

XCTF-Reverse-ExerciseArea-005-writeup

原创

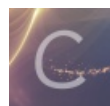
y4ung 于 2019-08-01 08:39:59 发布 4671 收藏 1

分类专栏: [ctf](#) 文章标签: [CTF ReverseEngineering](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/qq_35056292/article/details/98025948

版权



[ctf](#) 专栏收录该内容

35 篇文章 0 订阅

订阅专栏

0x00 介绍

本题是xctf攻防世界中Reverse的新手第五题。

根据题目描述: 菜鸡拿到了一个被加壳的二进制文件, 可以知道这次的二进制文件被加壳处理了, 因此需要先查壳, 脱壳, 再进行逆向分析找到flag

实验环境: IDA Pro 7.0

0x01 解题过程

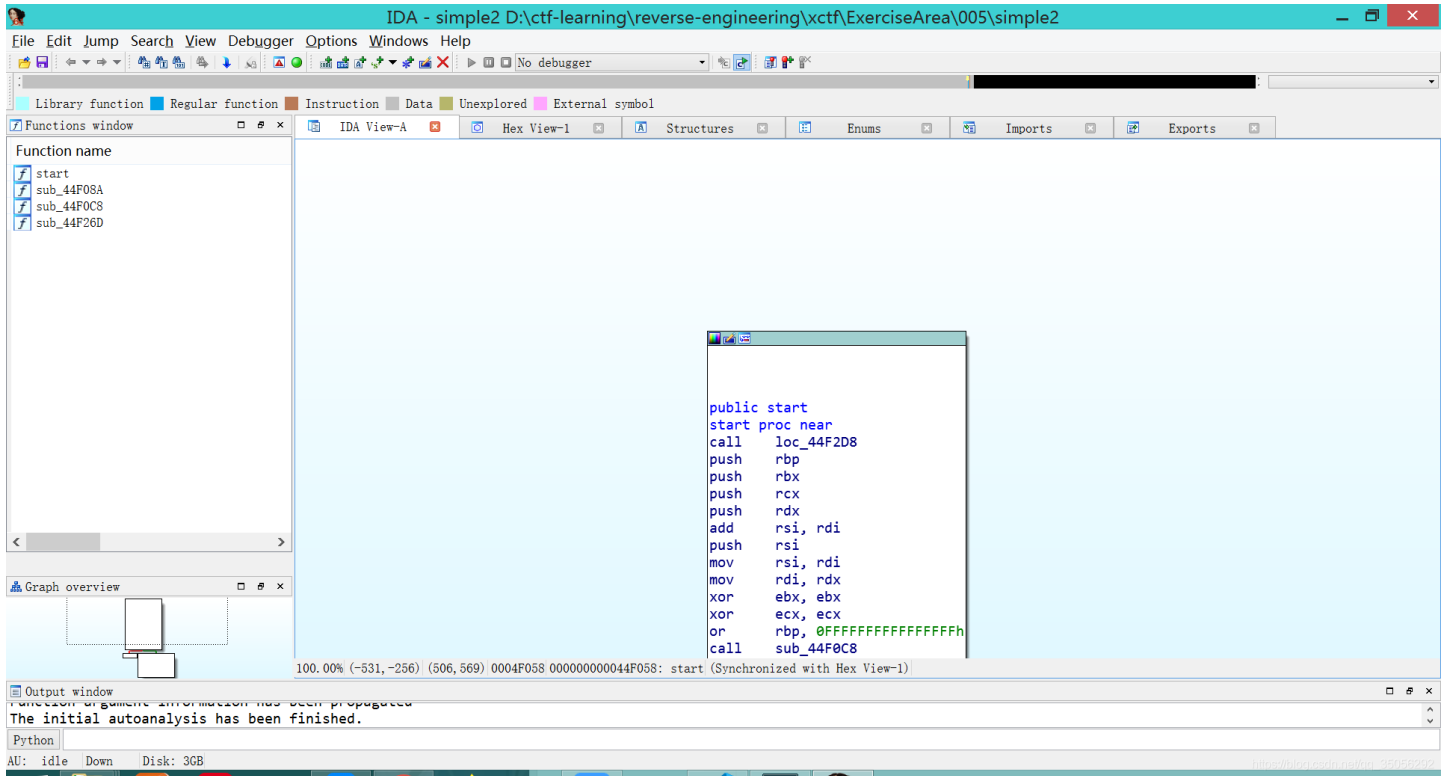
1.1 判断文件类型

在Vscode中安装插件: [hexdump for VSCode](#), 用Vscode打开, 显示文件的十六进制:

```
1 | Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F | .ELF...
2 | 00000000: 7F 45 4C 46 02 01 01 03 00 00 00 00 00 00 00 00 | -->...XpD...
3 | 00000010: 02 00 3E 00 01 00 00 00 58 F0 44 00 00 00 00 00 | @...@...
4 | 00000020: 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ...@.8...@...
5 | 00000030: 00 00 00 00 40 00 38 00 02 00 40 00 00 00 00 00 | .....@...@...
6 | 00000040: 01 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00 | ..@.....@...
7 | 00000050: 00 00 40 00 00 00 00 00 00 00 40 00 00 00 00 00 | lx.....lx.....
8 | 00000060: 6C F8 04 00 00 00 00 00 6C F8 04 00 00 00 00 00 | (T.....(T1....
9 | 00000070: 00 00 20 00 00 00 00 00 01 00 00 00 06 00 00 00 | (T1.....
10 | 00000080: 28 D4 0C 00 00 00 00 00 28 D4 6C 00 00 00 00 00 | T-."UPX!.....
11 | 00000090: 28 D4 6C 00 00 00 00 00 00 00 00 00 00 00 00 00 | (m..(m.....
12 | 000000a0: 00 00 00 00 00 00 00 00 20 00 00 00 00 00 00 00 | ...w{...ELF...
