实验环境：

平台环境：Ubuntu16-04，64bit

Ruby 2.5.0

Rails 5.0.7.2

# attack 1 Warn-up exercise: Cookie Theft

## 【攻击要求】

开始网址http://localhost:3000/profile?username=

评分员将提前以user1的身份登录bitbar，然后打开以上的开始网址

你的目标是偷取user1的会话cookie并且将cookie发送到http://localhost:3000/steal\_cookie?cookie=...cookie\_data\_here...

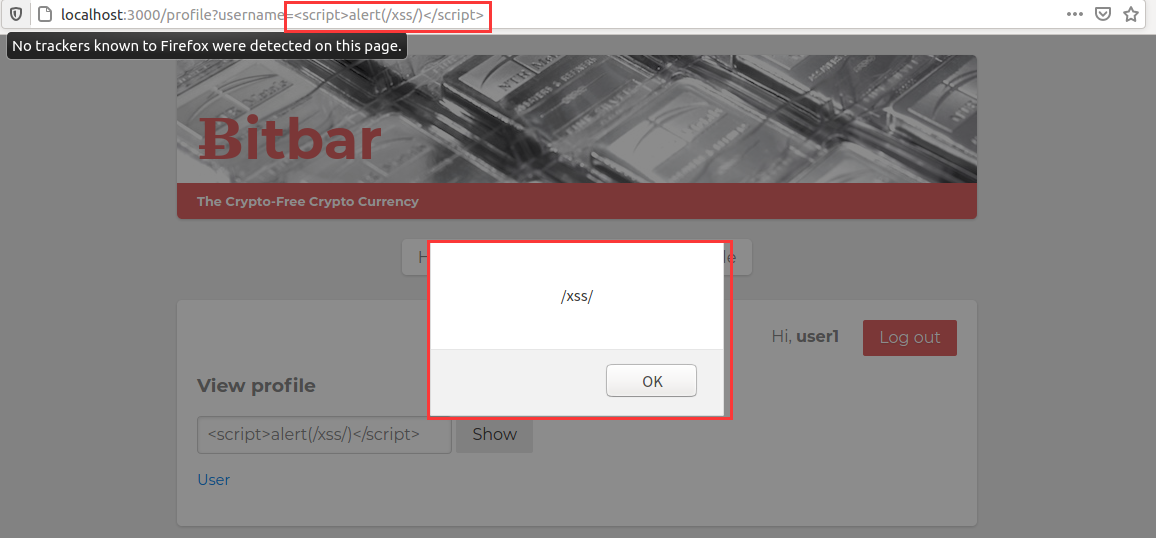
你可以在以下网址上查看最近被偷取的cookiehttp://localhost:3000/view\_stolen\_cookie

请将你的答案写在warmup.txt中

提示：XSS漏洞 <script>alert(/xss/)</script>

## 【漏洞分析】

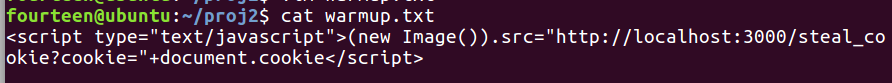
根据提示，打开开始网页（http://localhost:3000/profile?username=）后，登录user1，在view profile框中输入<script>alert(/xss/)</script>，回车，网页弹框如下，说明没有过滤用户输入，存在XSS漏洞。



我们可以利用这个漏洞来劫持cookie，并发送到目标网址。

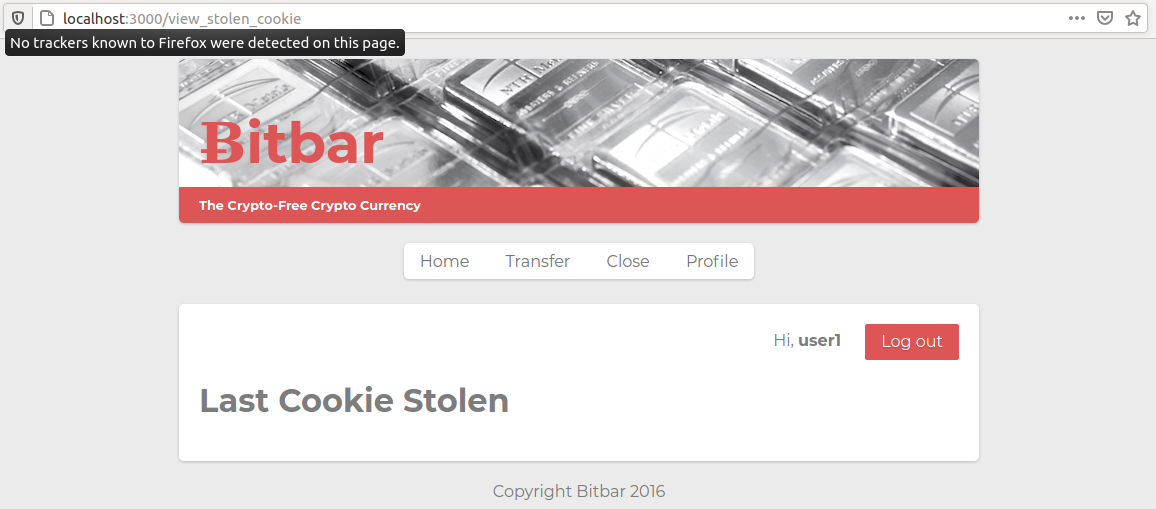
## 【攻击原理及过程】

按照题目前提， user1已经登录目标网页，所以此时cookie已经存在，利用document.cookie属性就可以获得cookie。所以可在view profile框中输入js脚本，脚本内容如下（warmup.txt）：

