ZCTF - 2016 - Reverse100

原创

风靡义磊 于 2016-07-21 13:38:11 发布 1580 收藏

分类专栏： 逆向 文章标签： ZCTF 逆向

本文链接：https://blog.csdn.net/w597013296/article/details/51982072

题目链接

题目只给了一个程序，没有其他信息。打开程序，发现是控制台，接受用户输入。输入完回车后就退出了。打开OD载入，发现没有加壳。一路F8直到程序处于“运行”状态（获取用户输入）。在call 004011F0处停住。如果在这一句的下一句设断，输入完后程序就退了，说明主程序就在004011F0里面。

```
00407B80  |.  50            push    eax
00407B81  |.  FF35 44754100 push    dword ptr [417544]
00407B87  |.  FF35 40754100 push    dword ptr [417540]
00407B8D  |.  E8 5E96FFFF   call    004011F0
00407B92  |.  83C4 0C       add     esp, 0C
00407B95  |.  8945 E4       mov     dword ptr [ebp-1C], eax
00407B98  |.  50            push    eax
00407B99  |.  E8 A1F3FFFF   call    00406F3F
```

于是重开程序，在call 004011F0这里F7跟进，同样发现程序在call 00401C00这一句运行起来。在后一句设断，发现输入完后程序就停住了。猜想00401C00是获取输入的，我把他注释为scanf。下一步的lea就是把输入字符串的地址存到edi里面。

```
004011FF   .  68 C0714100   push    004171C0
00401204   .  E8 F7090000   call    00401C00
00401209   .  8D7C24 58     lea     edi, dword ptr [esp+58]
0040120D   .  83C9 FF       or      ecx, FFFFFFFF
00401210   .  33C0          xor     eax, eax
```

下面会有3步操作，经过分析发现是将字符串长度存到ecx里。

```
00401215   .  F2:AE         repne   scas byte ptr es:[edi]
00401217   .  F7D1          not     ecx
00401219   .  49            dec     ecx
```

然后程序判断ecx是不是10(h)，如果不是则不跳。可以分析出不跳的话进入的函数为退出程序，及时做好Mark。所以我们输入的字符串应该是16长度的，很可能就是最后的Flag。到达je的那一句时，我们先双击修改ZF的值让他跳过去。

```
0040121A   .  83F9 10       cmp     ecx, 10
0040121D   .  74 06         je      short 00401225
0040121F   .  50            push    eax
00401220   .  E8 1A5D0000   call    <exit>
```

跳过来后，接下来这一段是把一串字符复制到一个位置并补0，累计占了4*10(h)+1+2+1+8*4=100长度。复制的字符百度一下，为茶花女片段。

```
00401225   > \B9 10000000   mov     ecx, 10
0040122A   .  BE 9C544100   mov     esi, 0041549C                           ;  ASCII "My dear,for th
0040122F   .  8D7C24 60     lea     edi, dword ptr [esp+60]
00401233   .  33C0          xor     eax, eax
00401235   .  F3:A5         rep     movs dword ptr es:[edi], dword ptr [esi]
00401237   .  66:A5         movs    word ptr es:[edi], word ptr [esi]
00401239   .  A4            movs    byte ptr es:[edi], byte ptr [esi]
0040123A   .  B9 08000000   mov     ecx, 8
0040123F   .  8DBC24 A30000>lea     edi, dword ptr [esp+A3]
00401246   .  F3:AB         rep     stos dword ptr es:[edi]
00401248   .  AA            stos    byte ptr es:[edi]
```

后面是进行类似的操作，总计16句话，在内存中依次存放，长度均补成100。最后我们来到一串赋值处，将16个dword值写在上面那些句子之前。

```
004014B5   .  C74424 10 86B>mov     dword ptr [esp+10], 2BC86
004014BD   .  C74424 14 B8A>mov     dword ptr [esp+14], 2ADB8
004014C5   .  C74424 18 18C>mov     dword ptr [esp+18], 2C218
004014CD   .  C74424 1C 5EC>mov     dword ptr [esp+1C], 2C85E
004014D5   .  C74424 20 42C>mov     dword ptr [esp+20], 2CA42
004014DD   .  C74424 24 58C>mov     dword ptr [esp+24], 2C158
004014E5   .  C74424 28 C0C>mov     dword ptr [esp+28], 2C4C0
004014ED   .  C74424 2C ABB>mov     dword ptr [esp+2C], 2B8AB
004014F5   .  C74424 30 49C>mov     dword ptr [esp+30], 2C549
004014FD   .  C74424 34 5BB>mov     dword ptr [esp+34], 2B15B
00401505   .  C74424 38 DBA>mov     dword ptr [esp+38], 2A7DB
0040150D   .  C74424 3C AAB>mov     dword ptr [esp+3C], 2BEAA
00401515   .  C74424 40 81B>mov     dword ptr [esp+40], 2BA81
0040151D   .  C74424 44 F1A>mov     dword ptr [esp+44], 2ACF1
00401525   .  C74424 48 08B>mov     dword ptr [esp+48], 2BA08
0040152D   .  C74424 4C D8A>mov     dword ptr [esp+4C], 2ACD8
```