13 | 000000b0: D4 AD 80 A2 55 50 58 21 1C 08 0D 16 00 00 00 00 | ...>.....@_/[[@
14 | 000000c0: A8 ED 0D 00 A8 ED 0D 00 90 01 00 00 91 00 00 00 | /he.E&8...!..1'?
15 | 000000d0: 08 00 00 00 F7 FB 93 FF 7F 45 4C 46 02 01 01 03 | .W...@.../{m...
16 | 000000e0: 00 02 00 3E 00 01 0E 90 08 40 1F DF 2F EC DB 40 | .o..8g.R2./..1.[p
17 | 000000f0: 2F 68 E5 0D 45 26 38 00 06 0A 21 00 1F 6C 60 BF | 5Zvv0.o.....+@..
18 | 00001000: 1E 57 05 00 01 40 0F 86 96 0C AF 7B 6D 20 00 20 | y@D...m6D6..._OP
19 | 00001100: 0B 6F 06 06 B8 67 13 D2 B2 9E 2F 0E 6C 1C 5B 70 | .MA~I.Qetd....
20 | 00001200: 35 5A F6 76 B0 00 6F 04 07 90 01 2B 0E 40 93 03 | 4;.Rn_H...I..[.
21 | 00001300: F9 40 44 00 00 04 ED 36 C4 B6 07 17 DF 20 4F 50 | ...$.v...g1...I.
22 | 00001400: 0F 4D C1 FE C9 08 51 E5 74 64 06 00 01 10 60 0F | .5{yM....GNU...
23 | 00001500: B4 BB 0F 52 6E DF 48 01 00 0F 49 92 84 DB 0D 00 | 7W.g...?...?Ron.
24 | 00001600: 00 00 95 24 FF F6 94 0C 00 67 EC 04 00 08 49 19 | .{.[.s91.F.,y'9S
25 | 00001700: 00 B5 FB 79 CD 04 00 10 06 01 47 4E 55 0A 00 02 | .'B<g'.lw_1;=%..
26 | 00001800: B7 D7 9D E7 06 20 3F 14 06 03 3F D2 EF EE FF | ..B.X/.BH1Y.}.P/
27 | 00001900: FF FB 1A DB 99 73 39 31 7F C6 8E 2C F9 60 B9 D3 | P..H/H...y'h@0o.2
28 | 00001a00: 86 27 42 3C 67 60 A0 6C 77 5F EC BB BD 25 0B 00 |
29 | 00001b00: 90 14 42 0F 58 2F 00 C2 48 B1 D9 8B 7D 0D 50 2F |
30 | 00001c00: D0 16 5F 48 2F C8 0B 19 79 60 68 40 30 6F 17 32 |
```

