网络空间安全实验基础实验四实验报告

基本信息：

完成人姓名：黄浩　　学号：57119134　　完成日期：2021 年 7 月 15 日

实验内容：

环境配置：

搭建实验环境(启动各个 Docker)：

```
[07/15/21]seed@VM:~/.../Labsetup$ docker-compose build
Building elgg
Step 1/10 : FROM handsonsecurity/seed-elgg:original
 ---> e7f441caa931
Step 2/10 : ARG WWWDir=/var/www/elgg
 ---> Using cache
 ---> a06950e00398
Step 3/10 : COPY elgg/settings.php $WWWDir/elgg-config/settings.php
 ---> Using cache
 ---> 16930f5ee193
Step 4/10 : COPY elgg/Csrf.php     $WWWDir/vendor/elgg/elgg/engine/classes/Elgg/
Security/Csrf.php
 ---> Using cache
 ---> 9cae3debb47b
Step 5/10 : COPY elgg/ajax.js      $WWWDir/vendor/elgg/elgg/views/default/core/j
s/
 ---> Using cache
 ---> f706efd3fa79
Step 6/10 : COPY apache_elgg.conf /etc/apache2/sites-available/
 ---> Using cache
 ---> cdcb32a6353b
Step 7/10 : RUN  a2ensite apache_elgg.conf
```

```
[07/15/21]seed@VM:~/.../Labsetup$ docker-compose up
WARNING: Found orphan containers (server-1-10.9.0.5, server-3-10.9.0.7, server-2
-10.9.0.6, server-4-10.9.0.8) for this project. If you removed or renamed this s
ervice in your compose file, you can run this command with the --remove-orphans
flag to clean it up.
Creating elgg-10.9.0.5       ... done
Creating attacker-10.9.0.105 ... done
Creating mysql-10.9.0.6      ... done
Attaching to elgg-10.9.0.5, mysql-10.9.0.6, attacker-10.9.0.105
mysql-10.9.0.6 | 2021-07-15 06:51:45+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2021-07-15 06:51:45+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2021-07-15 06:51:45+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2021-07-15 06:51:46+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2021-07-15T06:51:46.134558Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 43
mysql-10.9.0.6 | 2021-07-15T06:51:46.147762Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
attacker-10.9.0.105 |  * Starting Apache httpd web server apache2
 *
```

手动指定 DNS：

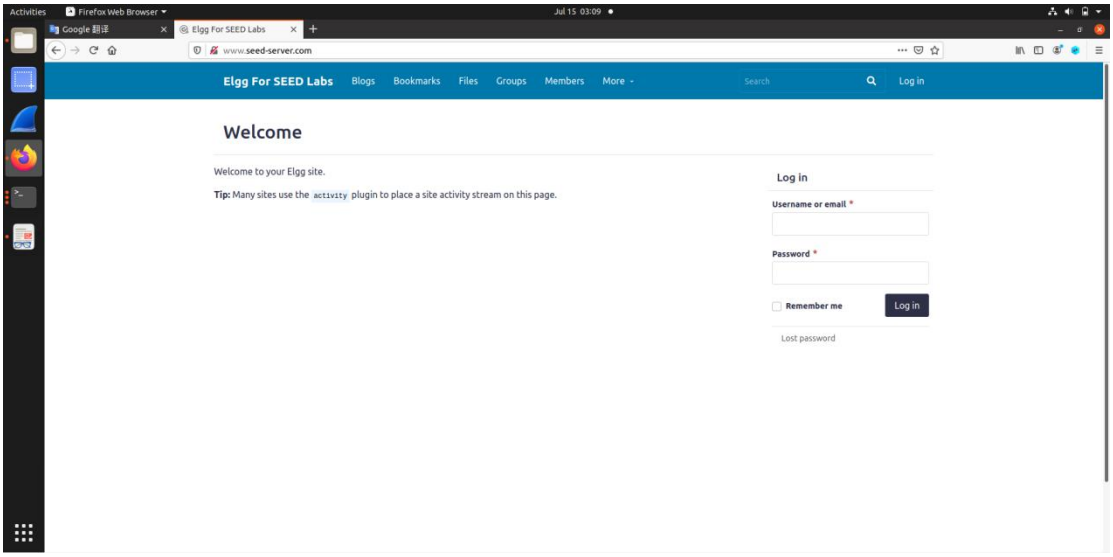
```
# For CSRF Lab
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32.com
10.9.0.105      www.attacker32.com
```

清洁 Mysql 数据库(洁癖行为)：

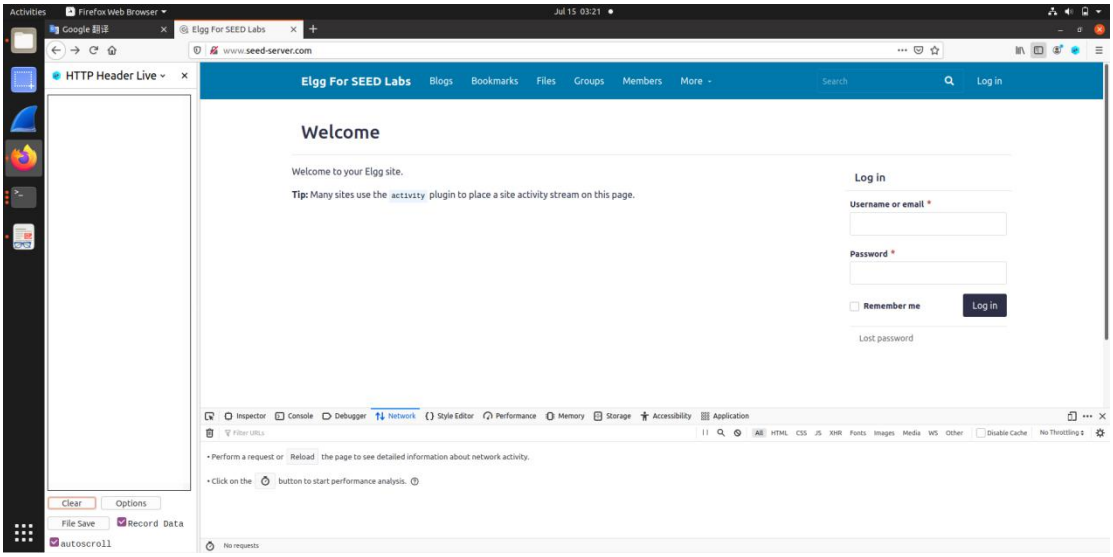
```
[07/15/21]seed@VM:~$ sudo rm -rf mysql_data
[07/15/21]seed@VM:~$
```