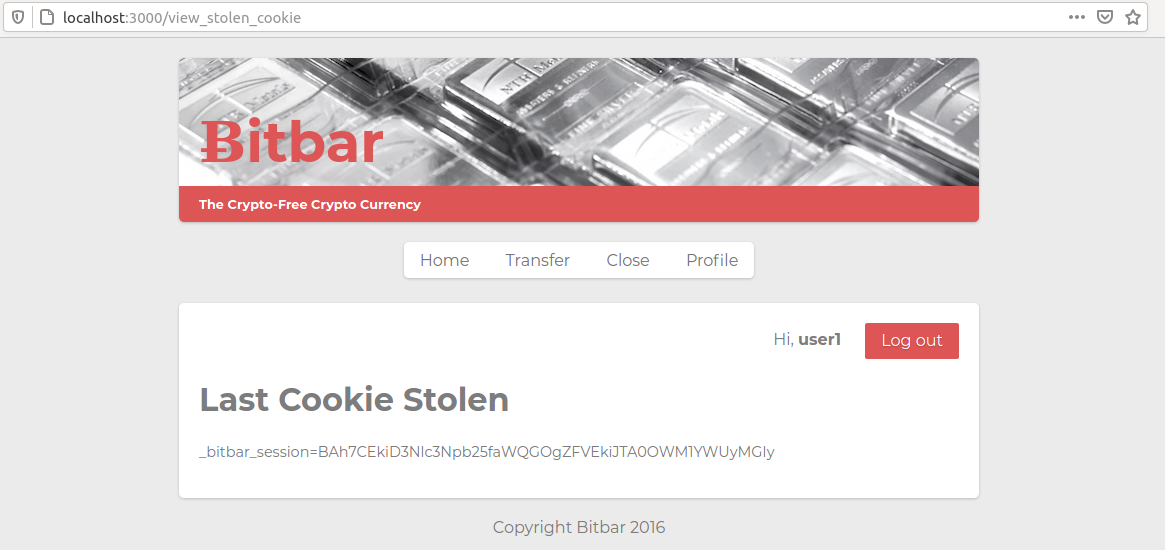


声明这是一段js脚本，将要发送的cookie以image对象向源地址发送请求的方式，向目的网址发送。

注入之前，被偷取的cookie查看：



注入之后，可以看到显示了被偷取的cookie的session了。



# Attack 2: Session hijacking with Cookies

## 【攻击要求】

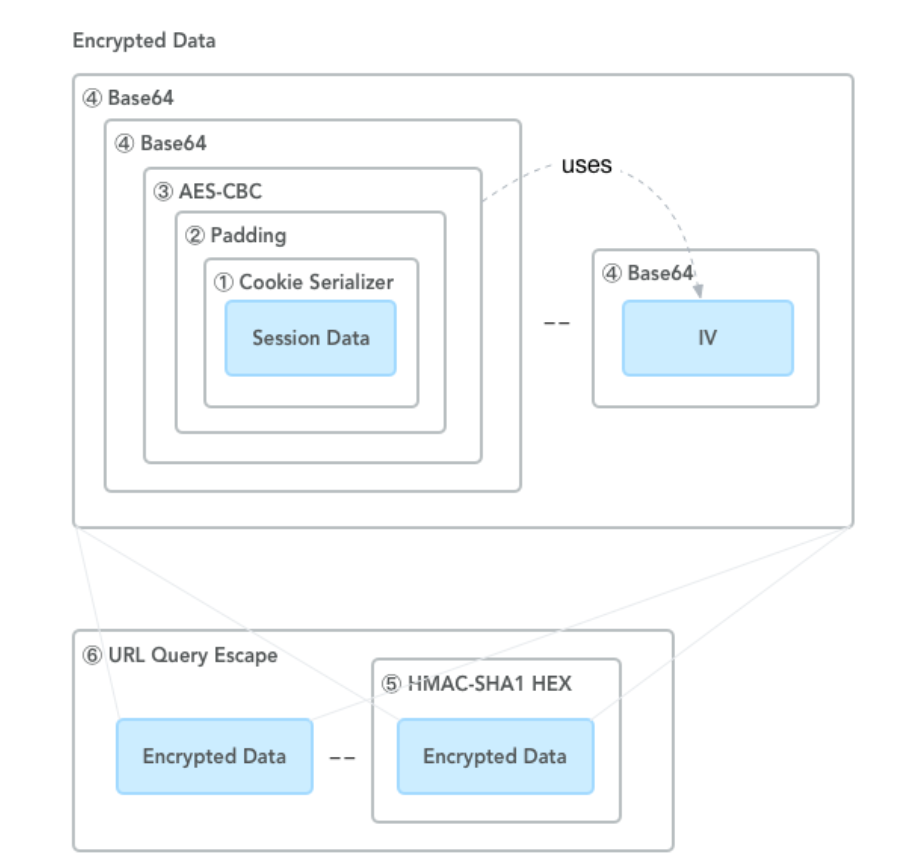
在本次试验中，你将会获得attacker的身份：用户名attacker，密码attacker。你的目的是伪装成用户user1登录系统。

你的答案应该是一个脚本。当这个脚本在JavaScript控制台中执行时，bitbar将误认为你是以user1。请将这个脚本写到a.sh中。

提示：网站是如何保存会话的？网站是如何验证用户当前是否登录？网站是如何验证cookie的真实性的？

## 【漏洞分析】

题目要求attack伪装成user1来登录，网站识别用户使用的是cookie，所以我们需要伪造user1的cookie。rails的cookie构造如下：（图源自<https://ruby-china.org/topics/34235>）

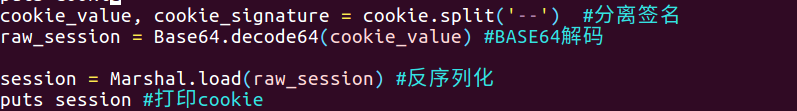


Session data的登录信息保存在warden.user.user.key，session = { "warden.user.user.key" => [[1],"secret"] }。

