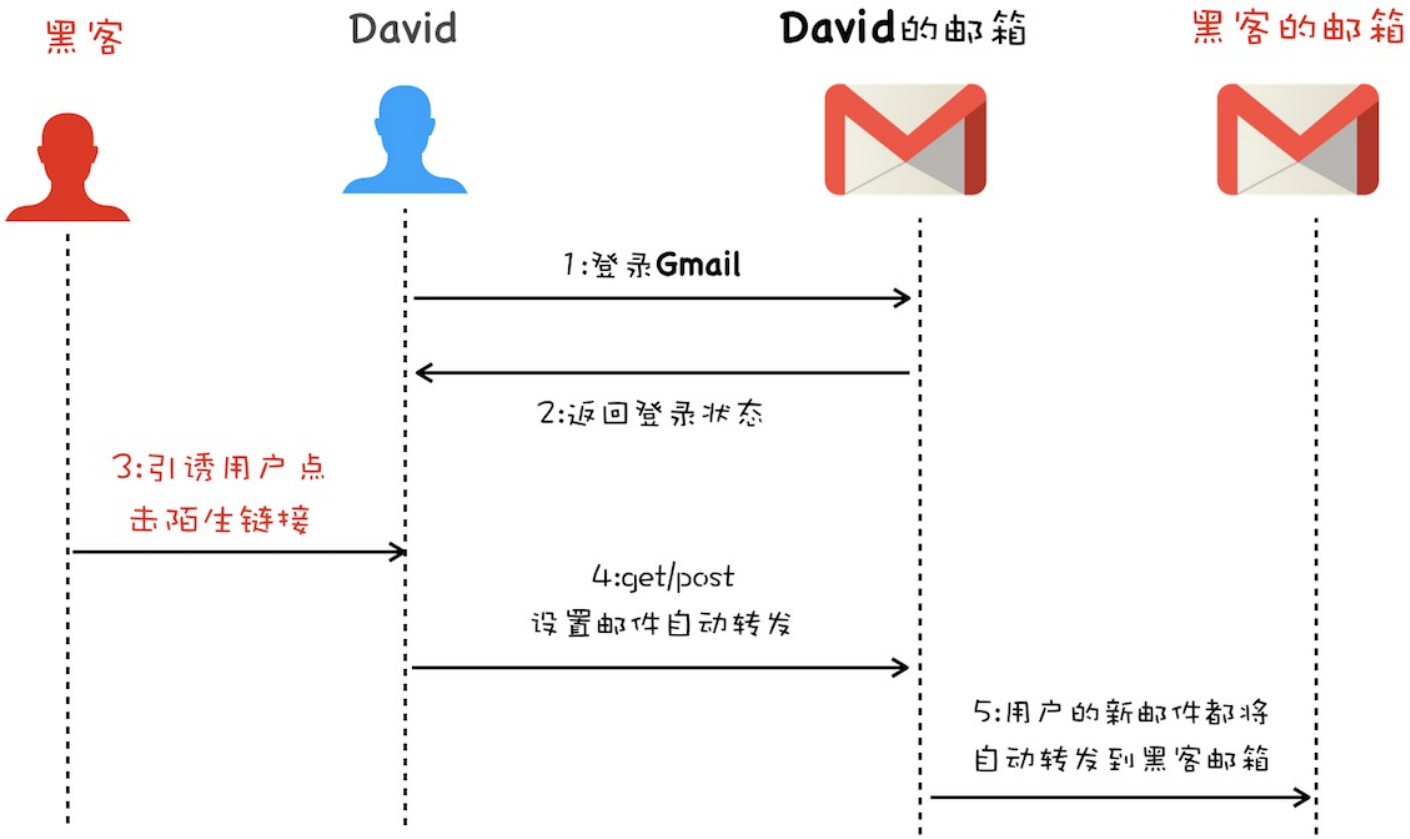
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

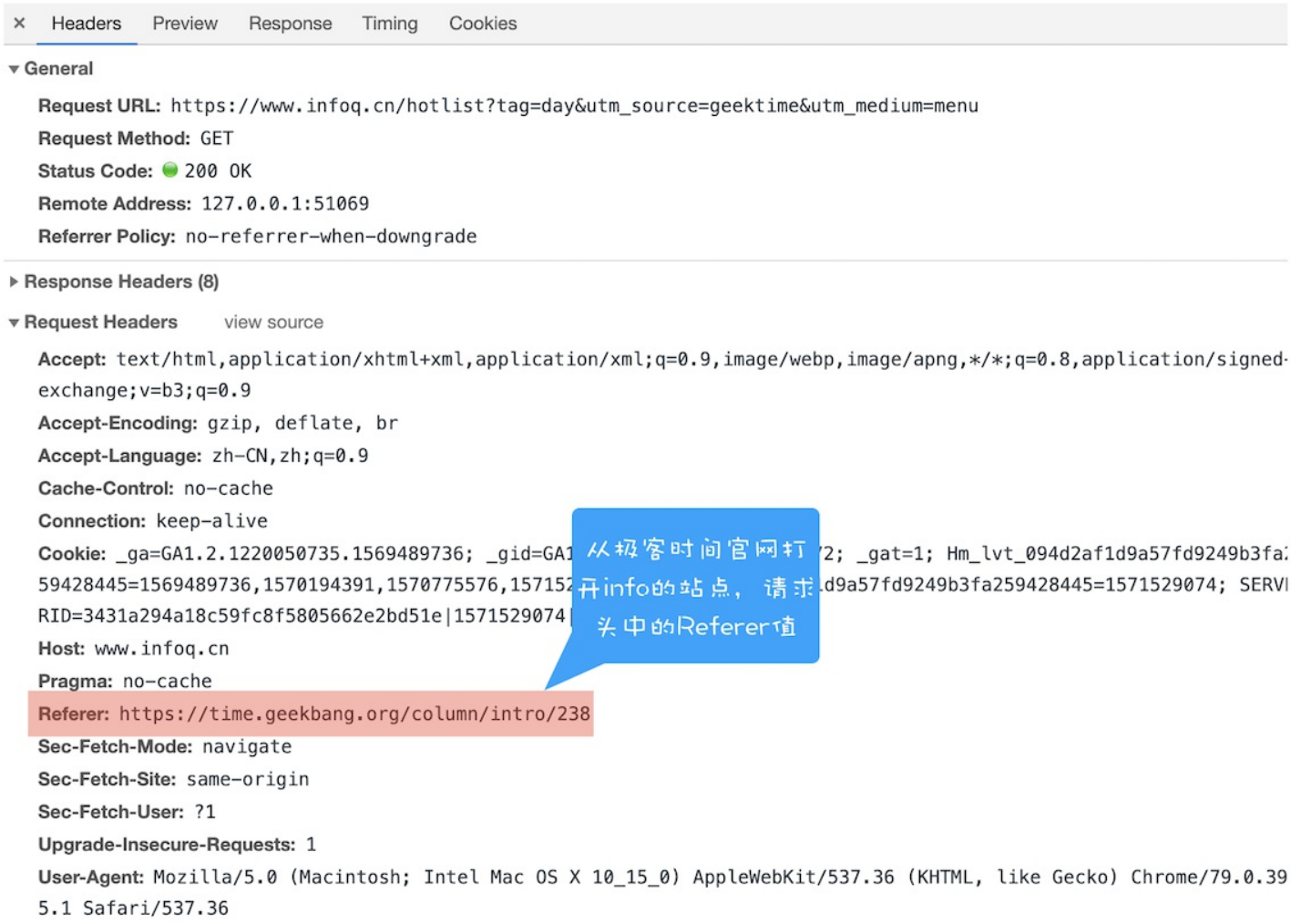
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

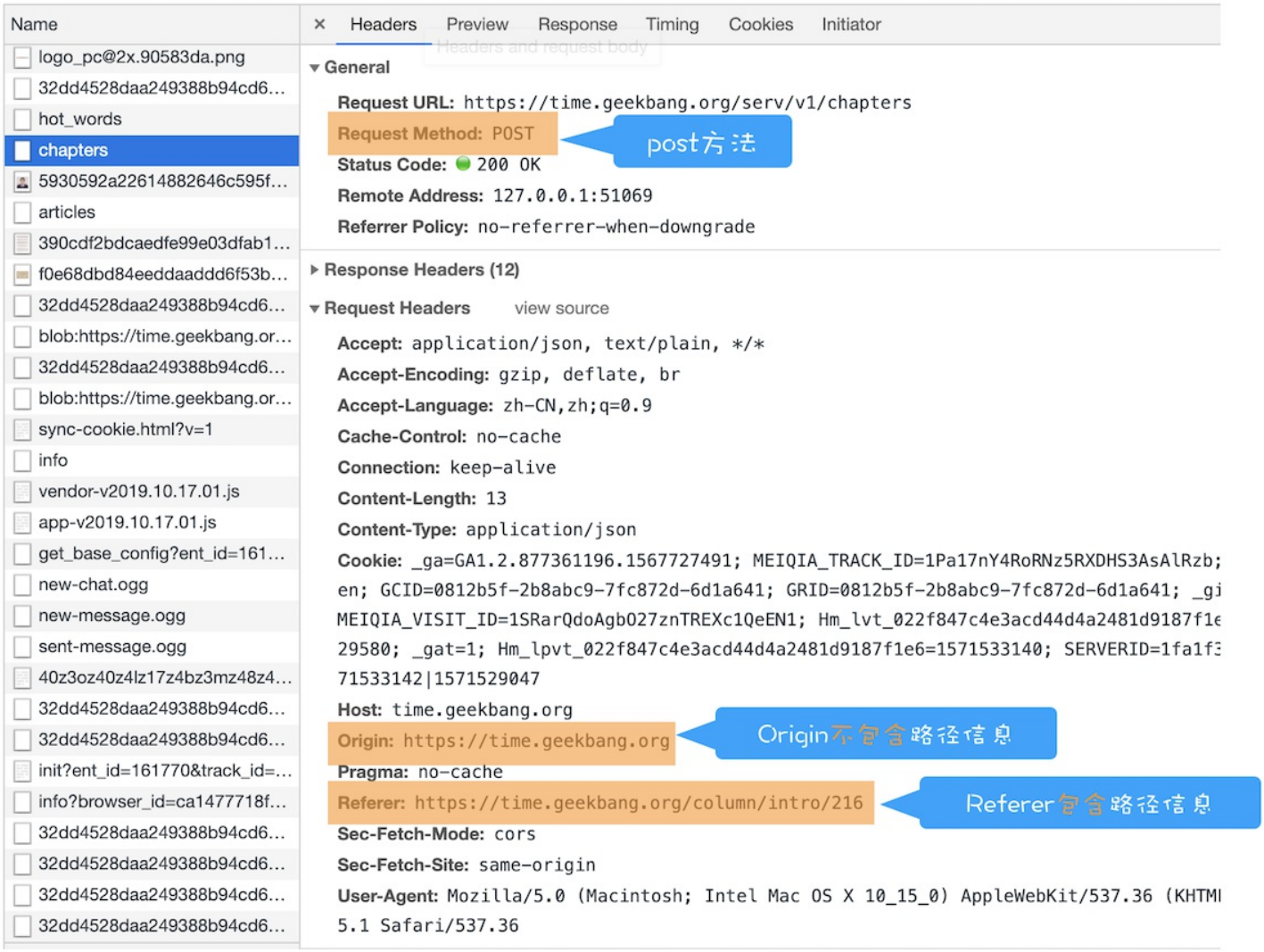
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

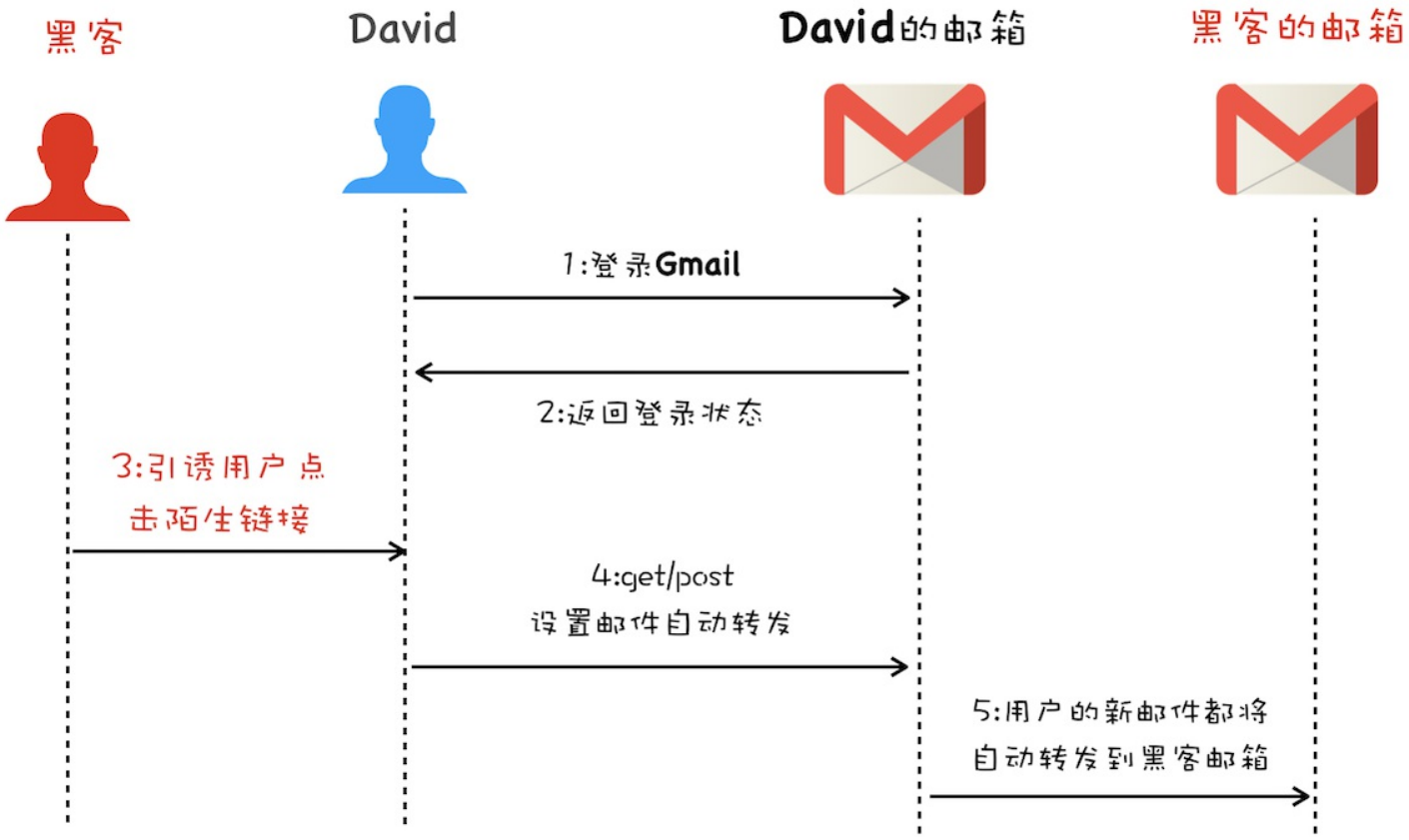
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