接下来一段将第一句英文的地址压栈，（观察数据窗）路过call之后发现英文的逗号空格都没了，但是占位还是100。不妨先做好注释。

```
00401535   .  33DB          xor     ebx, ebx
00401537   .  8D6C24 10     lea     ebp, dword ptr [esp+10]
0040153B   .  8D7424 60     lea     esi, dword ptr [esp+60]
0040153F   >  56            push    esi
00401540   .  33FF          xor     edi, edi
00401542   .  E8 B9FAFFFF   call    <去掉字符串的空格逗号>
```

下面的操作就很重要了。程序读取第一段英文的第一个字符，读取用户输入的第一个字符，乘起来并加到edi里面。接着循环16次，即读取16个字符分别相乘相加存入edi，再判断值是否与ebp指向位置的值相同。在数据窗中来到ebp指向的位置，发现这个位置就是前面复制的16个dword值中的第一个。如果不相等程序就退出了，我们在jnz那里修改ZF的值，进行之后的循环，很快就找到了规律。程序让用户输入的16个字符与茶花女给出的16句话的前16个字符（去掉空格逗号等）相乘相加，再判断是否与给定的16个值相同。

```
00401547   .  83C4 04       add     esp, 4
0040154A   .  33C0          xor     eax, eax
0040154C   >  0FBE0C06      movsx   ecx, byte ptr [esi+eax]
00401550   .  0FBE5404 50   movsx   edx, byte ptr [esp+eax+50]
00401555   .  0FAFCA        imul    ecx, edx
00401558   .  03F9          add     edi, ecx
0040155A   .  40            inc     eax
0040155B   .  83F8 10       cmp     eax, 10
0040155E   .^ 7C EC         jl      short 0040154C
00401560   .  397D 00       cmp     dword ptr [ebp], edi
00401563   .  0F85 89000000 jnz     <exit>
00401569   .  43            inc     ebx
0040156A   .  83C6 64       add     esi, 64
0040156D   .  83C5 04       add     ebp, 4
00401570   .  83FB 10       cmp     ebx, 10
00401573   .^ 7C CA         jl      short 0040153F
```

为了解这个16元线性方程组，写个程序将这些数据以十进制输出，然后用matlab解，再以字符格式输出（我不知道有没有办法在matlab里面直接输入16进制数），发现用户输入的应该是“zctf{Wrong_Flag}”。

这什么鬼。我一下子崩溃了。如果确信前面的步骤没有分析错，就知道后面还有文章。接着程序输出”zctf@You are almost so clever!”，然后来到call 00401050。如果之前查看程序字符串，就知道还有一句”zctf@You are so clever!”，这应该就是最终的结果了。这一段字符串在00401050被引用，可见这里面大有玄机。选择跟进（不然就return了），并注意到用户输入的字符串地址被压栈。

```
004015DA   > \8D4C24 50     lea     ecx, dword ptr [esp+50]
004015DE   .  51            push    ecx
004015DF   .  E8 6CFAFFFF   call    00401050
004015E4   .  83C4 04       add     esp, 4
004015E7   .  5F            pop     edi
004015E8   .  5E            pop     esi
004015E9   .  5D            pop     ebp
004015EA   .  5B            pop     ebx
004015EB   .  81C4 90060000 add     esp, 690
004015F1   .  C3            retn
```

进入后发现又是一个scanf，接着把字符串长度存入ecx并压栈调用00401600输出字符串长度，但是不知道有什么用。然后还是获取字符串长度，与14(h)比较，不相等就退出。说明第二次输入的字符串长度为20。在je处把ZF改掉，跳转后继续分析。

```
0040106A  |.  E8 910B0000   call    <scanf>
0040106F  |.  8D7C24 14     lea     edi, dword ptr [esp+14]
00401073  |.  83C9 FF       or      ecx, FFFFFFFF
00401076  |.  33C0          xor     eax, eax
00401078  |.  83C4 08       add     esp, 8
0040107B  |.  F2:AE         repne   scas byte ptr es:[edi]
0040107D  |.  F7D1          not     ecx
0040107F  |.  49            dec     ecx
00401080  |.  51            push    ecx
00401081  |.  B9 30714100   mov     ecx, 00417130
00401086  |.  E8 75050000   call    00401600
0040108B  |.  8D7C24 0C     lea     edi, dword ptr [esp+C]
0040108F  |.  83C9 FF       or      ecx, FFFFFFFF
00401092  |.  33C0          xor     eax, eax
00401094  |.  F2:AE         repne   scas byte ptr es:[edi]
00401096  |.  F7D1          not     ecx
00401098  |.  49            dec     ecx
00401099  |.  83F9 14       cmp     ecx, 14
0040109C  |.  74 06         je      short 004010A4
0040109E  |.  50            push    eax
0040109F  |.  E8 9B5E0000   call    <exit>
```

