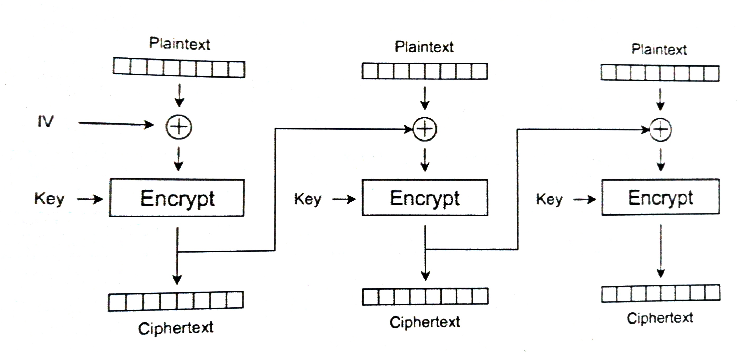
Padding Oracle攻击

最近在复现LCTF2017的一道题目，里面有一个padding oracle攻击，也算是CBC翻转攻击，这个攻击主要针对CBC加密模式的

网上有关这个攻击的博客文章很多，但是其中有一些细节可能是个人的理解不同，或者作者省略了，我这里将我在学习这个攻击中遇到的问题，记录一下

之前有写过关于CBC翻转攻击的文章，但是之前的文章是知道了密文和iv的，而且只要翻转一位就可以了（翻转多位也是可以的）

现在这个padding oracle攻击是在只知道iv，其他什么信息都不知道的情况下进行的，所以这里我一开始脑袋就绕不过了，我们知道CBC模式的加密是这样子的



一开始随机生成一个iv，然后将明文的第一块和iv进行异或，然后将第一个加密得到的密文，当作iv进行第二块的加密，然后一直下去

参考了这本书《包帽子将web安全》吴翰清的

里面有说到有一个值是固定的，我们称为middle，在其他文章中都有出现

这个攻击主要就是利用这个值，然后推导出所有的值

因为有下面这个关系

原明文^原IV = Middle

新明文^新IV = Middle

也就是说只要知道了Middle，那就可以修改原明文了，因为新明文是我们构造的

在CBC模式解密的时候，是可以从后面判断解密的，我们这里的加密解密的位数都是16位的。

Padding类型

例如我们每一块的长度是16位

如果我们的明文是15个a

那么进行异或的时候的明文就是

aaaaaaaaaaaaaaa0x01

注意后面的0x01是一个字符，是16进制的

类似的，如果我们的明文是8个a

那么进行异或的明文就是

Aaaaaaaa0x080x080x080x080x080x080x080x08

按照这样的规律，我们就可以向服务器提交一些数据，然后让服务器自动解密，如果可以解密，那就是最后一位或几位的padding值了。

这个过程就好像sql注入的盲注



例如上的例子，我们知道可控的是token（也就是IV），其他参数都是未知的，

当我们提交token的时候，程序就会自动解密，这里就有3种情况：

1. token长度不对，无法解密，服务器会报错提示
2. token可以解密，但是解出来的不是原来的明文
3. token完全正确，出现正确的明文

根据上面几点，我们就可以利用了，只要我们判断服务器不出错，程序可以解密，那就是最后的一位或几位是对的，那就可以知道我们虚拟出来的iv值，配合上我们虚拟出来的明文，就可以得到正确的Middle（因为Middle是不变的，每一次会话的Middle是一样的）

例如：Middle未知，明文未知，iv可控 （都是16进制的）

因为 Middle[15] = 明文[15]^iv[15]

我们要知道Middle值，我们虚拟Middle只有15位长度，那最后一位肯定就是0x01了，

Middle不需要知道，因为它在服务器里面，我们只要输入iv，解密函数就会自动调用了， 假如我们输入iv的值为00000000000000000000000000000065

那我们就知道了Middle的最后一个值了，也就是Middle[15] = (0x65)^(0x01)，这个值是固定的。

接着求Middle[14]，我们假设明文只要14个值，那么最后两位的值就是0x020x02，用16进制，因为明文改变了，那么相应的iv也要改变，也就是按照上面那个iv是不行的，所以，这里要重新计算iv值，这里要padding的值有两个，也就是要知道iv的后两位，倒数第一位可以根据上面的公式得到， iv[15] = Middle[15]^(0x02)

iv倒数第二个值是我们要得到的，继续按照上面的方式，爆破倒数第二个值，只要服务器不出错，能解密那就是正确了，得到第二个值，也就是得到了Middle的倒数第二个值，重复上面的步骤，就可以得到全部的Middle