TASK 1：观察 Http 请求

访问我们搭建的 seed-server.com：

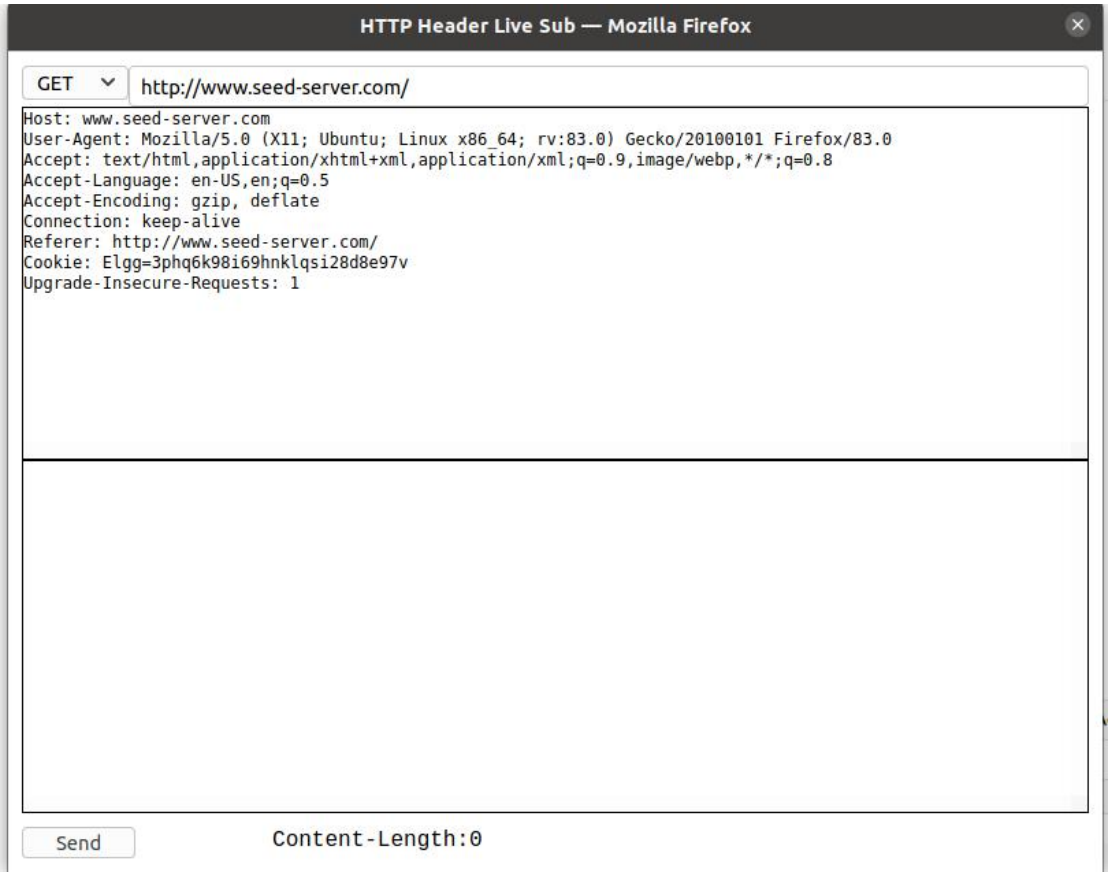


打开 Http Header Live:

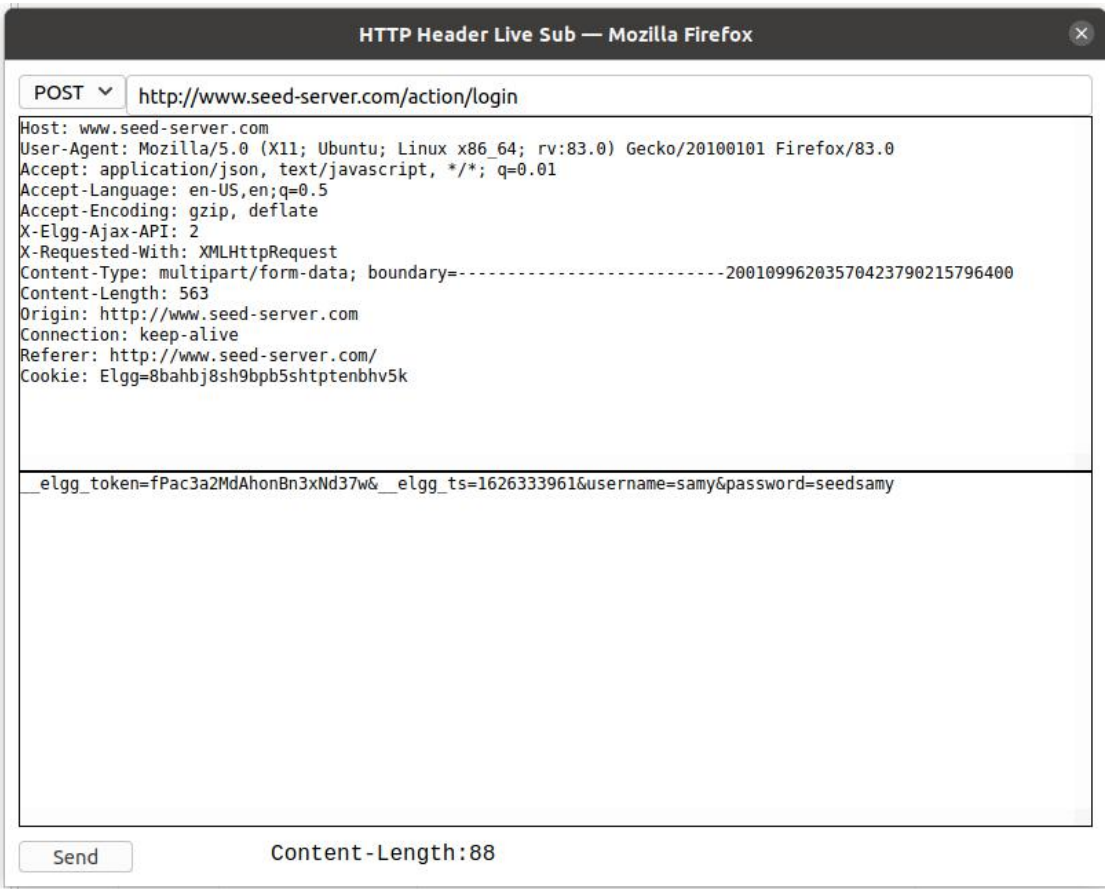


捕获网站的 get 和 post 请求(登陆一个用户 Samy)：

get 请求：

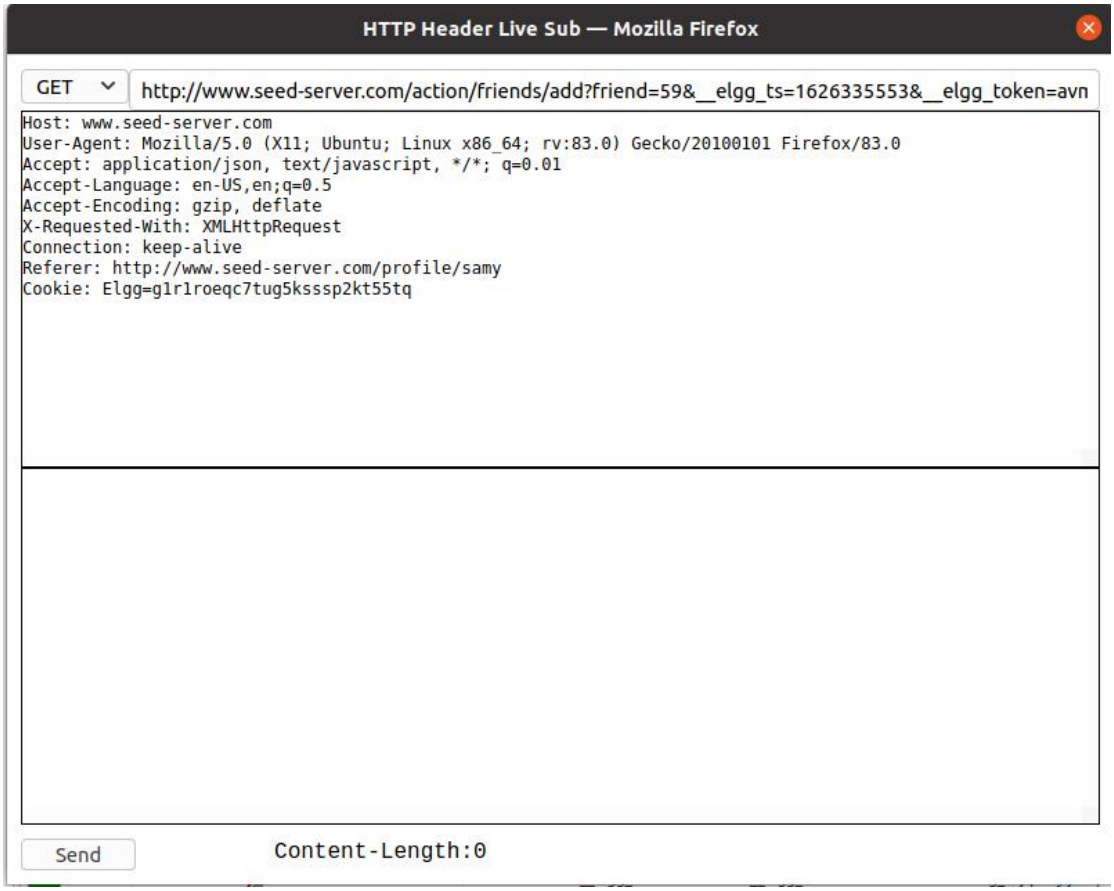


post 请求：



TASK 2：运用 get 请求进行 CSRF 攻击：

利用 Charlie 添加 Samy 为好友从而获取添加 Samy 好友的 get 请求报文：

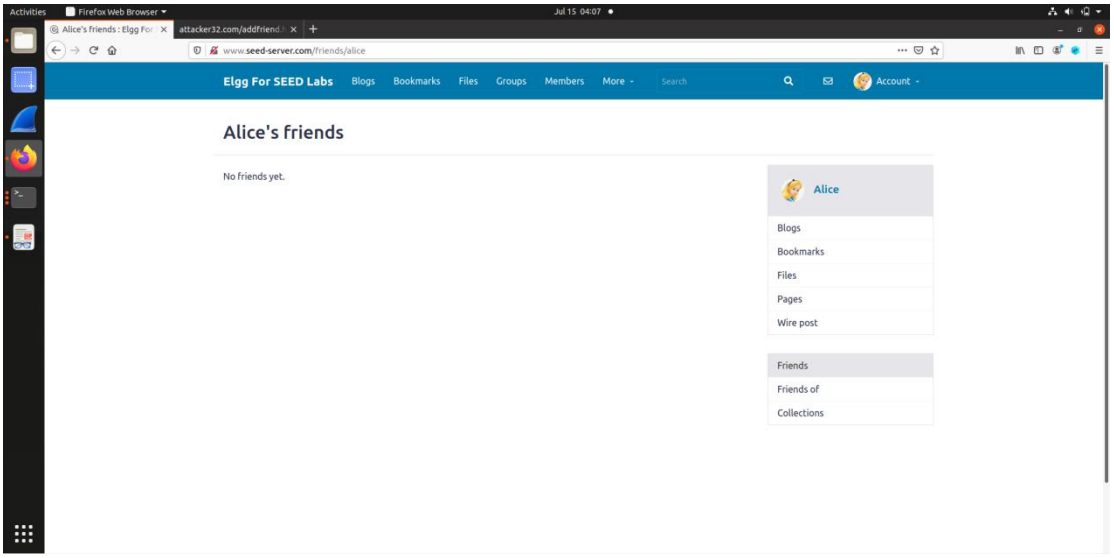


搭建 Samy 的攻击网站：

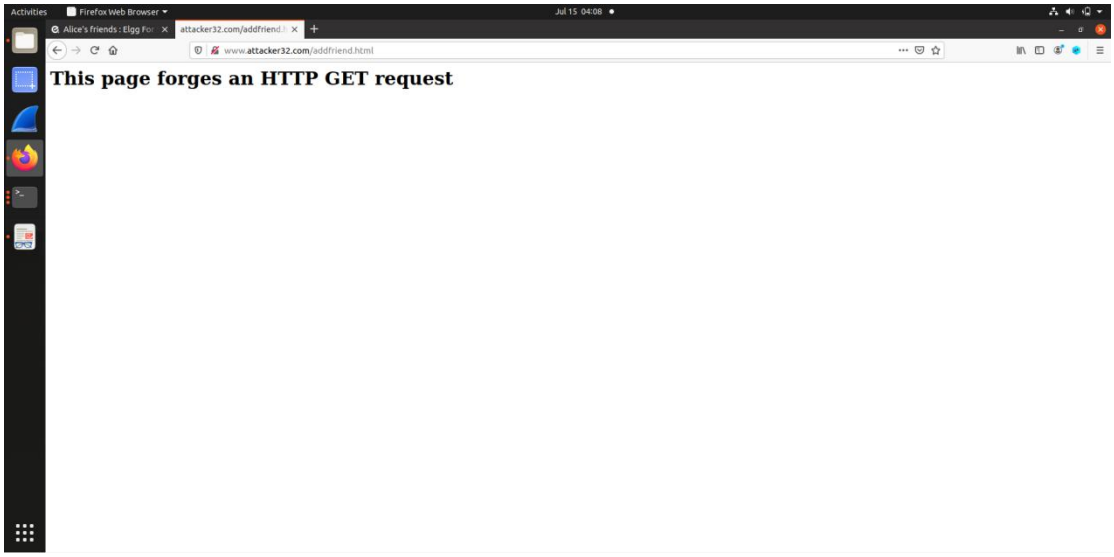
```
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59" alt="image" width="1" height="1" />
</body>
</html>
```