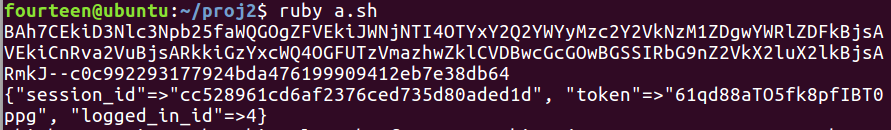
加密过程：序列化、填充、AES-CBC加密（用base64编码）、hmac-sha1签名、拼装加密内容和IV、签名 HMAC-SHA1、拼装签名。解密即将这六步倒过来进行。我们可以抓取attacker的cookie进行解密，然后改变登录信息，伪造成user1的cookie加密后发给系统。

## 【攻击原理及过程】

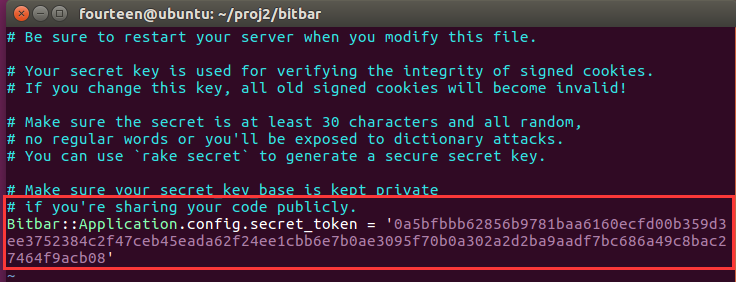
首先获得attacker的cookie。可以通过网页的查看元素获得，也可以直接用脚本抓取，然后进行解密，解密如下：



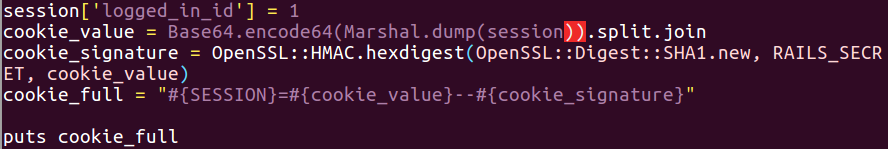
解密结果如下图，第一个是attacker的cookie，第二个是解密后的结果，可以看到需要修改的关键信息：token、logge\_in\_id.

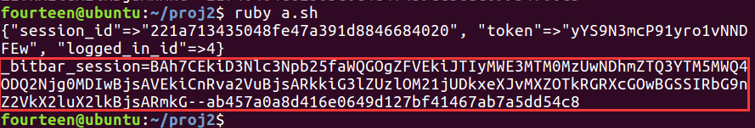


logged\_in\_id很简单，目前只有4个用户，user1应该logged\_in\_id=1.而token是服务器要验证的签名，伪造签名需要秘钥，可以在本地的源代码中获得，如下图。