但是我们的的明文不可能是空值，所以最起码明文的第一位我们是要猜的，也就是Middle的第一位，我们只能通过爆破得到。

得到了Middle的后15位，如果我们要将原明文解密出我们想得到的数据，

那根据公式

原明文^原IV = Middle

新明文^新IV = Middle

原明文 = 新明文^新IV^原IV

我们只要输入新IV，那解出来的就是一个新的明文，而且还是我们构造的新明文。

下面拿一个例子结合代码详细分析：

**<?php**error\_reporting(0);  
define("SECRET\_KEY", "\*\*\*\*\*\*"); //key不可知  
define("METHOD", "aes-128-cbc");  
session\_start();  
  
**function** get\_random\_token(){  
 $random\_token = '';  
 $str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz1234567890";  
 **for**($i = 0; $i < 16; $i++){  
 $random\_token .= substr($str, rand(1, 61), 1);  
 }  
 **return** $random\_token;  
}  
  
**function** get\_identity(){  
 $id = '\*\*\*'; //原明文不可知  
 $token = get\_random\_token();  
 $c = openssl\_encrypt($id, *METHOD*, *SECRET\_KEY*, *OPENSSL\_RAW\_DATA*, $token);  
 $\_SESSION['id'] = base64\_encode($c);  
 setcookie("token", base64\_encode($token));  
 $\_SESSION['isadmin'] = **false**;  
}  
**function** test\_identity(){  
 **if** (**isset**($\_SESSION['id'])) {  
 $c = base64\_decode($\_SESSION['id']);  
 $token = base64\_decode($\_COOKIE["token"]);  
 **if**($u = openssl\_decrypt($c, *METHOD*, *SECRET\_KEY*, *OPENSSL\_RAW\_DATA*, $token)){  
 **if** ($u === 'admin') {  
 $\_SESSION['isadmin'] = **true**;  
 }  
 }**else  
 echo** "Error!";  
 }  
}  
**if**(!**isset**($\_SESSION['id']))  
 get\_identity();  
test\_identity();  
**if** ($\_SESSION["isadmin"])  
 **echo** "You are admin!";  
**else  
 echo** "false";  
**?>**

(

这个例子是拿

<http://f1sh.site/2017/08/04/%E5%88%9D%E5%AD%A6padding-oracle-attack/> 的

）

源码大致流程是

1. get\_identity();

生成一个token，然后作为加密算法的IV进行对明文加密，这里的明文是不知道的，然后就是set-cookie

1. test\_identity();

进行解密，然后判断解出来的明文是什么，如果是admin，那就设置

$\_SESSION['isadmin'] = **true**;

1. **if** ($\_SESSION["isadmin"])

如果正确，那就输出You are admin

否则输出false

这里可以知道token的值是可控的，根据我们上面的分析，我们是可以控制token（也就是IV）使得解出任意的明文的

按照上面的流程走一遍

Python代码编写

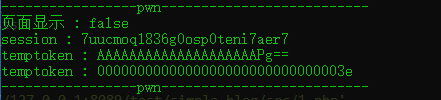
先拿到

PHPSESSID和token值



然后构造token发送给服务器

先构造15个零，最后一位爆破



得到我们构造的IV的最后一位，3e，然后相对应的Middle值就是

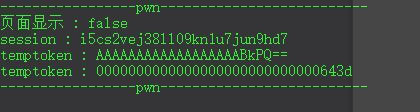


然后到将middle值记下来继续倒数第二个值

我们要构造14个零，倒数第二位是爆破的，最后一位可以算出来

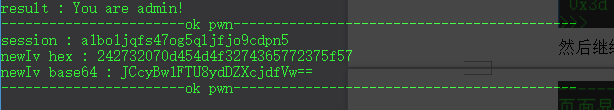


然后继续跑代码

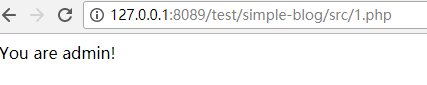


可以看到已经页面显示为false，没有Error，也就是可以知道后两位的Middle了，依次，可以得到我们所要知道的后15位Middle

完整代码跑出来的结果



在浏览器中使用相同的session和token访问



可以看到已经成功登陆进来了