然后我们登录 Alice 的账户模拟 Alice 浏览 Samy 的攻击网站：

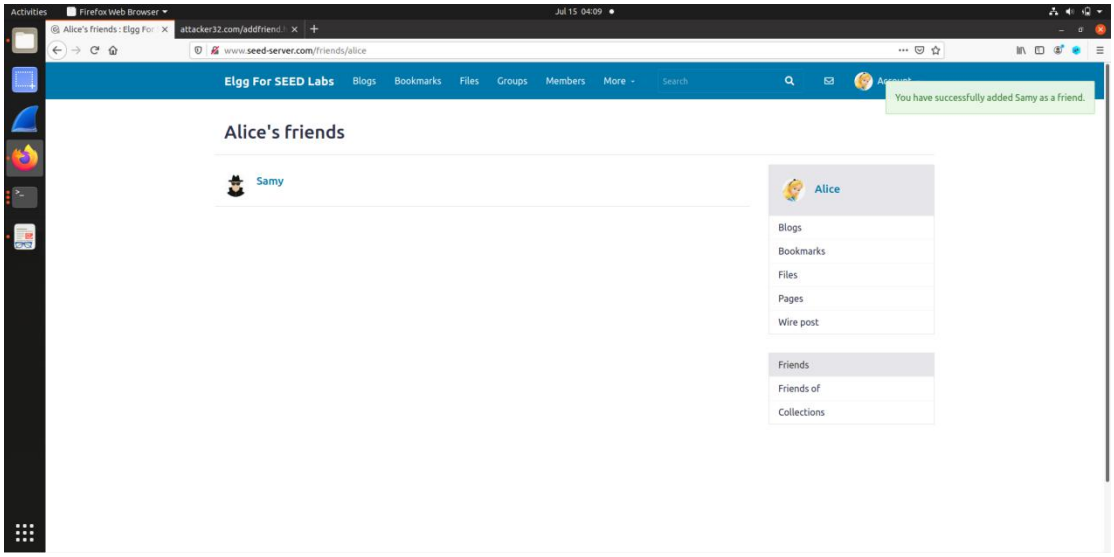
Alice 最开始没有好友：



然后 Alice 浏览 Samy 发送给她的网址：www.attacker32.com/addfriend.html



然后攻击成功，Alice 添加了 Smay 的好友！

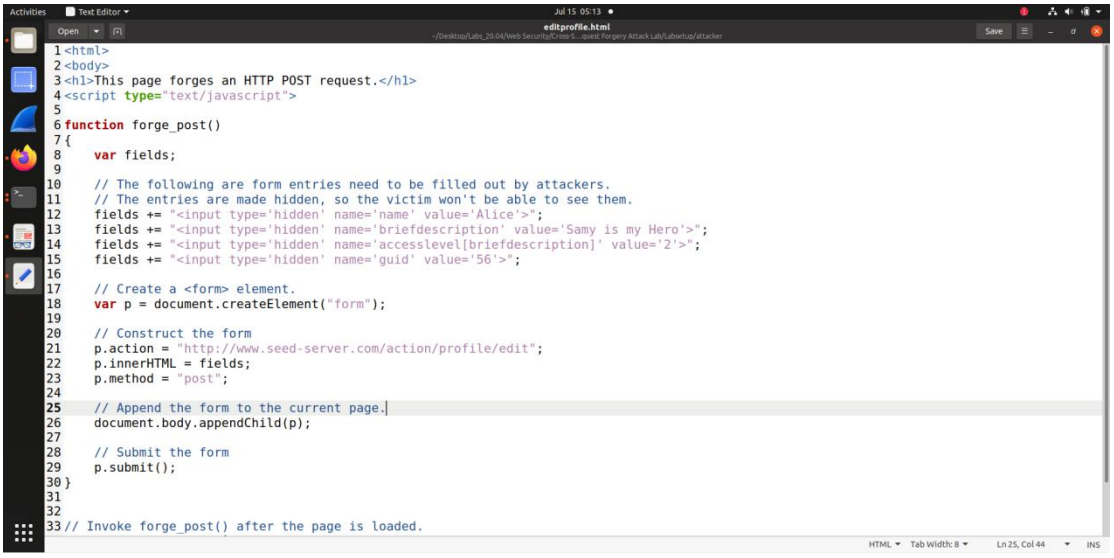


TASK 3:运用 post 请求进行 CSRF 攻击：

Samy 通过修改自己的简介来获取 post 报文：

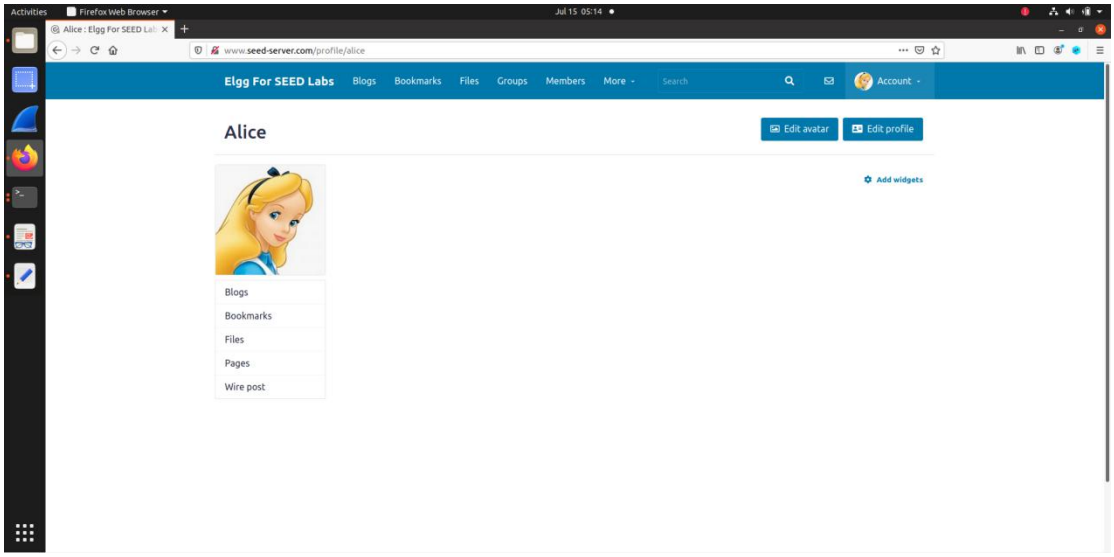
同时我们可以通过访问 Alice 的主页，然后查看网页源码获取 Alice 的 guid(56)：

根据上述信息，使用 JavaScript 编写攻击页面：

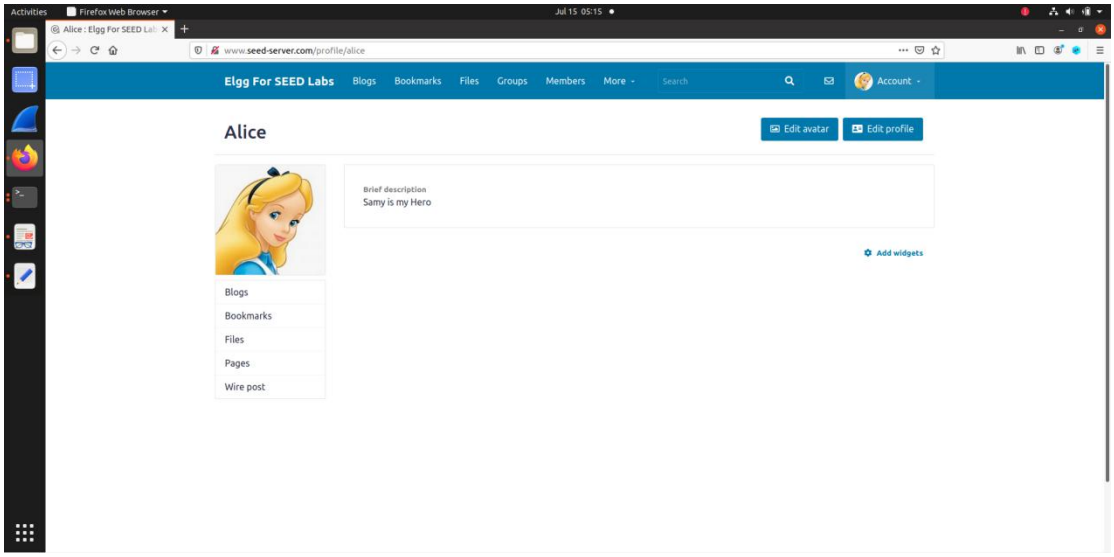


现在登录 Alice 的账号：

浏览 Samy 的网站前：



浏览 Samy 的网站 www.attacker32.com/editprofile.html 后：



攻击成功！Alice 的个性描述已经变成了 Samy is my Hero。

Question 1:

Body 可以用自己的账号进入 Alice 的主页，然后查看页面源代码，在其中搜索关键字 uid，便可以找到 Alice 的 uid。

Question 2:

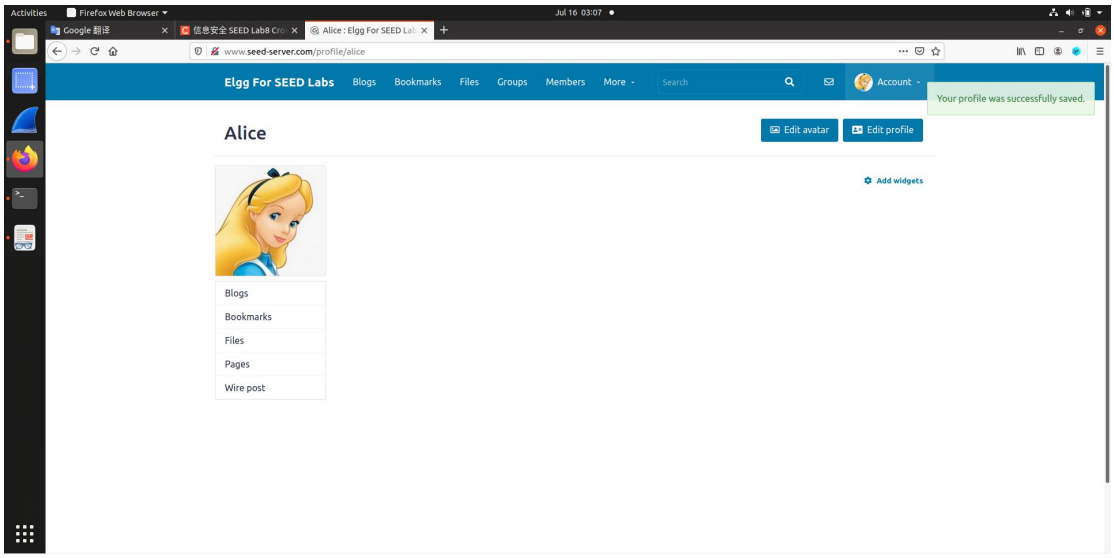
由于此网站 uid 可通过访问用户主页源代码查出，所以我们可以通过暴力手段在我们的代码中插入所有用户的 uid(即枚举所有用户 id)，这样无论是谁访问我们的钓鱼网站，他都会被攻击成功。