可以看到文件的开头有 **ELF**，说明这是一个在Linux下的可执行文件；相应的，如果再文件开头看到 **MZ**，说明是在Windows下的可执行文件；

用IDA打开原始文件simple_2，可以看到识别出来的函数很少，应该就是被加壳了



1.2 脱壳

首先应当查壳，可以用PEID查。

在ctf比赛中的pwn大多在Linux下，一般linux下很少有强力的壳，利用upx工具对该二进制文件进行脱壳

```
upx.exe -d D:\ctf-learning\reverse-engineering\xctf\ExerciseArea\005\simple2 -o simple2_upx
```

```

cmd

振洋@LENOVO-HZY D:\吾爱破解工具包\Tools\PETools\CFF_Explorer\Extensions\CFF Explorer\UPX Utility
$ upx.exe -d D:\ctf-learning\reverse-engineering\xctf\ExerciseArea\005\simple2 -o simple2_upx
      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2011
UPX 3.08w Markus Oberhumer, Laszlo Molnar & John Reiser Dec 12th 2011

-----
File size  Ratio    Format    Name
-----
914906 <-  352624  38.54%  linux/ELFAMD  simple2_upx

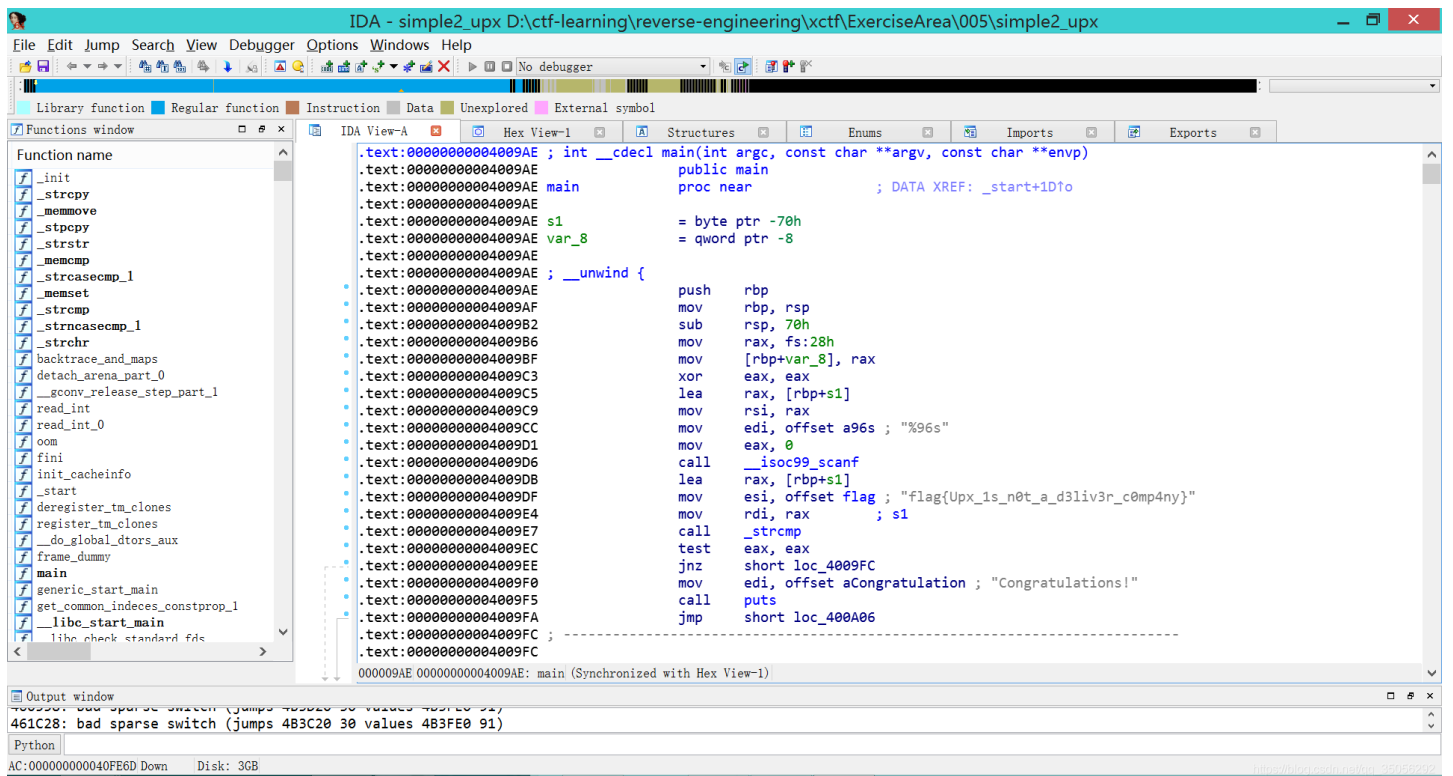
Unpacked 1 file.

振洋@LENOVO-HZY D:\吾爱破解工具包\Tools\PETools\CFF_Explorer\Extensions\CFF Explorer\UPX Utility
5

