mrctf-wp-令则

> 题目贼棒，出题师傅们tql！
>
> 比赛的id: 1ing23
>
> 主要是reverse题目和前仨pwn。

re

> 似乎对于动调考察比较多，
>
> ida远程动调其实挺香，go的题目是按看雪夜影师傅的文章的方法找到main函数在汇编级调试，注意参数传递啥的，

Transform

就一个异或和换位，ida-python导出数据就可，这个题还比较简单

```python
cipher = [103, 121, 123, 127, 117, 43, 60, 82, 83, 121, 87, 94, 93, 66, 123, 45, 42, 102, 66, 126, 76, 87, 121, 65, 107, 126, 101, 60, 92, 69, 111, 98, 77]
data = [9L, 10L, 15L, 23L, 7L, 24L, 12L, 6L, 1L, 16L, 3L, 17L, 32L, 29L, 11L, 30L, 27L, 22L, 4L, 13L, 19L, 20L, 21L, 2L, 25L, 5L, 31L, 8L, 18L, 26L, 28L, 14L, 0L]
for i in range(33):
    cipher[i] ^= data[i]
flag = [0] * 33
for i in range(33):
    flag[data[i]] = cipher[i]

print(''.join(map(chr, flag)))
```

PixelShooter

搜索字符串'flag'可以找到，其实在apk解压后PixelShooter/assets/bin/Data/Managed目录下面的dll文件里，是Assembly-CSharp.dll，直接明文，或者直接打游戏搞出来应该也可。(这个dll是个.net文件，用dnSpy)

```
            {
                text += "少年继续努力！要拿到flag还差亿点点\n";
            }
            else if (score < 100)
            {
                text += "战绩不错！但是要拿到flag还差亿点";
            }
            else if (score < 500)
            {
                text += "惊人的成绩！！但是要拿到flag还差一点\n";
            }
            else
            {
                text += "MRCTF{Unity_1S_Fun_233}\n";
            }
            if (Time.time - this.lastTime < 15f)
            {
                text += "以及，别作死啊！\n";
            }
            else if (Time.time - this.lastTime < 60f)
            {
                text += "以及注意闪避！";
            }
            this.gameOverText.text = text;
            this.gameOverUI.SetActive(true);
```

lol

main函数在：.text:000000000049A2D0

这个还是比较顶，后面是做了一个lua解释器么，其实动调，会看到一堆字符串处理成了指令一样的东西，大概可以有点感觉了，直接对着处理就行了。

```
.data:00007FF7FDE54B00 ; _BYTE aCmps8380738076[1008]
.data:00007FF7FDE54B00 aCmps8380738076 db 'cmps={83,80,73,80,76,125,61,96,107,85,62,63,121,122,101,33,123,82'
.data:00007FF7FDE54B00                                         ; DATA XREF: sub_7FF7FDDE1000+2↑o
.data:00007FF7FDE54B00                                         ; main+93↑o
.data:00007FF7FDE54B00 db ',101,114,54,100,101,97,85,111,39,97}',0Ah
.data:00007FF7FDE54B00 db 'print("Give Me Your Flag LOL!:")',0Ah
.data:00007FF7FDE54B00 db 'flag=io.read()',0Ah
.data:00007FF7FDE54B00 db 'if string.len(flag)~=28 then',0Ah
.data:00007FF7FDE54B00 db 9,'print("Wrong flag!")',0Ah
.data:00007FF7FDE54B00 db 9,'os.exit()',0Ah
.data:00007FF7FDE54B00 db 'end',0Ah
.data:00007FF7FDE54B00 db 'for i=1,string.len(flag) do',0Ah
.data:00007FF7FDE54B00 db 9,'local x=string.byte(flag,i)',0Ah
.data:00007FF7FDE54B00 db 9,'if i%2==0 then',0Ah
.data:00007FF7FDE54B00 db 9,9,'x=x~i',0Ah
.data:00007FF7FDE54B00 db 9,'else',0Ah
.data:00007FF7FDE54B00 db 9,9,'x=x+6',0Ah
.data:00007FF7FDE54B00 db 9,'end',0Ah
.data:00007FF7FDE54B00 db 9,'if x~=cmps[i] then',0Ah
.data:00007FF7FDE54B00 db 9,9,'print("Wrong flag!")',0Ah
.data:00007FF7FDE54B00 db 9,9,'os.exit()',0Ah
.data:00007FF7FDE54B00 db 9,'end',0Ah
.data:00007FF7FDE54B00 db 'end',0Ah
.data:00007FF7FDE54B00 db 'print("Right flag!")',0Ah
.data:00007FF7FDE54B00 db 'os.exit()',0Ah,0
.data:00007FF7FDE54C9C db    0
```

```python
arr = [83,80,73,80,76,125,61,96,107,85,62,63,121,122,101,33,123,82,101,114,54,100,101,97,85,111,39,97]
s = ''
for i in range(len(arr)):
    if i%2 ==0:
        s += chr(arr[i] - 6)
    else :
        s += chr(arr[i] ^ (i + 1))
print(s)
```