TASK 4: 开启 Elgg 的对策

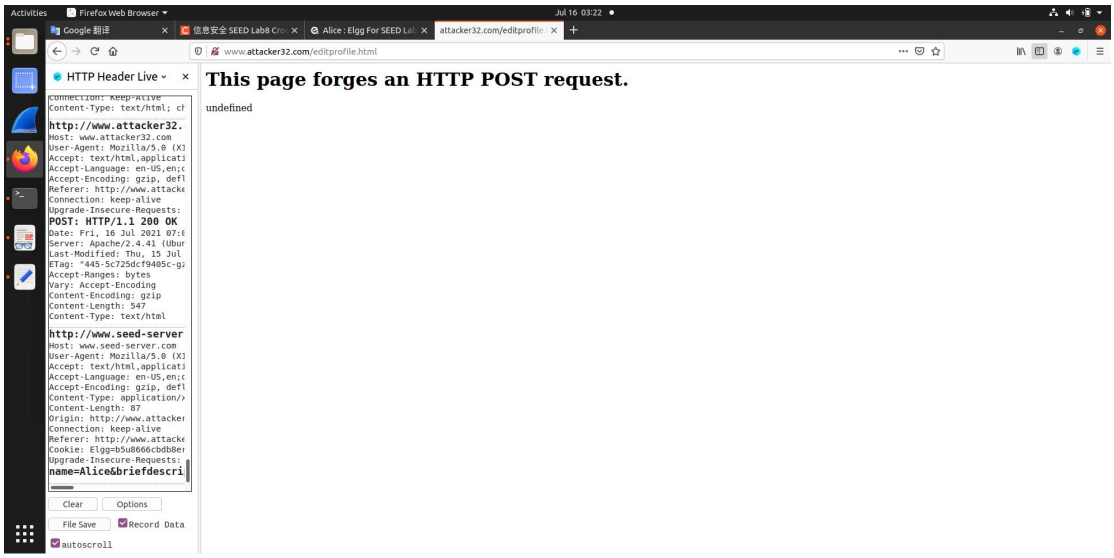
在 image_www/elgg/Csrf.php 中开启防御措施(删除 return 语句)，并重新构建服务器：

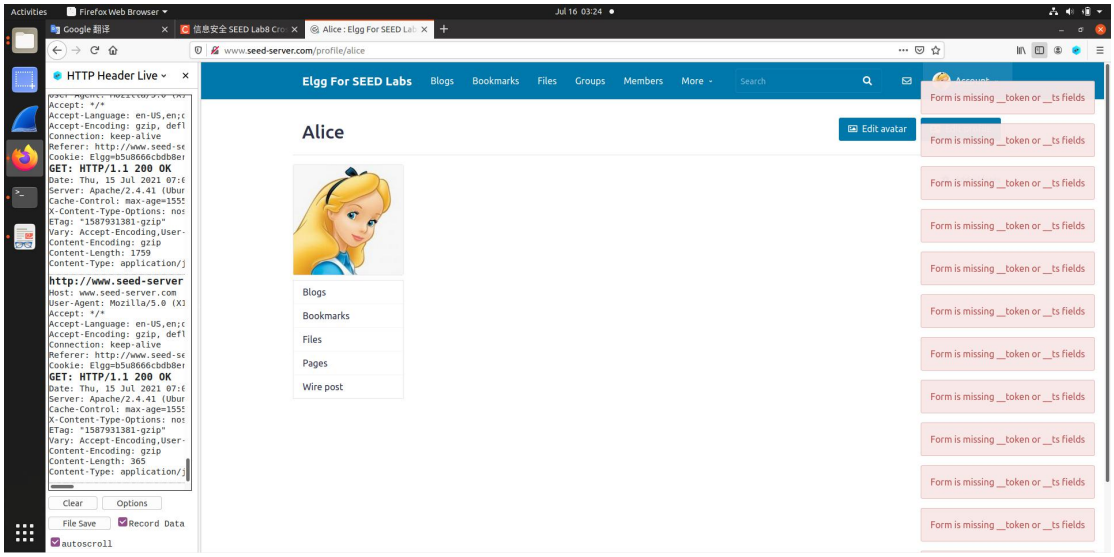
```
68          public function validate(Request $request) {
69                  // Added for SEED Labs (disabling the CSRF countermeasure)
70
71                  $token = $request->getParam('__elgg_token');
72                  $ts = $request->getParam('__elgg_ts');
```