```

得到脱壳后的二进制文件: simple2_upx

用IDA打开, 可以看到识别出来的函数变多了



```

IDA - simple2_upx D:\ctf-learning\reverse-engineering\xctf\ExerciseArea\005\simple2_upx

Function name
  _init
  _strcpy
  _memcpy
  _stpcpy
  _strchr
  _memcmp
  _strcasemp_1
  _memset
  _strcmp
  _strncasemp_1
  _strchr
  backtrace_and_maps
  detach_arena_part_0
  __gconv_release_step_part_1
  read_int
  read_int_0
  oom
  fini
  init_cacheinfo
  _start
  deregister_tm_clones
  register_tm_clones
  __do_global_dtors_aux
  frame_dummy
  main
  generic_start_main
  get_common_indeces_constprop_1
  __libc_start_main
  __libc_check_standard_file

.intel_syntax noprefix
.intel_syntax align=16
.intel_syntax noframepointer
.intel_syntax nostackframe
.intel_syntax nounwind
.intel_syntax nox87

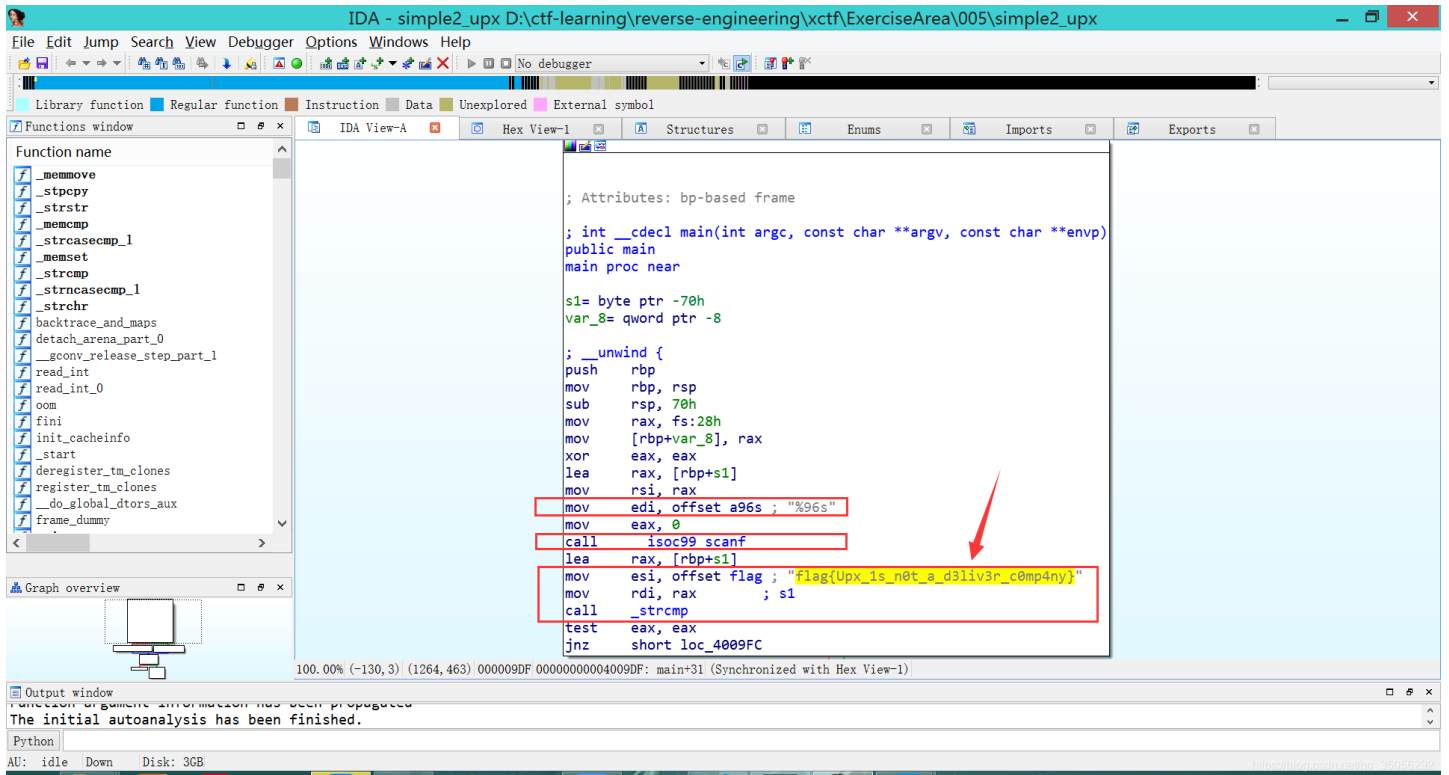
main
public main
proc near     ; DATA XREF: _start+1D10
  .text:0000000004009AE ; int __cdecl main(int argc, const char **argv, const char **envp)
  .text:0000000004009AE public main
  .text:0000000004009AE main
  .text:0000000004009AE     proc near     ; DATA XREF: _start+1D10
  .text:0000000004009AE     = byte ptr -70h
  .text:0000000004009AE     var_8
  .text:0000000004009AE     = qword ptr -8
  .text:0000000004009AE     ; __unwind {
  .text:0000000004009AE     push     rbp