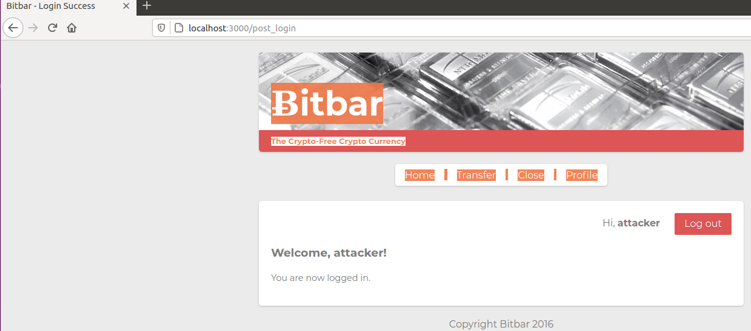


万事俱备，只需要开始伪造就行。按照上面加密过程，得到伪造的完整user1的cookie。代码如下：



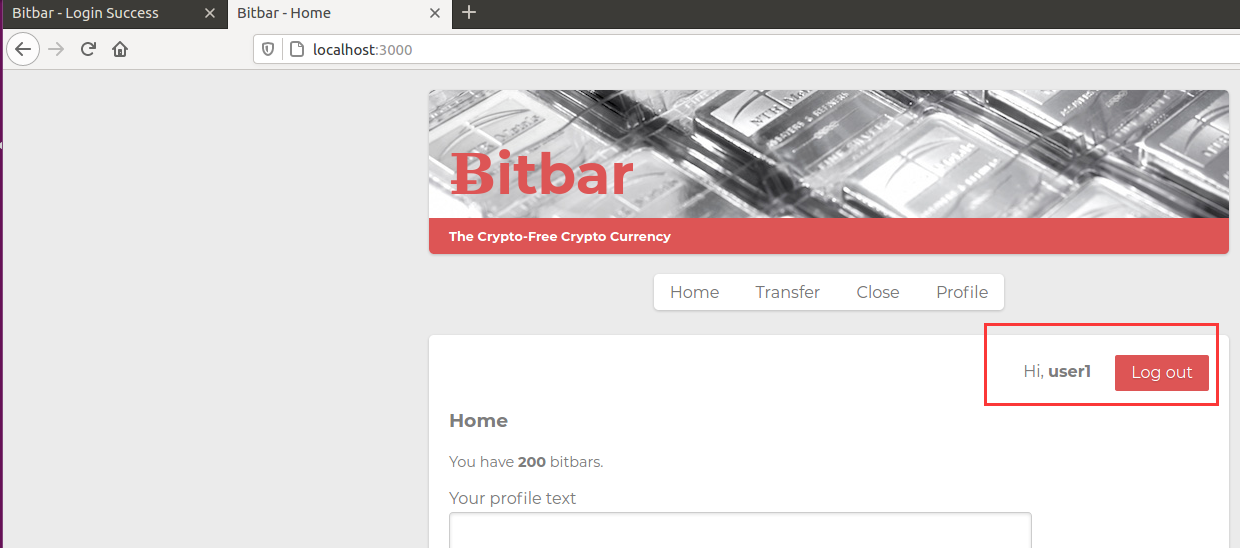


将得到的伪造cookie填入attacker登陆页面的docment.cookie





接着在新标签页打开localhost:3000，显示的是user1，说明伪造成功。



# Attack 3: Cross-site Request Forgery

## 【攻击要求】

你的答案是一个名字为b.html的html文件。

将提前使用user1的身份登录到bitbar，然后打开b.html。

打开b.html后，10个bitbar将从user1的账户转到attacker的账户，当转账结束时，页面重定向到www.baidu.com。

你可以在http://localhost:3000/view\_users 查看用户列表以及每个用户拥有的bitbar

在攻击的过程中，浏览器的网址中不能出现localhost:3000

提示：构造一个自动执行的html文件，通过XMLHttpRequest，构造发送的数据包，在登陆过User1的浏览器中执行。

## 【漏洞分析】

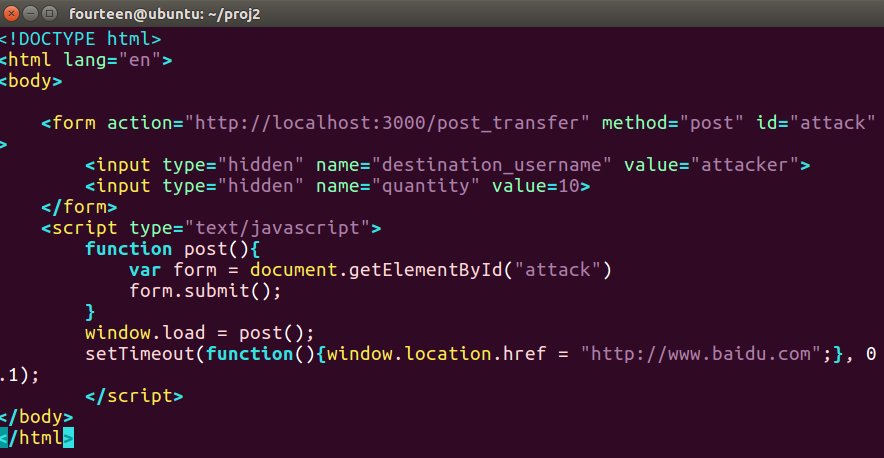
由于是先登录的user1，也就是user1的cookie信息被保留了，可以直接拿到user1的权限，所以html文件需要实现的是转账以及重定位到百度。

首先先抓取一个转账的数据包分析，如下图，可以发现这只是一个表单，提交的目标为”post\_transfer”，方式为”post”。所以我们要做的就是做一个可以自动提交表单的网页。



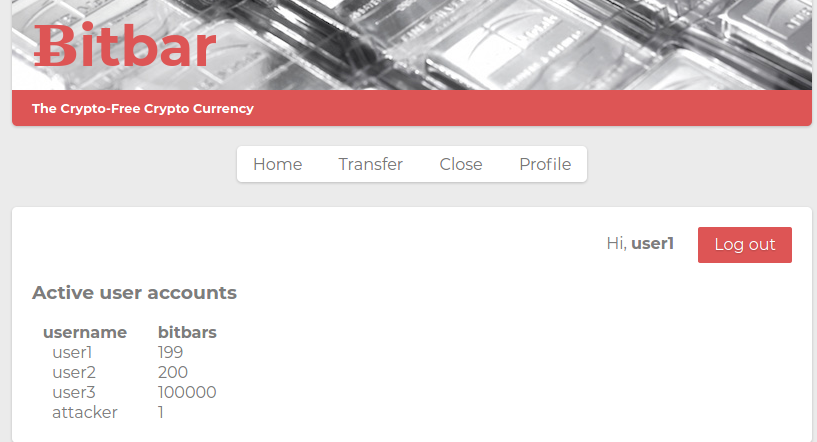
## 【攻击原理及过程】

b.html的内容如下。提交的目标、方式以及关键字都与上面抓到的数据包相同，将表单设置为隐藏的，每次提交表单都是给attacker账户转账10元。

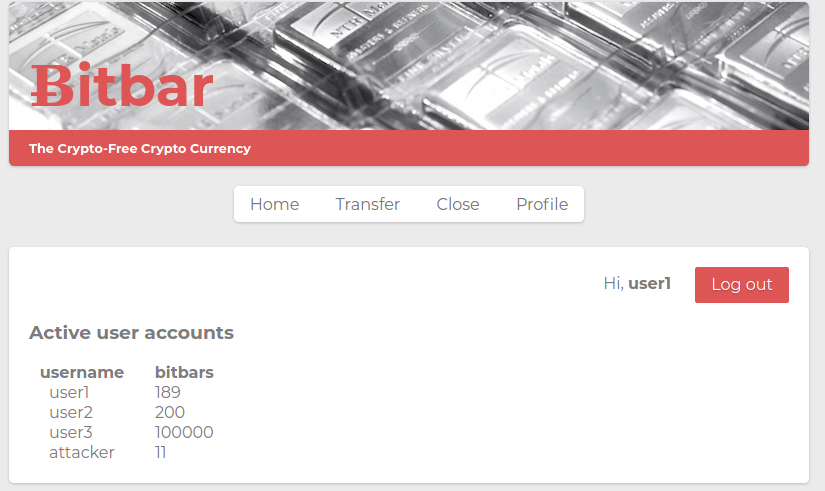


要实现自动转账，需要设置当b.html被打开即调用函数提交表单，可以使用window.load。最后添加重定向到baidu.com的代码，延时0.1s转到百度。

转账之前：



点击b.html之后：





# Attack 4: Cross-site request forgery with user assistance

## 【攻击要求】

你的答案是一个或者两个html页面，命名为bp.html，bp2.html(可选)

在打开bp.html前，系统中已经登录了user1

接下来将在bp.html页面进行交互，因此bp.html的设置要合理。也就是说，如果在页面上有一个表格或者有一个按钮，并且在页面上有一些提示要求用户进行一些操作，引导用户依照这些提示执行。

在用户与bp.html页面进行交互后，10 bitbars将会从user1账户转到attacker的账户。当这个转账操作执行完成后，页面将重定向到www.baidu.com