又是一连串赋值。因为ebx已经被xor置零了，所以操作里面的ebx就是0。

```
004010A4  |> \33DB          xor     ebx, ebx
004010A6  |.  C74424 30 320>mov     dword ptr [esp+30], 232
004010AE  |.  895C24 20     mov     dword ptr [esp+20], ebx
004010B2  |.  895C24 24     mov     dword ptr [esp+24], ebx
004010B6  |.  895C24 28     mov     dword ptr [esp+28], ebx
004010BA  |.  895C24 2C     mov     dword ptr [esp+2C], ebx
004010BE  |.  C74424 34 850>mov     dword ptr [esp+34], 285
004010C6  |.  C74424 38 F40>mov     dword ptr [esp+38], 2F4
004010CE  |.  C74424 3C 530>mov     dword ptr [esp+3C], 353
004010D6  |.  C74424 40 980>mov     dword ptr [esp+40], 398
004010DE  |.  C74424 44 F90>mov     dword ptr [esp+44], 3F9
004010E6  |.  C74424 48 6C0>mov     dword ptr [esp+48], 46C
004010EE  |.  C74424 4C E50>mov     dword ptr [esp+4C], 4E5
004010F6  |.  C74424 50 440>mov     dword ptr [esp+50], 544
004010FE  |.  C74424 54 930>mov     dword ptr [esp+54], 593
00401106  |.  C74424 58 FB0>mov     dword ptr [esp+58], 5FB
0040110E  |.  C74424 5C 5A0>mov     dword ptr [esp+5C], 65A
00401116  |.  C74424 60 A10>mov     dword ptr [esp+60], 6A1
0040111E  |.  C74424 64 100>mov     dword ptr [esp+64], 710
00401126  |.  C74424 68 740>mov     dword ptr [esp+68], 774
0040112E  |.  C74424 6C F10>mov     dword ptr [esp+6C], 7F1
```

下面是一个小循环，观察数据窗知，程序判断用户第二次输入的前4个字符与第一个输入的前4个字符是否相等，有不相等就退出。因为前面我们已经推出来了，所以第二次输入的前面4个字符就是zctf。

```
00401136  |.  33C0          xor     eax, eax
00401138  |>  8A4C04 0C     /mov     cl, byte ptr [esp+eax+C]
0040113C  |.  8A5404 74     |mov     dl, byte ptr [esp+eax+74]
00401140  |.  3ACA          |cmp     cl, dl
00401142  |.  75 35         |jnz     short <exit>
00401144  |.  40            |inc     eax
00401145  |.  83F8 04       |cmp     eax, 4
00401148  |.^ 7C EE         \jl      short 00401138
```

下面可以通过跟踪感觉得出结论。程序让ecx从5开始每次加1，取用户输入字符串的前ecx个求和，与上面写入的16个整数对比，如果有不相同的就会退出。写个程序把上面那16个整数错位相减一下，输出字符（第一个字符要用第一个整数减去zctf的字符和来得到），就得到最终结果：zctf{So_Easy_Oh_God}。（Easy个毛线！）

```
0040114A  |.  B9 05000000   mov     ecx, 5
0040114F  |.  8D7424 30     lea     esi, dword ptr [esp+30]
00401153  |>  33D2          /xor     edx, edx
00401155  |.  33C0          |xor     eax, eax
00401157  |.  3BCB          |cmp     ecx, ebx
00401159  |.  7E 0C         |jle     short 00401167
0040115B  |>  0FBE7C04 0C   |/movsx   edi, byte ptr [esp+eax+C]
00401160  |.  03D7          ||add     edx, edi
00401162  |.  40            ||inc     eax
00401163  |.  3BC1          ||cmp     eax, ecx
00401165  |.^ 7C F4         |\jl      short 0040115B
00401167  |>  3916          |cmp     dword ptr [esi], edx
00401169  |.  75 14         |jnz     short <exit>
0040116B  |.  41            |inc     ecx
0040116C  |.  83C6 04       |add     esi, 4
0040116F  |.  8D51 FF       |lea     edx, dword ptr [ecx-1]
00401172  |.  83FA 14       |cmp     edx, 14
00401175  |.^ 7C DC         \jl      short 00401153
00401177  |.  EB 0C         jmp     short 00401185
```

接下来可以重新载入程序，输入输出如下：

zctf{Wrong_Flag}

zctf@You are almost so clever!

zctf{So_Easy_Oh_God}

20zctf@You are so clever!

要稳妥一点还可以继续跟踪，已经没内容了。

注：writeup写于2016年1月25日，第一次接触CTF比赛，留作纪念