hello_world_go

main 函数在：.text:000000000049A2D0

直接去调试找到了main 函数，然后后面是明文flag吧，

可以看到一个flag长度，然后后面的check 函数参数中是明文flag

在.text:000000000049A428的check函数调用位置，参数中就有

```
004D3C56 db  72h ; r
004D3C57 db  74h ; t
004D3C58 aFlagHelloWorld db 'flag{hello_world_gogogo}function not im
004D3C58                                            ; DATA XREF: main:l
004D3C58                                            ; main+25C↑o ...
004D3C58 db 'correcthash of unhashable type initSpan: unaligned base
004D3C58 db 't synchronizedlink number out of rangeout of streams re
004D3C58 db 'uefinalizer during GCreflect.Value.SetComplexrungsteal:
```

Junk

main函数在：.text:00401390

然后几个处理的位置，一个类似rol，和一个ror 4的吧，后面一个变表的base64还可以看出来，

check函数： .text:00401090这里是先对flag进行的处理，然后进入cipher函数，base64编码下，后面是和一个写好的密文对比

cipher函数：.text:00401090是一个变表的base64

密文数据在： .text:00401090

就注意一个最后应该是有'..'， 是base64补齐的'=='的位置，

```python
import base64

diy_base   = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz)!@#$%^&*(+/'
diy_cipher = '%BUEdVSHlmfWhpZn!oaWZ(aGBsZ@ZpZn!oaWZ(aGBsZ@ZpZn!oYGxnZm%w'
base       = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
cipher = ''

def ro4(x):
    return ((x << 4) & 0xff) | ((x >> 4) & 0xff)

def rol(ver, num):
    return ((ver << num) & 0xffffffff) | ((ver >> (32 - num)) & 0xffffffff)

for i in range(len(diy_cipher)):
    cipher += base[diy_base.index(diy_cipher[i])]
cipher += '=='
cipher = base64.b64decode(cipher)

print(cipher)

for i in range(len(cipher)):
    a = ro4(cipher[i])
    a ^= 3
    print(chr(a), end='')
```

Hard-to-go

main函数在:.text:0000000000499B10

scanf函数在: .text:0000000000440FE0,

print函数在.text:000000000048D980

异或的位置在：.text:0000000000471824

仔细的动调了下，好好观察函数传递的参数，发现主要就是个异或，但是中间位置生成的异或的值不是直接明文的，似乎是rc4，但是我直接动调记录下来了，然后导出cipher，直接异或就可以:

```python
arr = [0x3e, 0x64, 0x2d, 0x8f, 0x98, 0x45, 0xd0, 0x4a, 0xc3, 0xe, 0x3, 0xb, 0x86, 0x12, 0xb7, 0xee, 0xf4, 0x78, 0x45, 0xfd, 0x36, 0x36, 0x49, 0xe5, 0x8e, 0x2b, 0xbc, 0xfb, 0xd5, 0x54, 0x54]
cipher  = [125, 48, 110, 201, 204, 3, 147, 30, 133, 77, 69, 95, 197, 70, 244, 168, 160, 62, 17, 190, 112, 117, 29, 163, 205, 127, 255, 189, 129, 18]

print(len(arr))
print(len(cipher))
s = ''
for i in range(30):
    s += (chr(arr[i] ^ cipher[i]))
print(s)
```

Shit

main函数: .text:007F1640

主要的加密函数在：.text:007F13CF

主要流程是吧flag分成组，然后在函数中加密，注意动调在汇编级，还有几个函数修改了返回值，要使用f7步入才不会跑飞，

主要是一个类似ror 和shl的操作，注意中间夹杂几个异或，

```python
arr = [3, 0x10, 0xd, 4, 0x13, 0xb]
cipher = [0x8c2c133a, 0xf74cb3f6, 0xfedfa6f2, 0xab293e3b, 0x26cf8a2a, 0x88a1f279]

def ror(m, num):
    return ((m << (num)) | (m >> (0x20 - num))) & 0xffffffff

def shl(m):
    return ((~(m & 0xffff) << 16) | ((m & 0xffff0000) >> 16)) & 0xffffffff

for i in range(5, 0, -1):
    cipher[i] ^= cipher[i-1]

for i in range(6):
    cipher[i] ^= (1 << arr[i])

for i in range(6):
    cipher[i] = shl(cipher[i])
    cipher[i] = ror(cipher[i], arr[i])
print(cipher)
s = ''

for i in cipher:
```

```
    s += ( hex(i).replace('0x','').decode('hex'))

print(s)
# flag{a_3a2y_re_for_test}
```