然后我们打开 Alice 的账号主页，将之前的攻击结果删除：

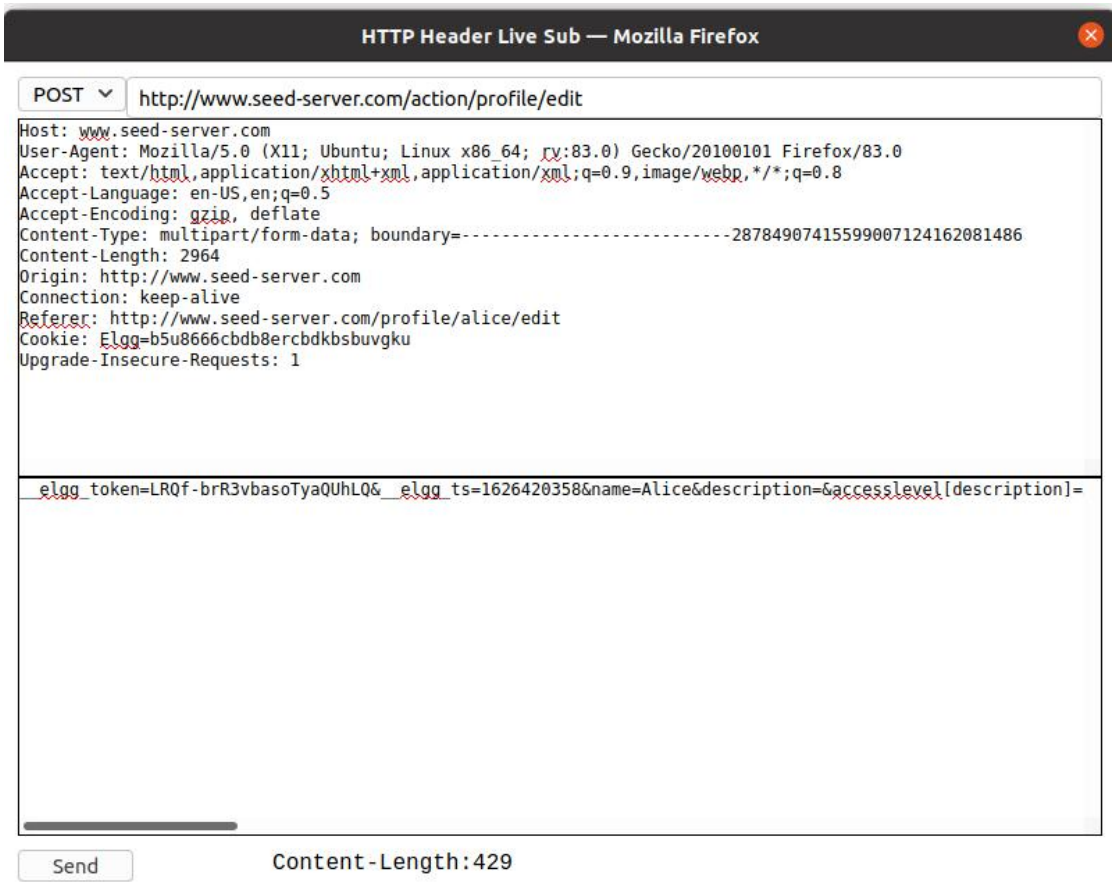


然后再次使用 Alice 浏览我们的钓鱼网站：





左侧一直在重复进行 post 请求(死循环)，所以我们攻击失败。
我们抓取的任一请求如下：



下栏中的__elgg_token 和 __elgg_ts 便是此网页的令牌。

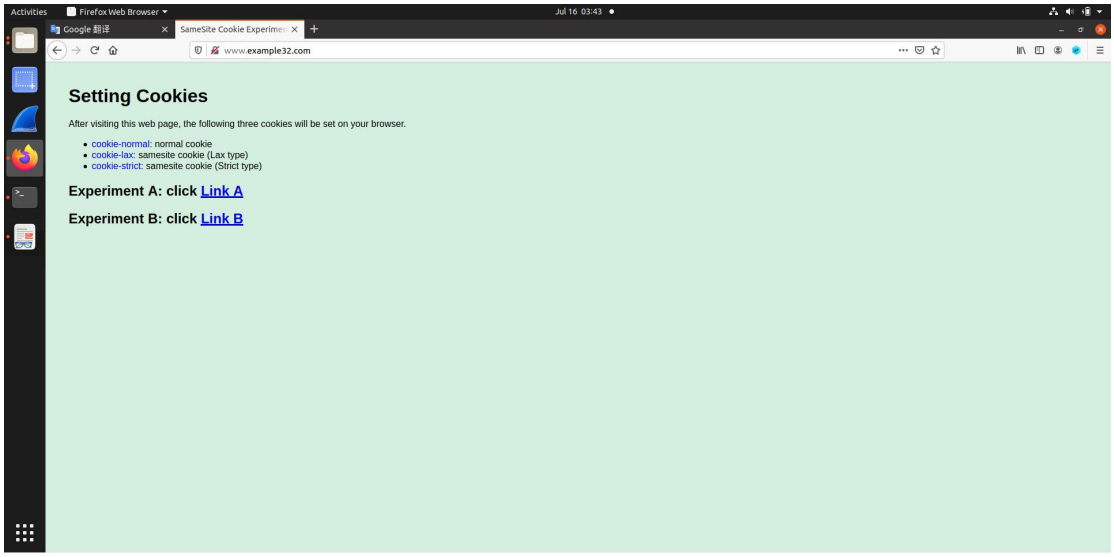
为什么攻击者不能将这些令牌信息加入自己的 post 报文？

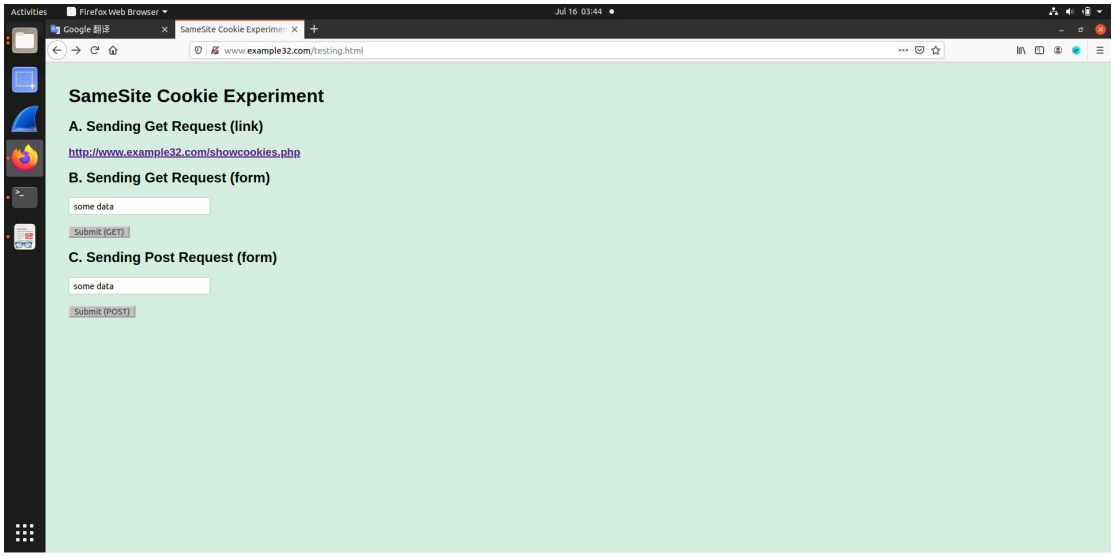
因为，这些令牌信息是通过数据库中的网站秘密值、时间戳、用户会话 id、随机生成的会话字符串等信息经过 MD5 加密后产生的。攻击者除非能够直接攻入网站数据库

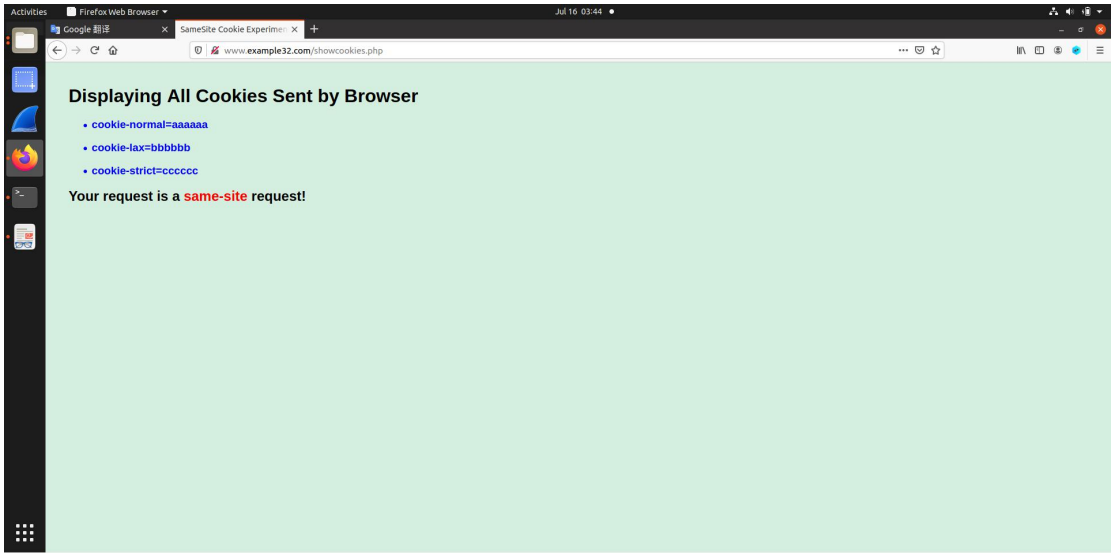
查询这些信息，否则只能通过彩虹桥等暴力手段堆凑此哈希值，所以攻击者的攻击行为成本大大上升，从而达到防范 CSRF 攻击的效果。