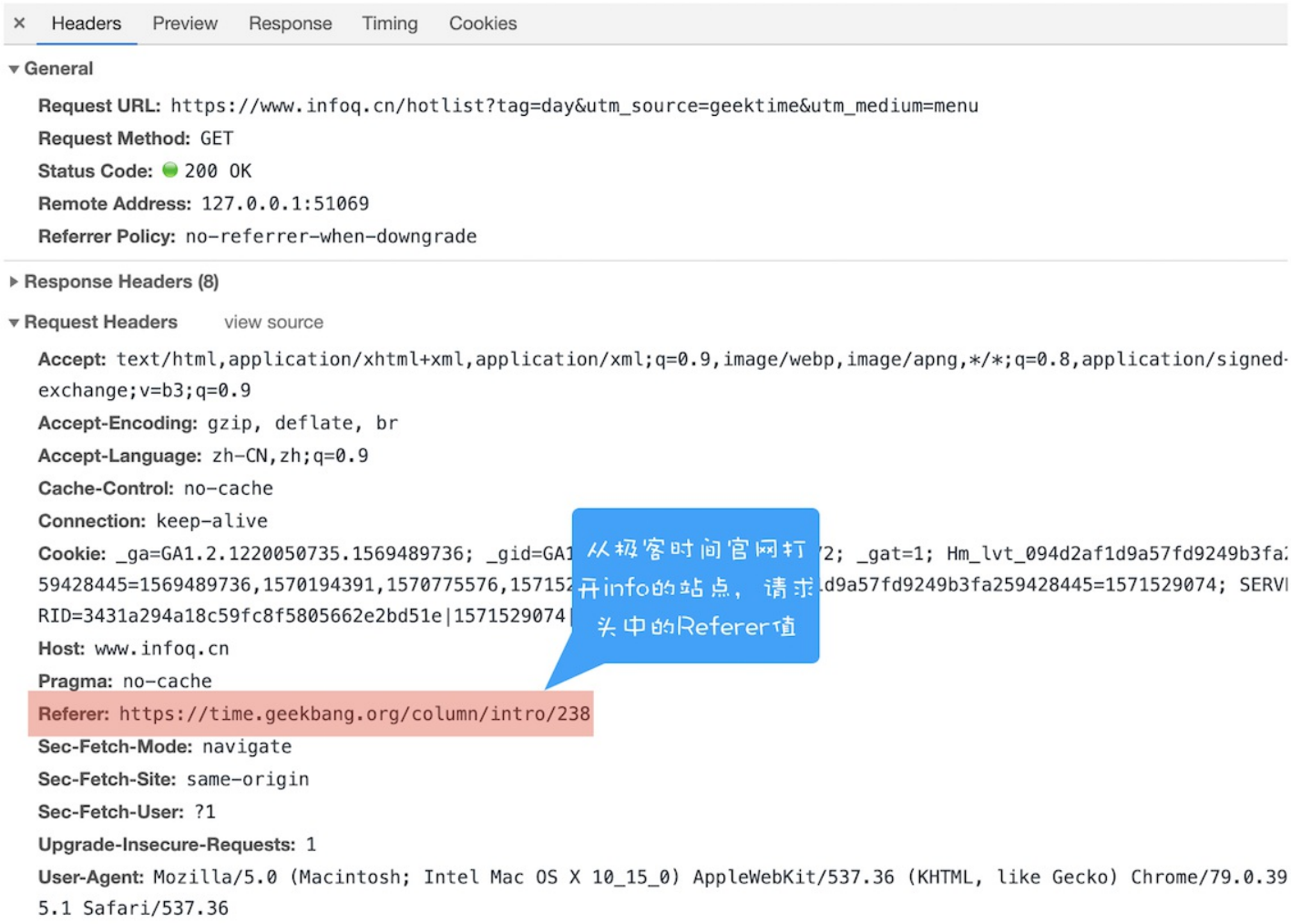
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

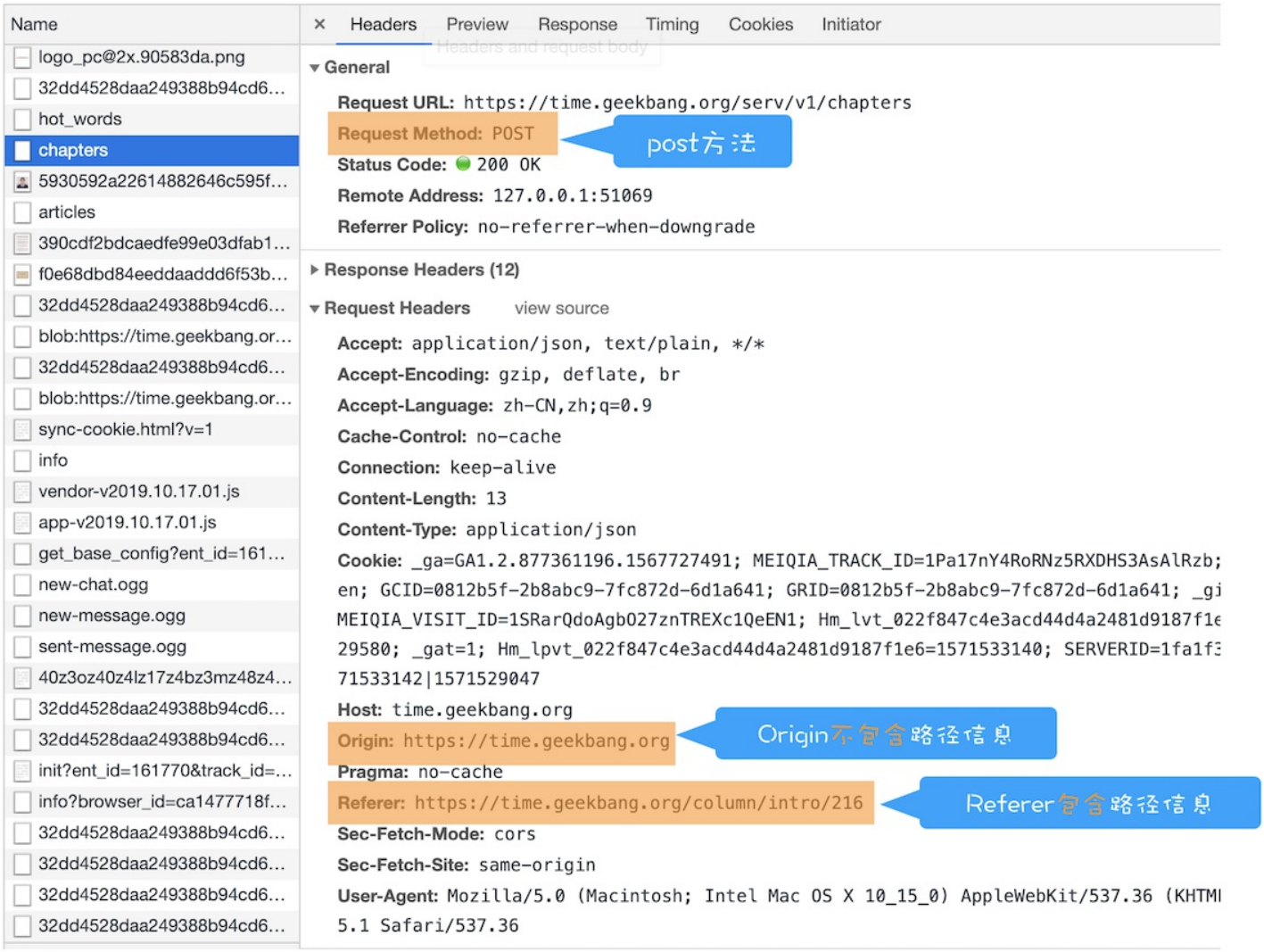
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

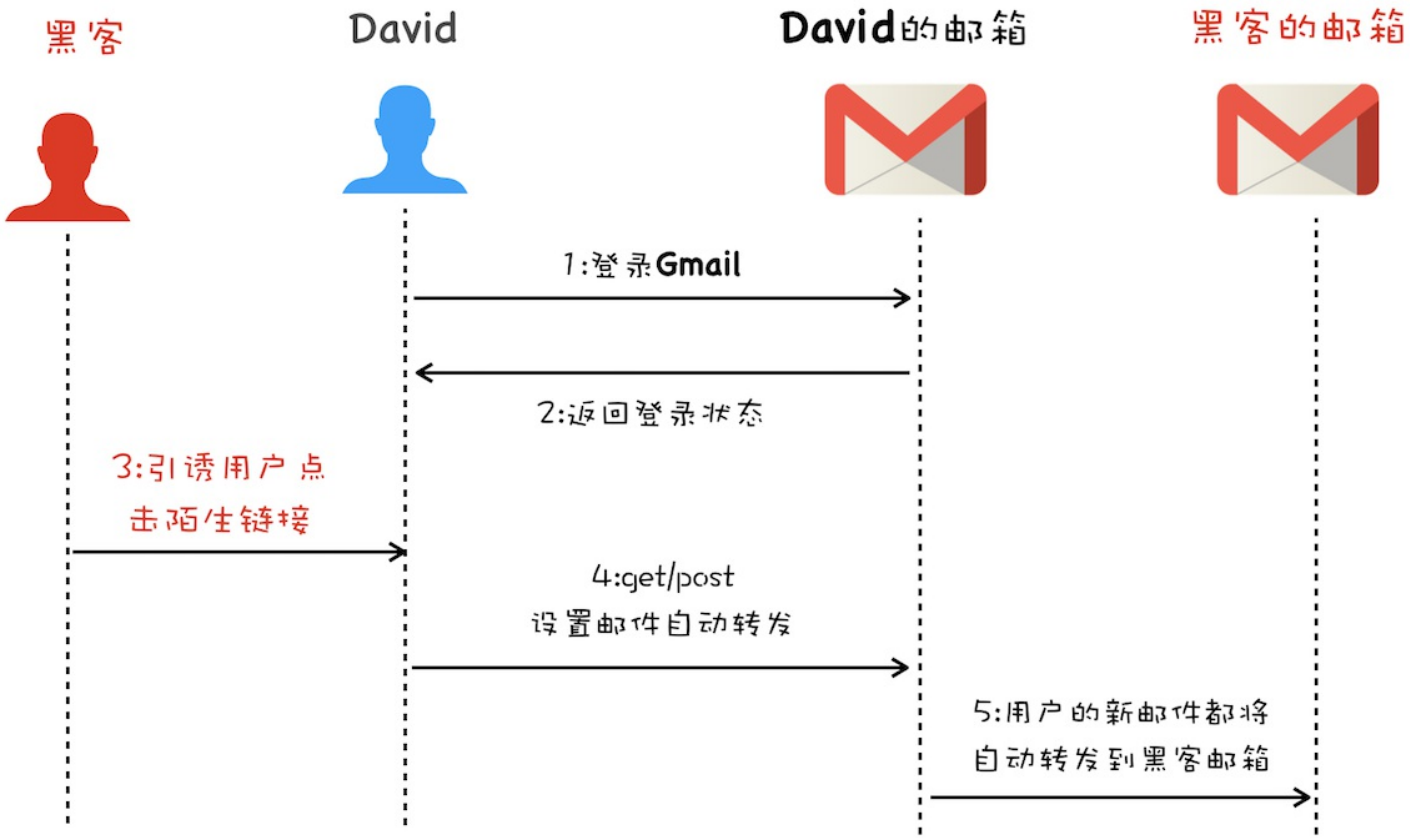
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

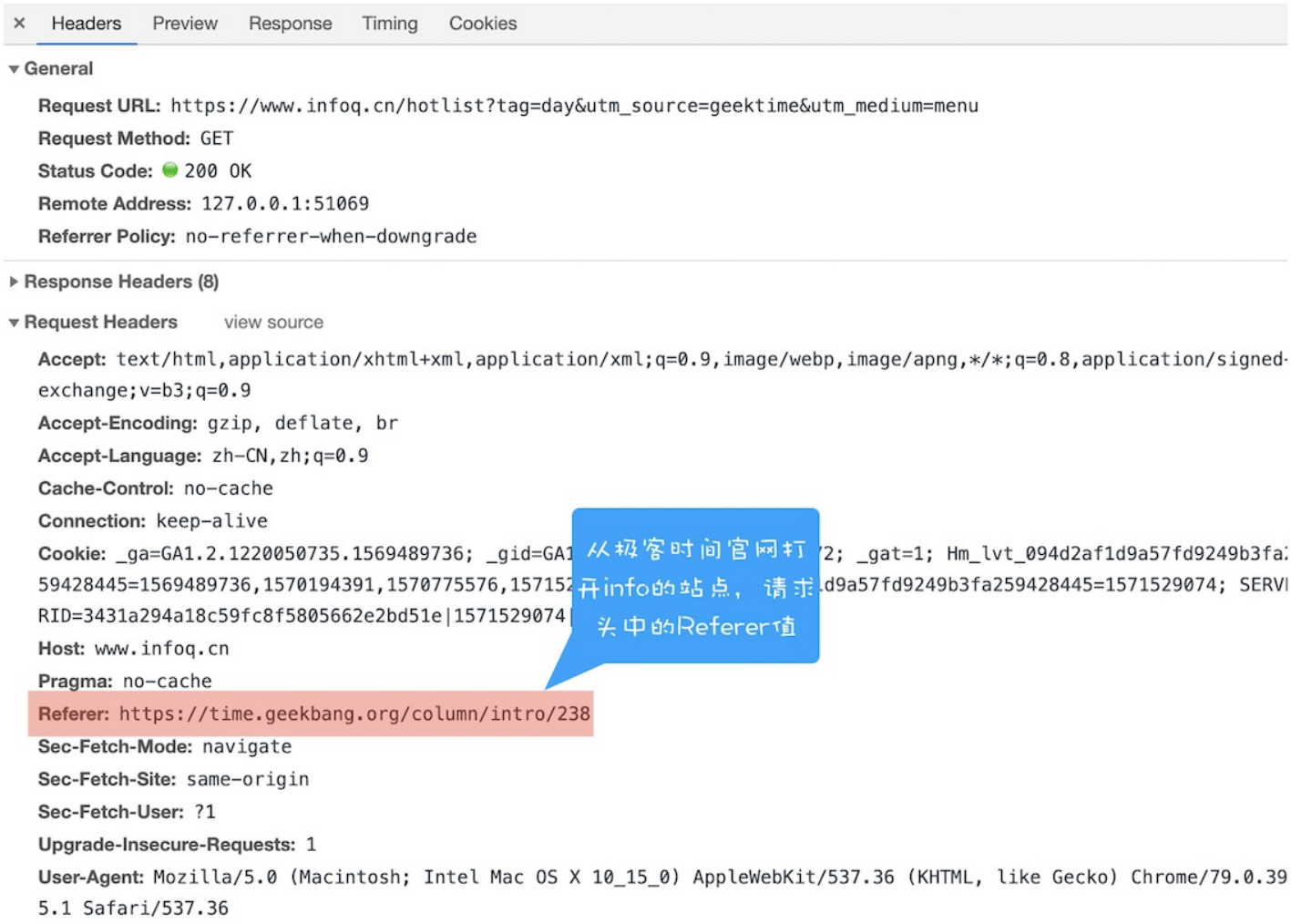
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