  .text:0000000004009AF     mov     rbp, rsp
  .text:0000000004009AF     sub     rsp, 70h
  .text:0000000004009B2     mov     rax, fs:28h
  .text:0000000004009B6     mov     [rbp+var_8], rax
  .text:0000000004009B9     xor     eax, eax
  .text:0000000004009C3     lea    rax, [rbp+s1]
  .text:0000000004009C9     mov     rsi, rax
  .text:0000000004009CC     mov     edi, offset a96s ; "%96s"
  .text:0000000004009D1     mov     eax, 0
  .text:0000000004009D6     call   __isoc99_scanf
  .text:0000000004009DB     lea    rax, [rbp+s1]
  .text:0000000004009DF     mov     esi, offset flag ; "flag{Upx_1s_n0t_a_d3l1v3r_c0mp4ny}"
  .text:0000000004009E4     mov     rdi, rax
  .text:0000000004009E7     call   _strcmp
  .text:0000000004009EC     test   eax, eax
  .text:0000000004009EE     jnz    short loc_4009FC
  .text:0000000004009F0     mov     edi, offset aCongratulation ; "Congratulations!"
  .text:0000000004009F5     call   puts
  .text:0000000004009FA     jmp    short loc_400A06
  .text:0000000004009FC     ; -----
  .text:0000000004009FC
  .text:0000000004009FC
  000009AE 000000000004009AE: main (Synchronized with Hex View-1)

```

1.3 逆向分析

用IDA打开, 可以看到main函数中, 需要输入96个字符, 然后将用户的输入与一个字符串比较。可以看到字符串是 flag{...} 的形式。因此, 到这里为止就拿到了flag:

flag{Upx_1s_n0t_a_d3l1v3r_c0mp4ny}



这题考察的就是查壳和脱壳了，逆向分析这块不是考察重点。