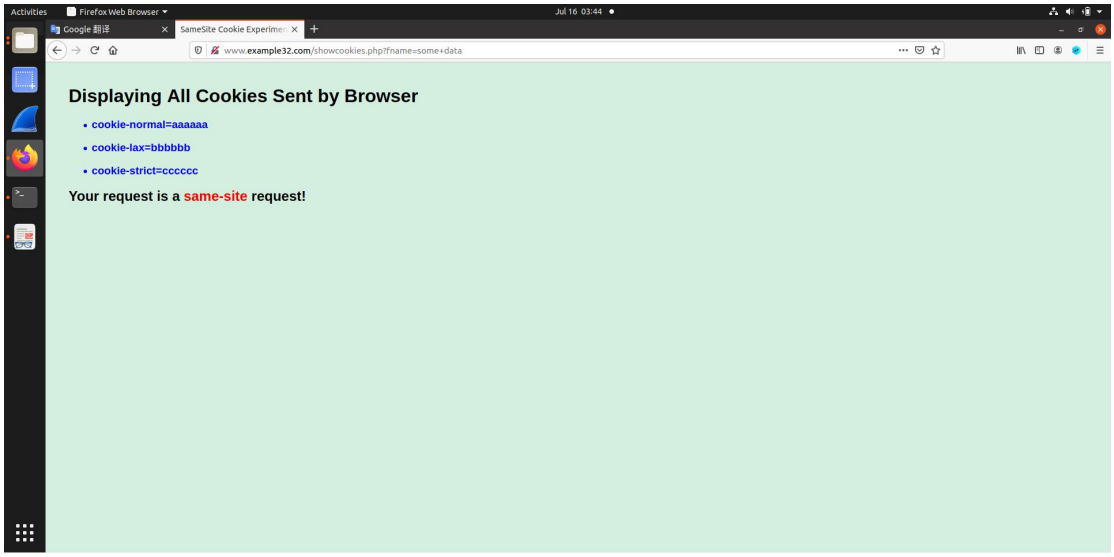
TASK 5：体验同站 cookie 方法：

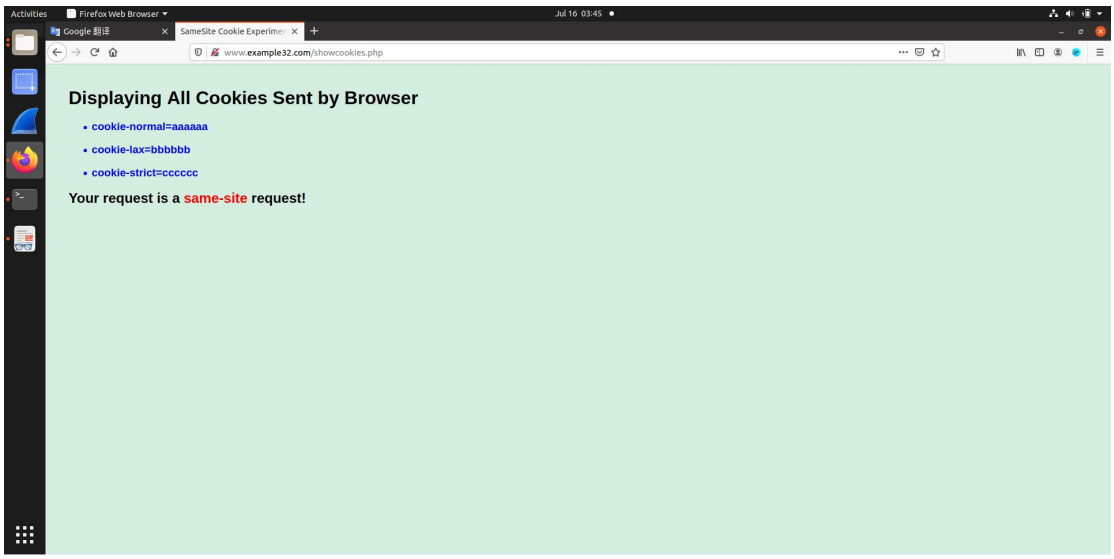
访问 www.example32.com,并依次点击各类情况记录数据:

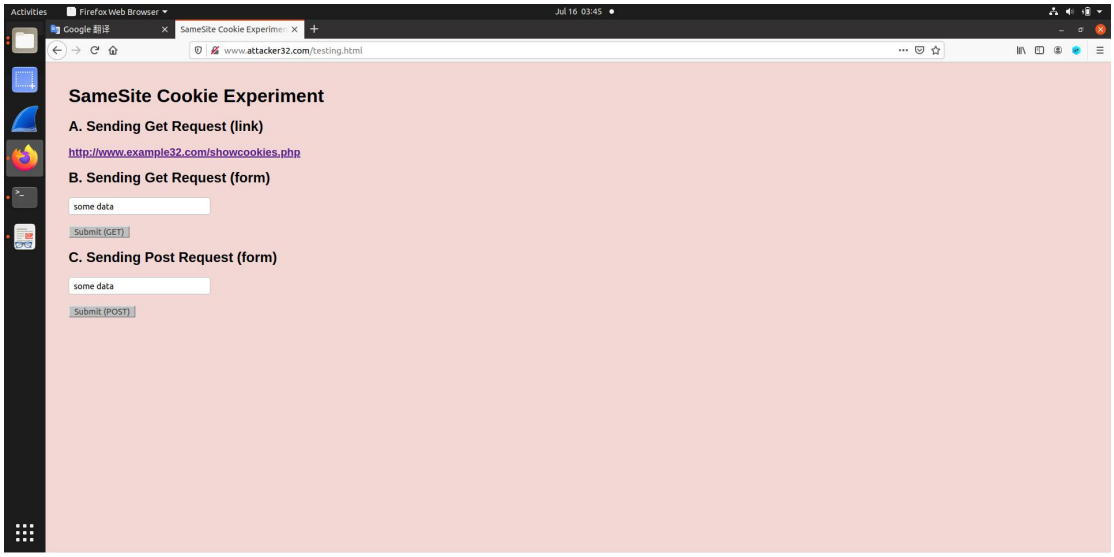


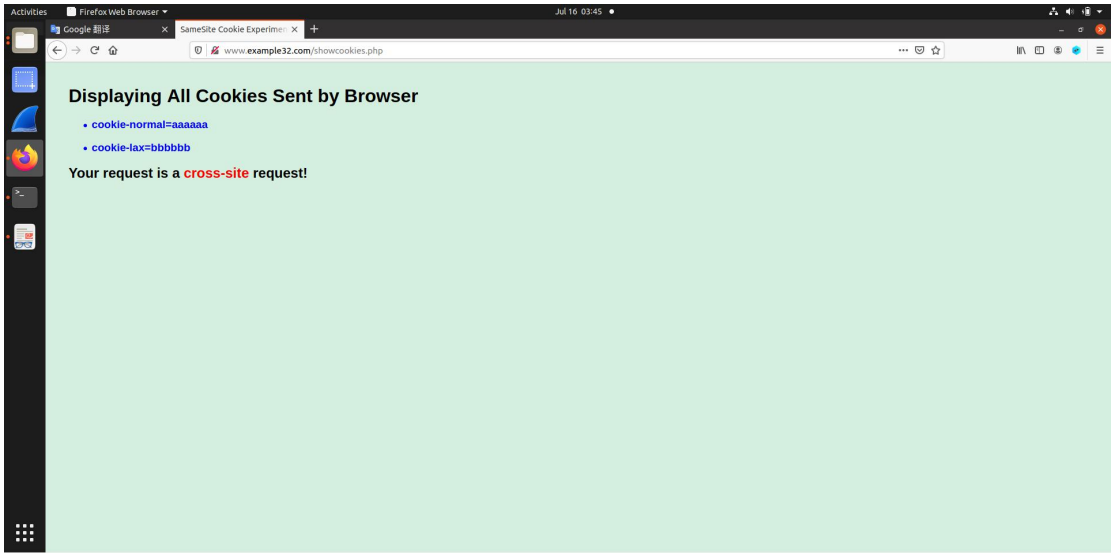


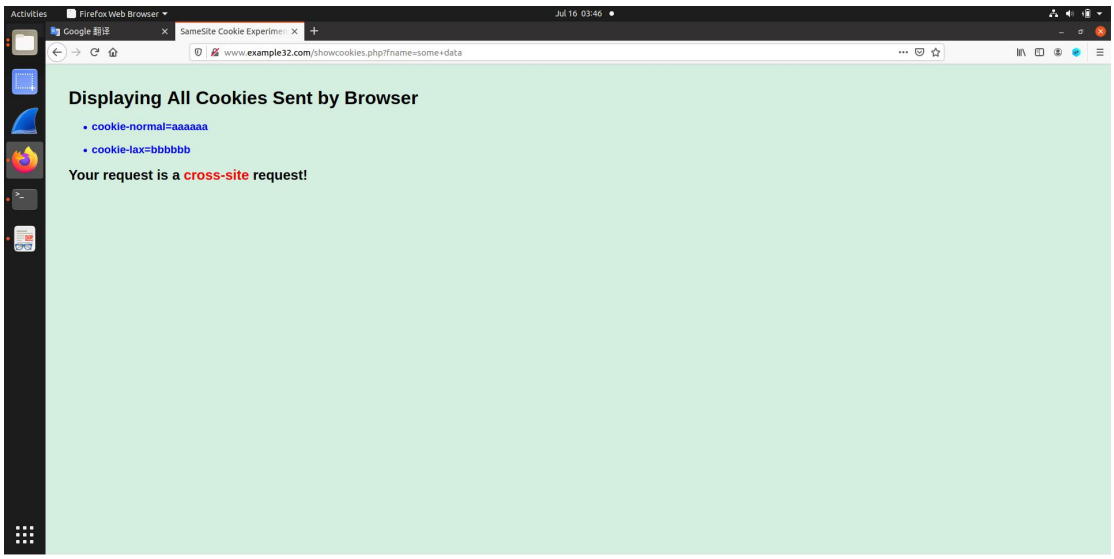

Activities
Firefox Web Browser
Jul 16 03:44
Google 翻译
SameSite Cookie Experime
www.example32.com/showcookies.php
Displaying All Cookies Sent by Browser
cookie-normal=aaaaaa
cookie-lax=bbbbbb
cookie-strict=cccccc
Your request is a same-site request!

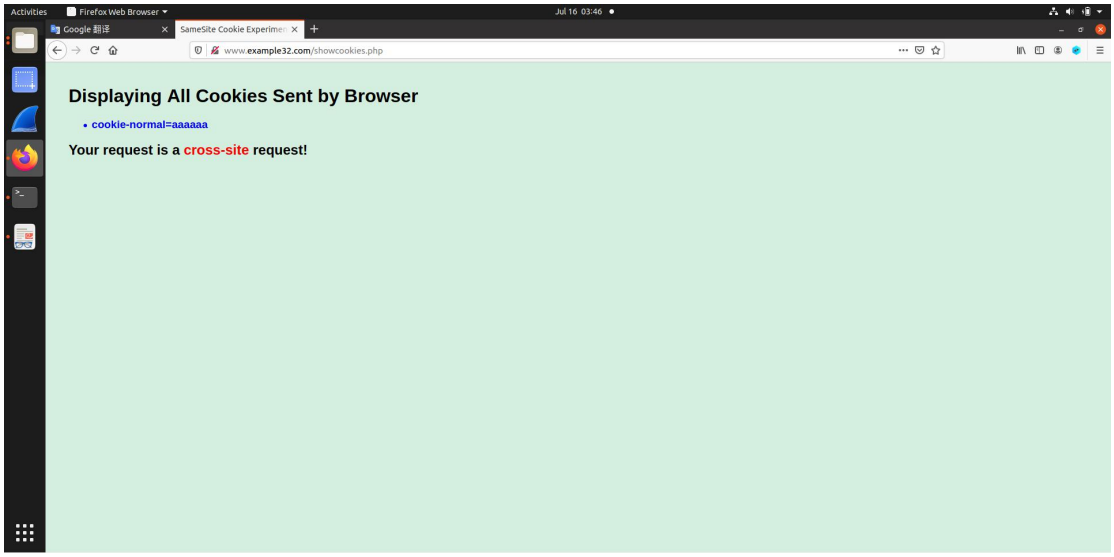

Activities
Firefox Web Browser
Jul 16 03:44
Google 翻译
SameSite Cookie Experime
www.example32.com/showcookies.php?fname=some+data
Displaying All Cookies Sent by Browser
cookie-normal=aaaaaa
cookie-lax=bbbbbb
cookie-strict=cccccc
Your request is a same-site request!


Activities
Firefox Web Browser
Jul 16 03:45
Google 翻译
SameSite Cookie Experime
www.example32.com/showcookies.php
Displaying All Cookies Sent by Browser
cookie-normal=aaaaaa
cookie-lax=bbbbbb
cookie-strict=cccccc
Your request is a same-site request!


Activities
Firefox Web Browser
Jul 16 03:45
Google 翻译
SameSite Cookie Experime
www.attacker32.com/testing.html
SameSite Cookie Experiment
A. Sending Get Request (link)
http://www.example32.com/showcookies.php
B. Sending Get Request (form)
some data
Submit (GET)
C. Sending Post Request (form)
some data
Submit (POST)


Activities
Firefox Web Browser
Jul 16 03:45
Google 翻译
SameSite Cookie Experime
www.example32.com/showcookies.php
Displaying All Cookies Sent by Browser
cookie-normal=aaaaaa
cookie-lax=bbbbbb
Your request is a cross-site request!


Activities
Firefox Web Browser
Jul 16 03:46
Google 翻译
SameSite Cookie Experime
www.example32.com/showcookies.php?fname=some+data
Displaying All Cookies Sent by Browser
cookie-normal=aaaaaa
cookie-lax=bbbbbb
Your request is a cross-site request!



结论：

①当当前网址和目的网址相同时(同站请求)，三种 cookie 都发送。

②当当前网址和目的网址不同时(跨站请求)，普通 cookie 仍然无差别发送，strict_cookie 完全不发送，lax_cookie 是在请求数据时发送，在上传数据时不发送。

③当在浏览器中设置 cookie 属性。strict 属性基本可以防范所有 CSRF 攻击，但对使用有一定影响；lax 属性基本可以防范大部分 CSRF 攻击，且对使用的影响更小。

实验总结：

本次实验是我们的第四次实验，经过本次实验，我总结了如下的知识点:

①当被攻击者维护一个活跃的我们想要攻击的网页时，我们可以通过发送我们的攻击网址给被攻击者，通过在攻击网址内嵌入加载图片等一些可以隐蔽发送我们希望的请求的机制，使得被攻击者进入时，我们便成功通过被攻击者的浏览器发送了一些我们希望的报文，从而达到攻击效果。这就是 CSRF 攻击。

②防范 CSRF 攻击主要有两个方面：

（1）网站方面：通过在请求内隐藏一些与用户有关的秘密令牌哈希值，使得只有是真正用户的操作才能通过后台检测，而 CSRF 攻击者不能提供正确的令牌哈希值，从而攻击失效。

（2）浏览器方面：通过设置 cookie 属性，使得用户的跨站请求要求更加严格，限制用户在跨站请求时发送 cookie 值，从而使得攻击者的网站无法利用用户 cookie 发送报文。