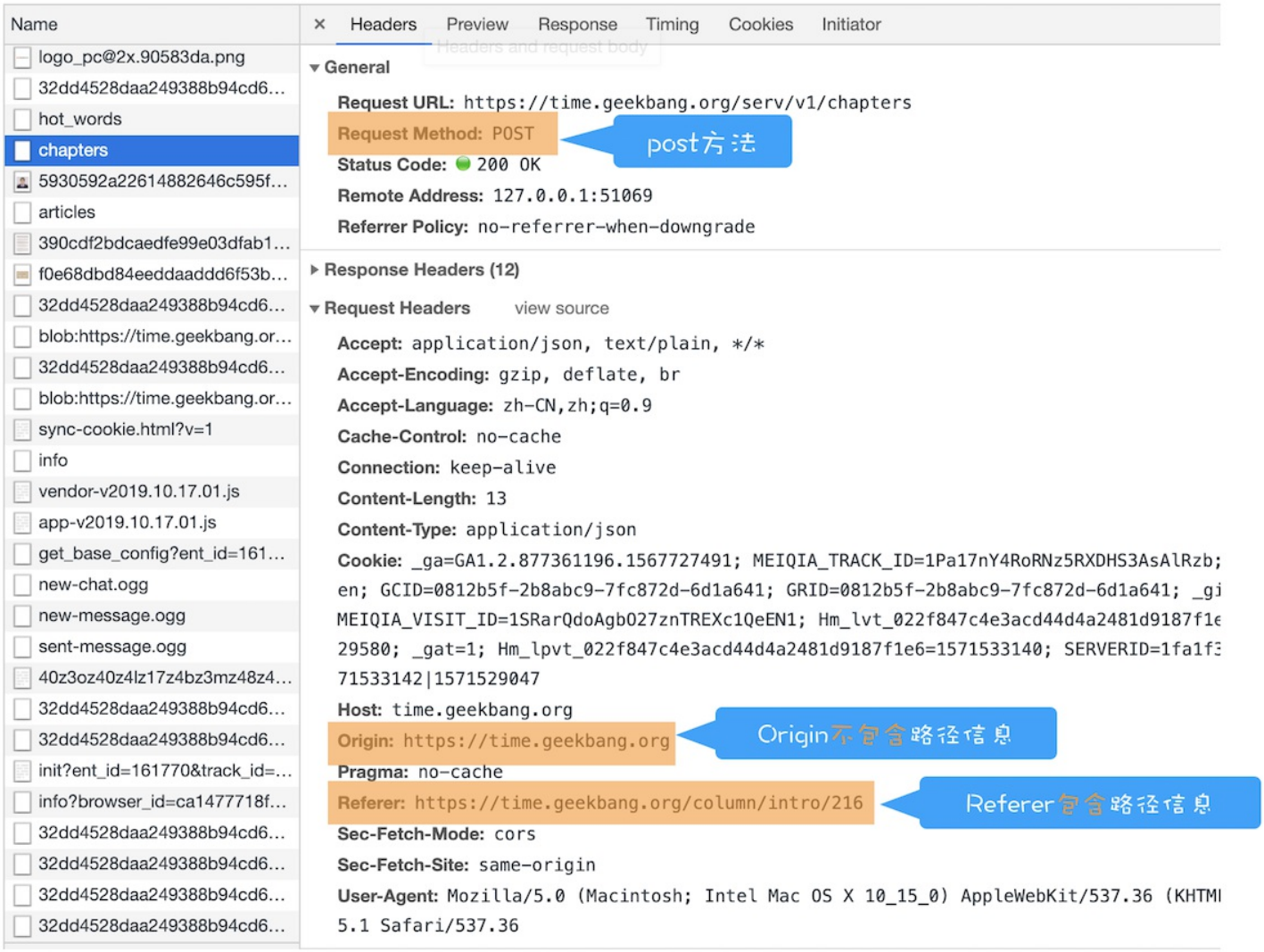
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。

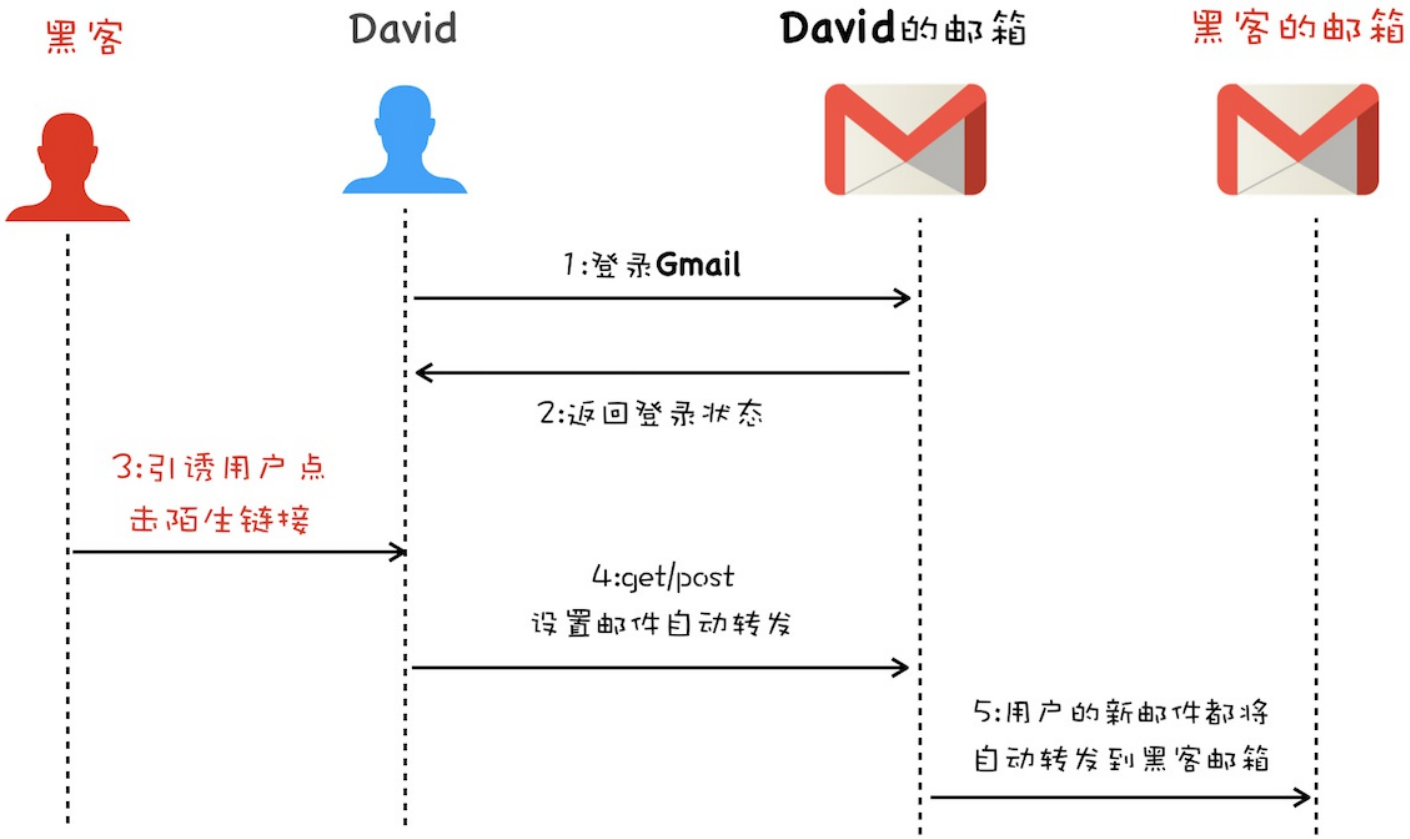
在上一篇文章中我们讲到了XSS攻击，XSS 的攻击方式是黑客往用户的页面中注入恶意脚本，然后再通过恶意脚本将用户页面的数据上传到黑客的服务器上，最后黑客再利用这些数据进行一些恶意操作。XSS攻击能够带来很大的破坏性，不过另外一种类型的攻击也不容忽视，它就是我们今天要聊的CSRF攻击。

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攻击的典型案例来分析下，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：