EasyCpp

应该main函数还是比较好找。

里面重点是一个depart函数要调试理解下，

主要是输入的数据在depart处理下成为数字的一串字符串，然后再替换对应字符，和另外写好的字符对比。

大致解密：

```python
arr_str  = ['=AT=lE=ll', '=EsE=s=z', '=lE=T=E=E=E', '=EOll=E', '=s=s=s=E=E=E', '=zATT', '=ll=T=s=s=E', '=lzzE', '=zqE=z=z=z'][::-1]

def replace(str1):
    str2 = ''
    str2 = str1.replace('O', '0')
    str2 = str2.replace('l', '1')
    str2 = str2.replace('z', '2')
    str2 = str2.replace('E', '3')
    str2 = str2.replace('A', '4')
    str2 = str2.replace('s', '5')
    str2 = str2.replace('G', '6')
    str2 = str2.replace('T', '7')
    str2 = str2.replace('B', '8')
    str2 = str2.replace('q', '9')
    str2 = str2.replace('=', ' ')
    return str2

for i in range(len(arr_str)):
    print(replace(arr_str[i]))
```

然后得到字符串，去逆depart函数的位置。

发现应该是全部乘起来就可以得到原数据。这个地方在ipython时候发现就手动去乘了下，然后去调试，这个数据会进行微调处理，当时没去细看，直接调试，然后加一减一的改了改搞出来了，

是这个：

```
2345
1222
5774
2476
3374
9032
2456
3531
6720
```

然后得到这个：flag:MRCTF{md5(23451222577424763374903224563531672O)}

去md5加密下就好了。

Virtual Tree

调试，里面有花指令，不过比较简单的，看着跳转语句去改下就可以了，

> ida中快捷键：
>
> a 分析为字符串
>
> c 分析为代码
>
> d 分析为数据， 多次点击切换数据类型
>
> u 取消分析
>
> p 讲某一段代码分析为一个函数。

流程大致：先异或了一下，然后在一个fun函数，去调用另外三个简单的函数处理了下。

不过其中有一个函数，使用了abs()，导致这个位置会出现分叉，

这个时候想法是设置一个data 数组存一下可能出现的情况，然后所有情况都跑一遍，

我是先简单写了几个情况然后去跑了下，结果运气不错，第一个就得到了结果，

```python
arr = [0x4d, 0x4c, 0x47, 0x50, 0x4f, 0x4b, 0x46, 0x43, 0x4a, 0x45, 0x4e, 0x49, 0x48, 0x44, 0x42, 0x41]
data = ['00000','10000','11000','01000','11100','01100','00100','11110','01110','00110','00010','11111','01111','00111','00011','00001']
cipher = [23, 99, 119, 3, 82, 46, 74, 40, 82, 27, 23, 18, 58, 10, 108, 98]

def xor():
    for i in range(16):
        cipher[i] ^= arr[i]

def fun1(a1, a2):
    cipher[a1] -= a2

def fun2(a1, a2):
    cipher[a1] ^= cipher[a2]

def fun3(a1, a2, a3):
    a = cipher[a1]
    if ord(data[j][a3]) - 48:
        cipher[a1] = cipher[a2] + a
    else:
        cipher[a1] = cipher[a2] - a

def fun():
    fun1(15, 2)
    fun2(14,15)
    fun3(12,2, 0)
    fun2(11,12)
    fun3(10,7,1)
    fun3(9,8,2)
    fun2(8,7)
    fun1(7,3)
    fun3(6,1,3)
    fun2(4,5)
    fun3(3,7, 4)
```

```
    fun1(2,7)
    fun2(1,2)
    fun1(0,10)


for j in range(16):
    fun()
    xor()
    print(cipher)
print(''.join(map(chr,cipher)))
#MRCTF{@_7r3e_f0r_fuNN!}
```

就是全部为0，都是减法的时候就得到flag，

pwn

easyoverflow

栈溢出覆盖，去修改他们比较的值就可以直接getshell

```python
payload = 'a' * 48 + 'n0t_r3@11y_f1@g'
```

shellcode

就是直接写shellcode的题目，

可以直接用pwntools里面的shellcraft去生成，

```python
context(os='linux', arch='amd64')
payload = asm(shellcode = shellcraft.amd64.linux.sh()
)
```

注意使用asm要先设置context

Easy_equation

先python爆破了下这个算式的答案，应该是2，我们修改为2就好了，存在格式化字符串漏洞， 就直接输入'aa%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,%p,'查了下偏移，应该是会有一个'a' 在前面占格， 然后便宜到后面第二个就是8

```python
payload = 'a' + 'a%9$naaa' + p64(0x60105C)
```

前面一个'a' 占格，然后是%9$n， 后面几个a也是占格，接上地址，ok。

crypto

keyboard

打开以后看到数字。之前看到过一个多重的加密的帖子，恰好里面第一重就是这个，

就直接对着手机九键写就好了，

Ethereum

SimpleReveal

emmm这个说是简单题，进去以后找找，能够在最后面那个没有红色叹号里面找到flag，

Feedback

MRCTF2020问卷调查

就点击进入链接，

反馈和最后几个框：" 出题人最强！"

得到flag。

```
逃
```