你的攻击必须要在与用户互动的前提下执行（与Attack 3不同）。特别的要注意的是，你的攻击要针对的网址是http://localhost:3000/super\_secure\_transfer或者http://localhost:3000/super\_secure\_post\_transfer。这两个网址做了一些CSRF攻击的防护。在攻击的过程中，你不能直接与http://localhost:3000/transfer或者http://localhost:3000/post\_transfer进行交互。

在你的攻击过程中，需要隐藏你的页面正从http://localhost:3000上下载内容的事实。

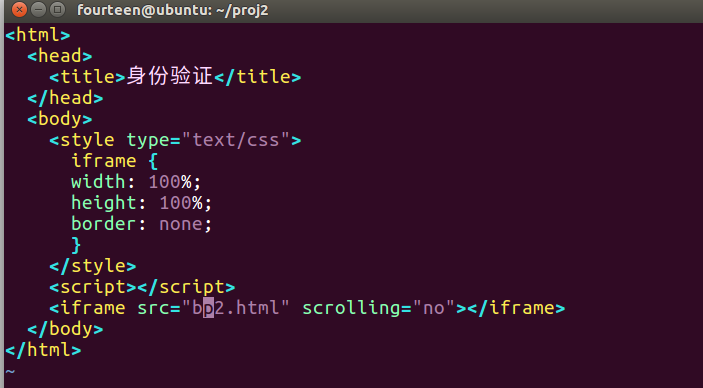
## 【漏洞分析】

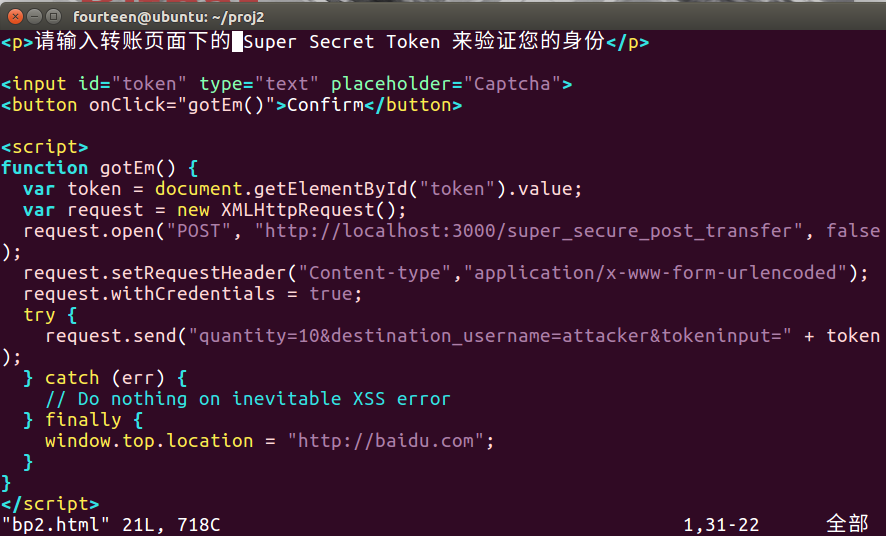
attack4要达到的目的与attack3相同，但是针对的网页不同。同样抓个数据包来分析，可以发现新网址比原来的多了一个随机的token，也就是新网址上的验证码。我们可以让用户与网页交互时，欺骗用户输入验证码，这样我们就能获取到这个token。我们可以给一些提示引导用户。