```
#同时支持POST和Get
#接口
https://time.geekbang.org/sendcoin
#参数
##目标用户
user
##目标金额
```

```
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：

```
<!DOCTYPE html>
<html>
  <body>
    <h1>黑客的站点：CSRF攻击演示</h1>
    <img src="https://time.geekbang.org/sendcoin?user=hacker&number=100">
  </body>
</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：

```
<!DOCTYPE html>
<html>
<body>
  <h1>黑客的站点：CSRF攻击演示</h1>
  <form id='hacker-form' action="https://time.geekbang.org/sendcoin" method=POST>
    <input type="hidden" name="user" value="hacker" />
    <input type="hidden" name="number" value="100" />
  </form>
  <script> document.getElementById('hacker-form').submit(); </script>
</body>
</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：

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

```
set-cookie: 1P_JAR=2019-10-20-06; expires=Tue, 19-Nov-2019 06:36:21 GMT; path=/; domain=.google.com; SameSite=none
```

**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 。

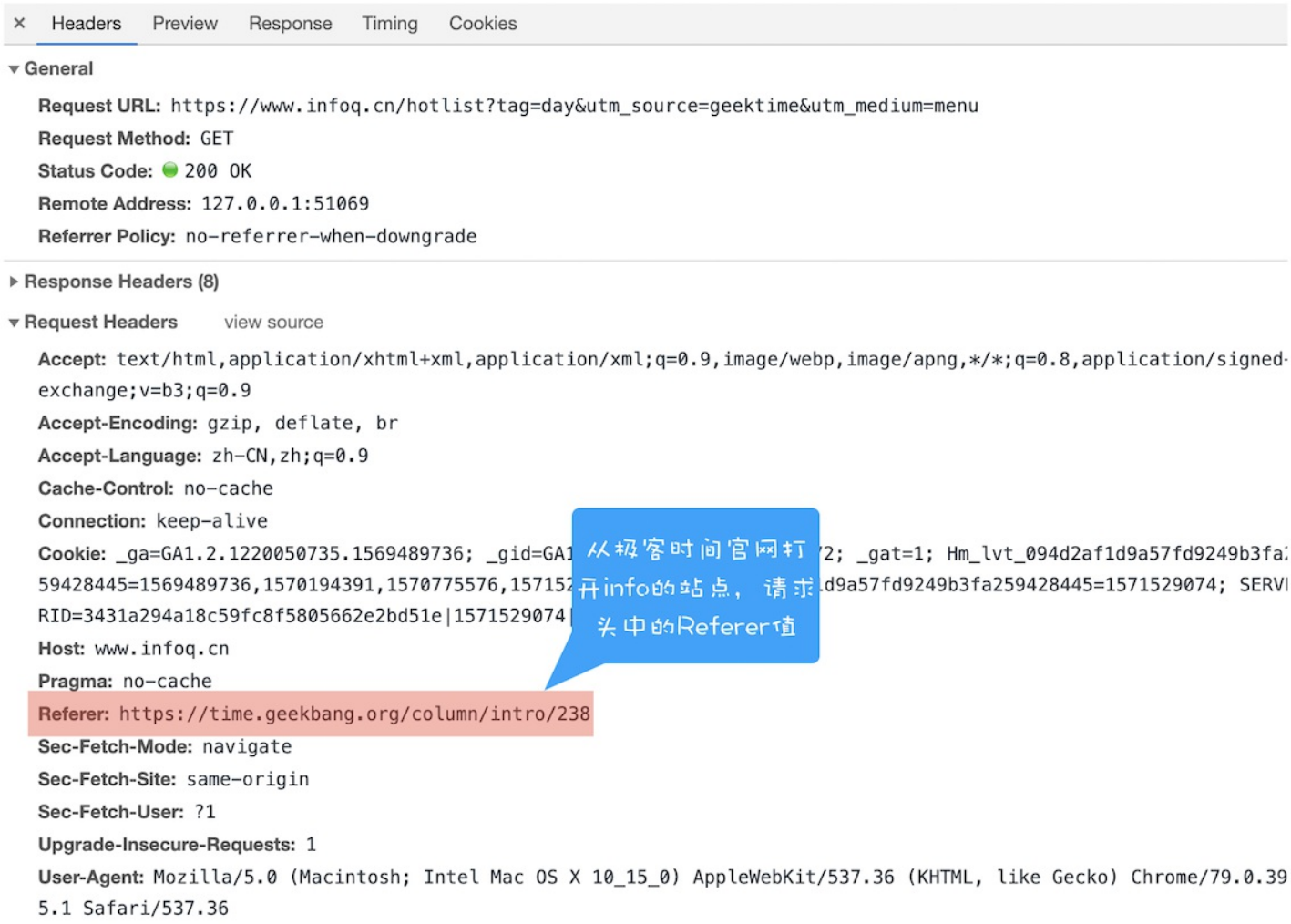
对于防范CSRF攻击，我们可以针对实际情况将一些关键的Cookie设置为Strict或者Lax模式，这样在跨站点请求时，这些关键的Cookie就不会被发送到服务器，从而使得黑客的CSRF攻击失效。