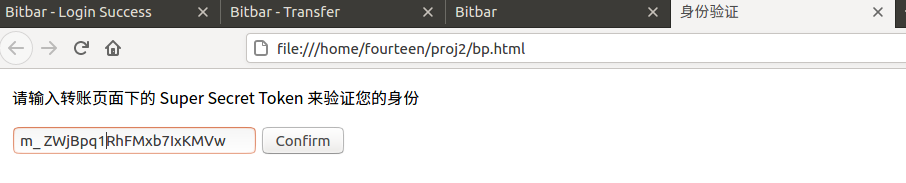
## 【攻击原理及过程】

设计一个网页，显示引导用户的提示以及交互的输入框和按钮，当用户输入后，接下来要完成的事与attack3一样，但是多一个token。

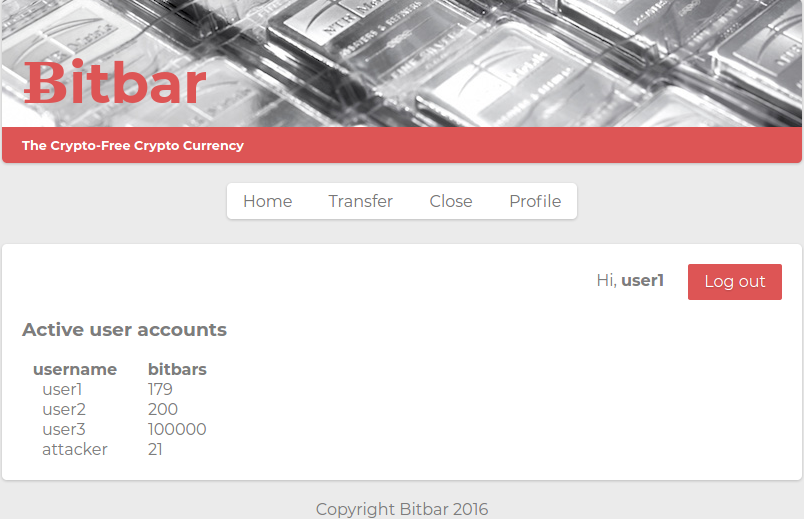




点击bp.html，跳转到如下界面，输入token后，跳转到百度界面，查看个用户账面，发现转账成功。







# Attack 5: Little Bobby Tables (aka SQL Injection)

## 【攻击要求】

你的答案是一个恶意的用户名。这个恶意的用户名允许你删除一个你不具有访问权限账户。

评分员将使用你提供的恶意用户名新建一个账户。并在“close”页面上确认删除该账户

作为结果，新建的账户以及user3的账户将会被删除。其他的账户不变

你可以在http://localhost:3000/view\_users页面上查看所用的用户

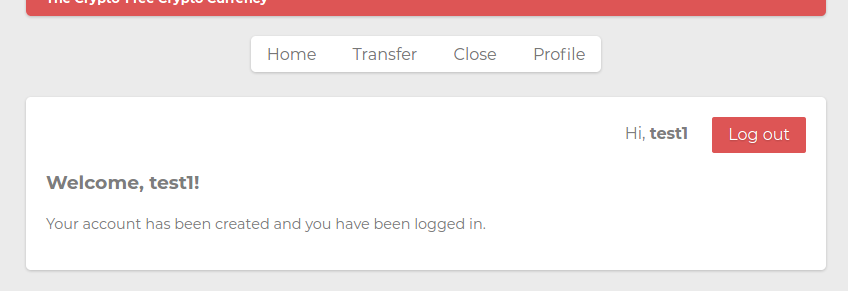
如果数据库在测试攻击的过程中被破坏了，你可以停止Rails然后使用rake db:reset命令使数据库复原。

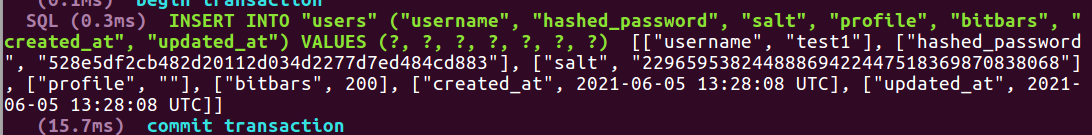
将你的最终答案写在c.txt中

提示：查看服务器删除用户时的操作。SQL注入；WHERE子句

## 【漏洞分析】

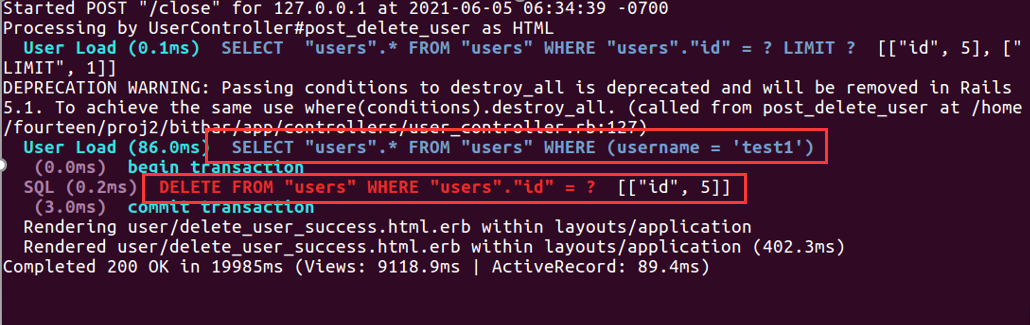
先创建一个新用户test1（如下图），看看是如何创建的。





可以看到如上图，终端进行了数据库的相关操作，插入了新用户的信息。

当删除用户时，终端的操作如下。



删除时，先查询test1的id，然后根据查询到的id进行删除。

插入时，用户名直接作为users.username作为信息，所以在之后的id查询语句中，用户名直接放在了单引号中。所以我们可以使新用户名与查询id语句一起构成新的查询语句，使得查询到的id为user3的id和新用户的id，这样就能达到删除新用户和user3的目的。

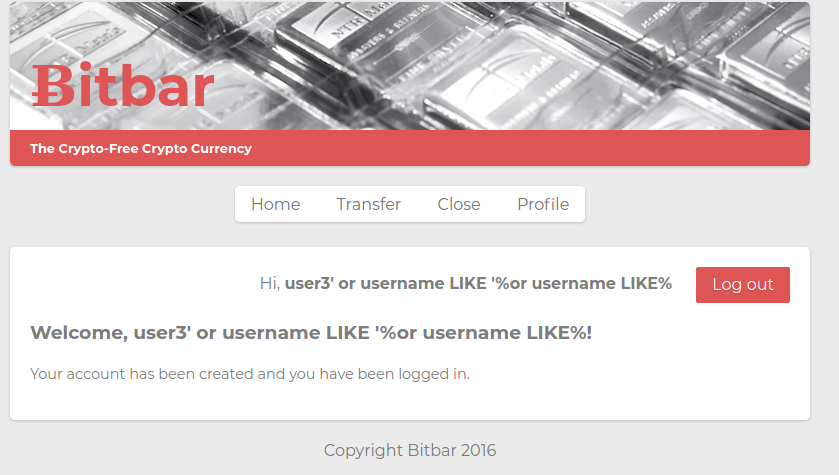
## 【攻击原理及过程】

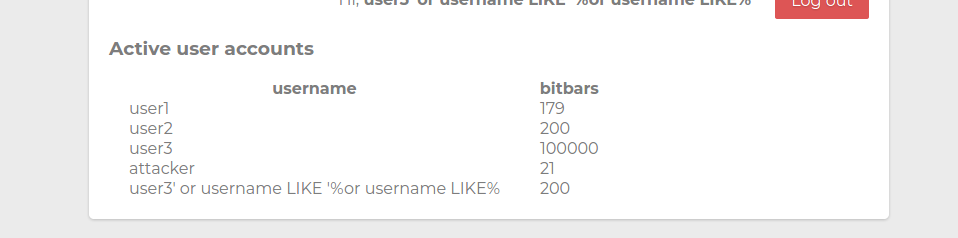
将新用户名设置为：user3’ or username LIKE ‘%or username LIKE%

则新的查询id语句（标红的符号为语句中原有的）：

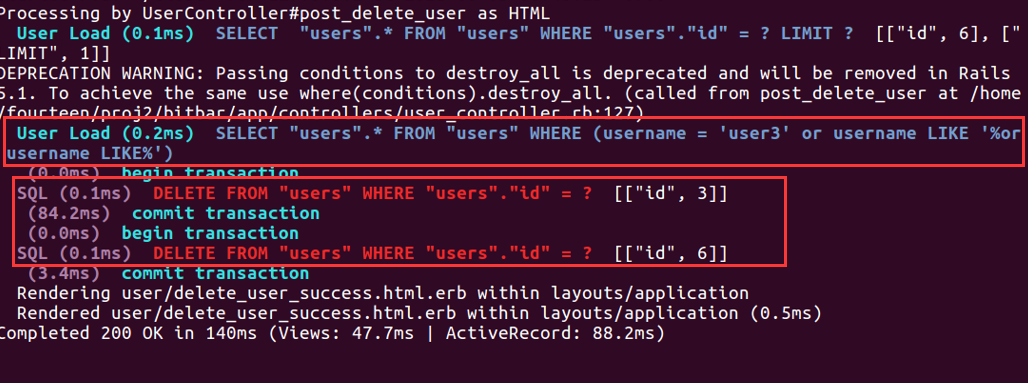
SELECT “users” .\* FROM “users” WHERE （username = ‘user3’ or username LIKE ‘%or username LIKE%’）

账号创建后如下：

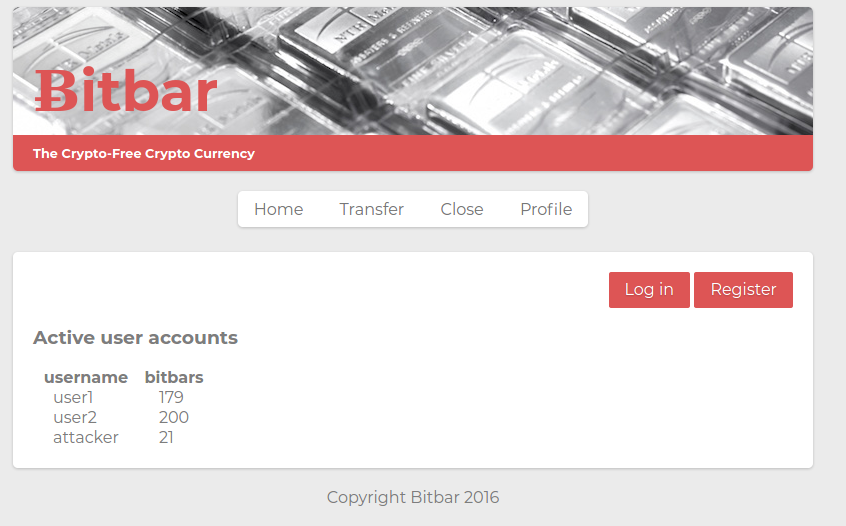




删除用户时，终端的语句如下，和预想中的一样。



可以看到，删除成功，user3和新用户都被删除了。



# Attack 6: Profile Worm

## 【攻击要求】

你的答案是一个用户的profile（简况）。当其他用户阅读这个profile时，1个bitbar将会从当前账户转到attacker的账户，并且将当前用户的profile修改成该profile。因此，如果attacker将他的profile修改成你的答案，以下情况会发生：

如果user1浏览了attacker的profile，那么1 bitbar将从user1的账户转到attacker的账户，user1的profile修改成你答案中的profile

之后，如果user2浏览了user1的profile，那么1 bitbar将从user2的账户转到attacker的账户，user2的profile也被替换成你答案中profile

因此，你的profile worm将会很快扩散到全部的用户账户中

将你的恶意的profile写在d.txt中。

评分过程：将你提供的恶意profile复制到attacker的profile上。然后，使用多个账户浏览attacker的profile。检查是否正常进行转账以及profile的复制

转账和profile复制的过程应该控制在15s之内。

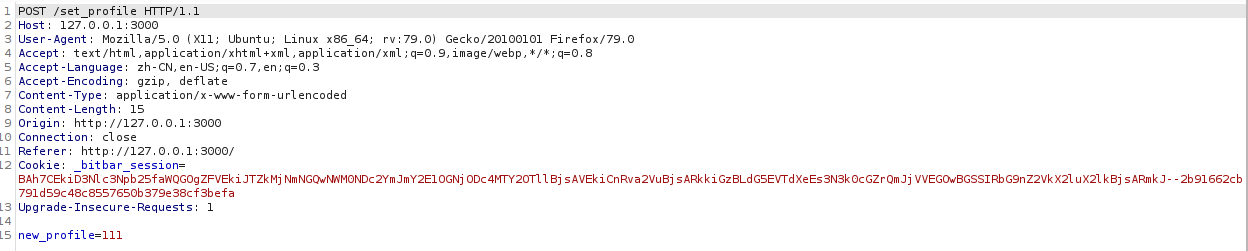
在转账和profile的复制过程中，浏览器的地址栏需要始终停留在http://localhost:3000/profile?username=x ，其中x是profile被浏览的用户名。

提示：MySpace vulnerability，存储型XSS

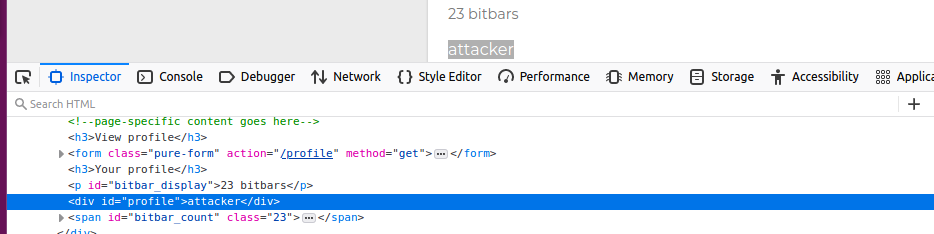
## 【漏洞分析】

这里有两个行为：1、转账。2、传播。每一个浏览的用户都会被“感染”，被感染的用户会继续感染下一个用户。转账的实现和attack3、4是一样的，这里不进行赘述。主要是讲述如何传播。