## 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。

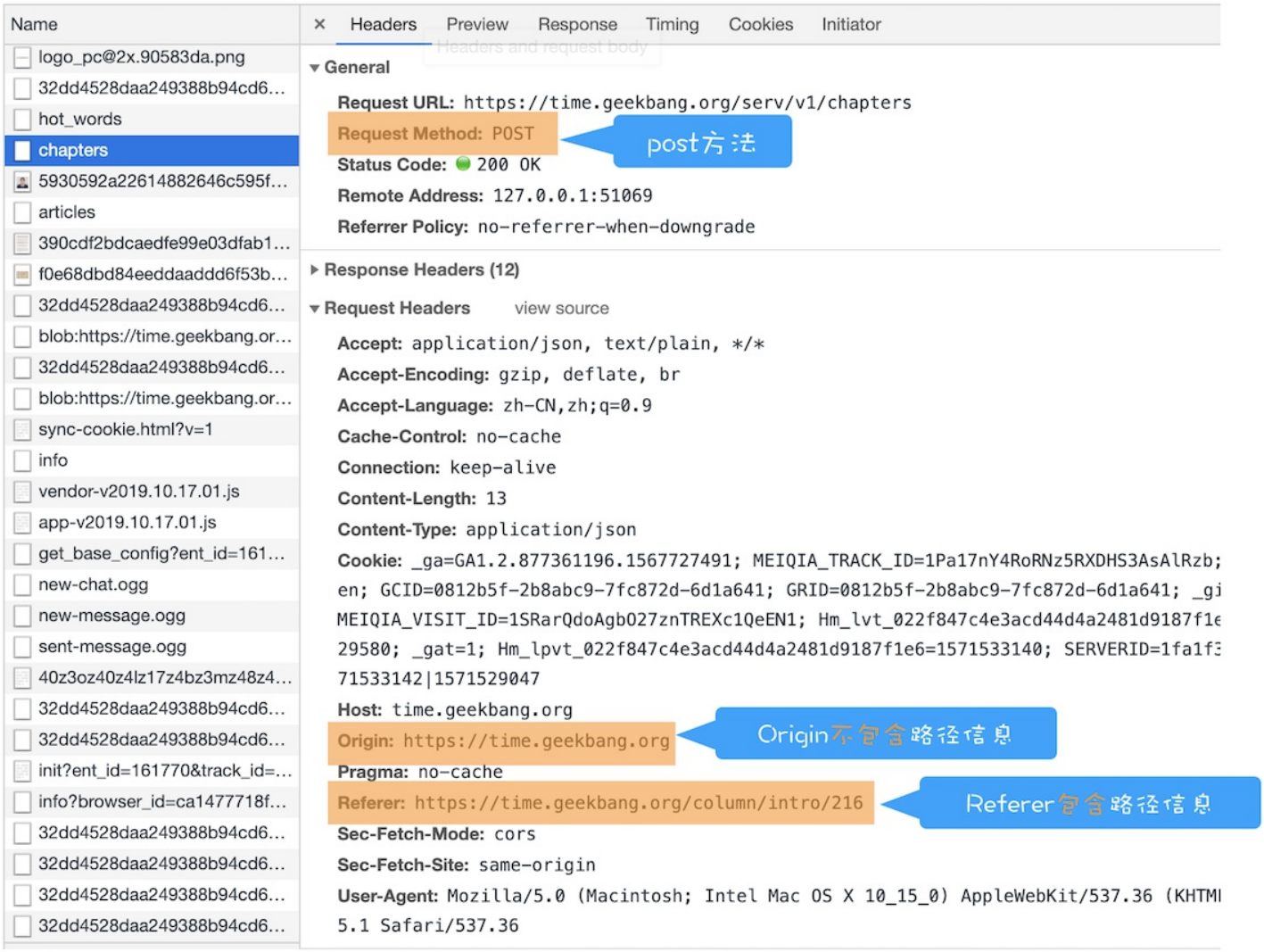
**Referer是HTTP请求头中的一个字段，记录了该HTTP请求的来源地址**。比如我从极客时间的官网打开了InfoQ的站点，那么请求头中的Referer值是极客时间的URL，如下图：




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、Fecth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：




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：

```
<!DOCTYPE html>
<html>
<body>
    <form action="https://time.geekbang.org/sendcoin" method="POST">
      <input type="hidden" name="csrf-token" value="nc98P987bcpncYhoadjoiydc9ajDlcn">
      <input type="text" name="user">
      <input type="text" name="number">
      <input type="submit">
    </form>
</body>
</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。