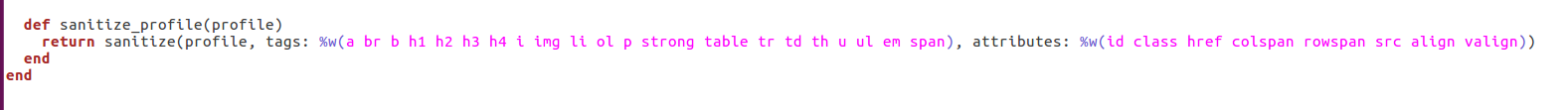
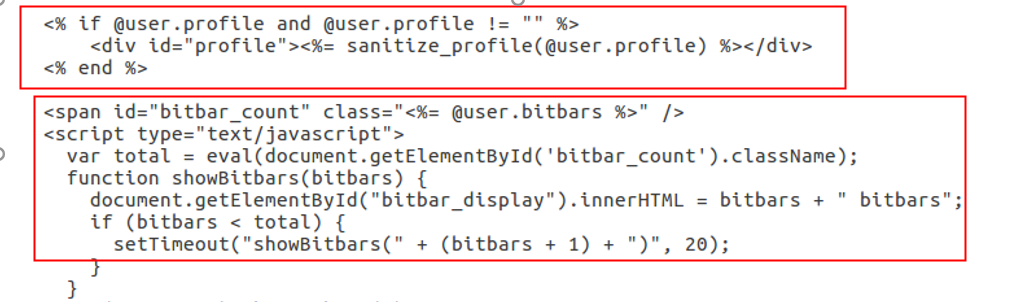
接着用户的profile被修改。登陆attacker修改profile进行抓包分析，数据包如下。可以发现要修改一个用户的profile，即向/set\_profile发送数据。



浏览profile时，查看对应源代码，发现用户的profile被<div id= “profile”>…… </div>标签包围。

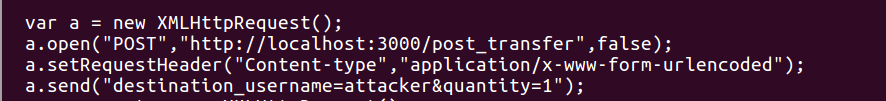


查看这一段的源代码，发现调用了sanitize\_profile，这个函数在application\_helperrb中有定义，通过 tags 和 attributes设定了白名单，设置了 attributes 参数，就会跳过一些属性过滤，我们可以利用这一点，来进行profile的修改。我们可以设置一个标签“wr”，将"wr"标签中的内容通过.outerHTML方法构造到提交的请求中。



## 【攻击原理及过程】

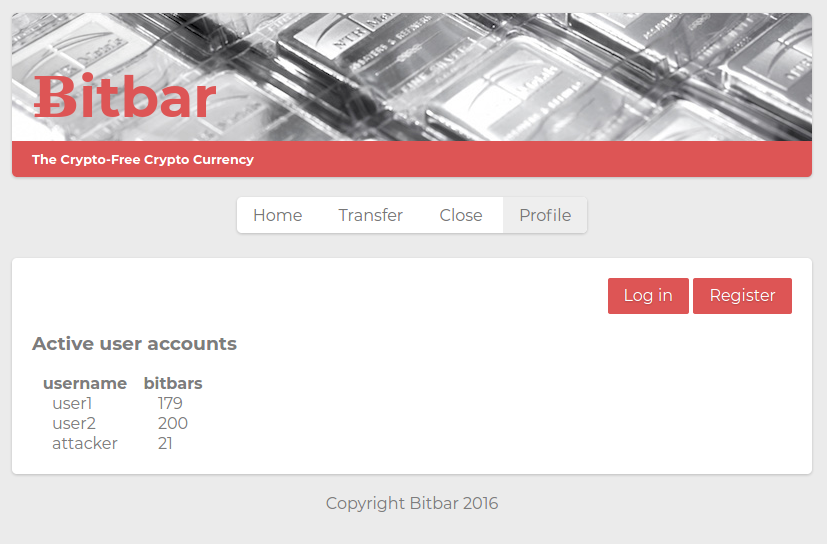
按照攻击顺序，先是浏览用户被转账。转账的代码如下：



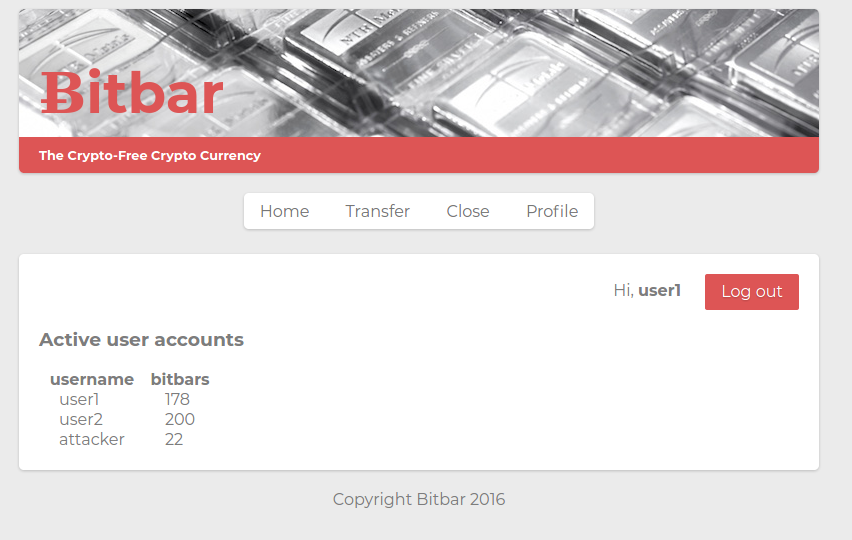
接着进行profile的复制。如下图：



在user1浏览attacker之前，各用户的余额如下：



登陆attacker，将profile更新。接着退出登陆user1，user1浏览attacker后，余额变更：



user2浏览user1后，余额变更：

