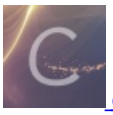
XCTF-Reverse-ExerciseArea-008-writeup

原创

y4ung 于 2019-08-05 19:04:09 发布 4055 收藏

分类专栏： ctf 文章标签： ctf ReverseEngineering



0x00 介绍

本题是xctf攻防世界中Reverse的新手第八题。题目来源： `9447 CTF 2014`

需要对该二进制文件no_strings_attached进行逆向分析，找到flag

实验环境：IDA Pro 7.0，gdb

0x01 解题过程

1.1 文件分析

1. 在Vscode中安装插件： `hexdump for VSCode` ，用Vscode打开，显示文件的十六进制：

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；

2. 在kali中用 file 命令，可以看到这是一个32bit的系统中编译的文件，同时可以看到该文件编译后符号表没有被strip掉

```
root@kali:~/hzy/ctf-learning# file no_strings_attached
no_strings_attached: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /
lib/ld-linux.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=c8d273ed1363a1878f348d6c506048f2354849d0, not stripped
```

3. 修改文件权限为可执行，运行该文件。多运行几次会发现数字1645260206是会变化的，并且注意这里有个段错误

```
root@kali:~/hzy/ctf-learning# ./no_strings_attached
Welcome to cyber malware control software.
Currently tracking 1645260206 bots worldwide
段错误
```

1.2 脱壳

用IDA打开，发现该二进制文件未被加壳，因此不需要进行脱壳操作

1.3 逆向分析

由于该文件的符号表未被去掉，因此直接用命令`b main`，在main函数处打断点进行调试

首先调用了C语言的库函数：char *setlocale(int category, const char *locale)，用于设置或读取地域化信息

```
.text:080487B2                mov     dword ptr [esp+4], offset locale ; locale
.text:080487BA                mov     dword ptr [esp], 6 ; category
```

在这里，参数locale为`""`，如果 locale 是 NULL 或空字符串 “”，则区域名称将根据环境变量值来设置；参数category的值为`6`

在gdb-peda中直接输入命令`ni`，step over，可以看到调用了`setlocale`函数后，返回值放在eax寄存器中：`EAX: 0x804e9d0 ("zh_CN.UTF-8")`

ps: 一开始没反应过来这是个库函数，看了好久= =|| 后来发现在IDA里这个函数是外部引用，并且函数名是下划线开头：`_setlocale`才反应过来

3. 接下来调用函数banner，获取当前时间作为随机数种子，生成随机数，打印banner信息：

```
Welcome to cyber malware control software.
Currently tracking 1369953942 bots worldwide
```

汇编代码如下：

```
.text:08048604
.text:08048604                public banner
.text:08048604 banner         proc near               ; CODE XREF: main+1D↓p
.text:08048604 ; __unwind {
.text:08048604                push    ebp
.text:08048605                mov     ebp, esp
.text:08048607                sub     esp, 18h
.text:0804860A                mov     dword ptr [esp], 0 ; timer
.text:08048611                call    _time           ; 获得距1970xxx的秒数，保存在eax寄存器中
.text:08048616                mov     [esp], eax      ; seed
.text:08048619                call    _srand          ; 随机数发生器的初始化函数
.text:0804861E                mov     eax, offset unk_80488B0 ; Welcome to cyber malware control software.
.text:08048623                mov     [esp], eax
.text:08048626                call    _wprintf
.text:0804862B                call    _rand
.text:08048630                mov     edx, offset unk_8048960 ; Currently tracking 1369953942 bots worldwide
.text:08048635                mov     [esp+4], eax
.text:08048639                mov     [esp], edx
.text:0804863C                call    _wprintf
.text:08048641                leave
.text:08048642                retn
.text:08048642 ; } // starts at 8048604
.text:08048642 banner         endp
.text:08048642
```

4. 接下来，调用函数prompt_authentication，输出提示信息：`Please enter authentication details:`

```
.text:08048643                 public prompt_authentication
.text:08048643 prompt_authentication proc near         ; CODE XREF: main+22↓p
.text:08048643 ; __unwind {
.text:08048643                 push    ebp
.text:08048644                 mov     ebp, esp
.text:08048646                 sub     esp, 18h
.text:08048649                 mov     eax, offset unk_80489F8 ; Please enter authentication details:
.text:0804864E                 mov     [esp], eax
.text:08048651                 call    _wprintf
.text:08048656                 leave
.text:08048657                 retn
.text:08048657 ; } // starts at 8048643
.text:08048657 prompt_authentication endp
.text:08048657
```

5. 看来最后一个函数`authenticate`就是今天的主菜了！

当运行到调用函数`wchar_t *fgetws(wchar_t *ws, int n, __FILE *stream)`时，会发现出现了segmentfault的错误

后面是writeup的内容了。 看了半天后面的函数，没发现flag就藏在decrypt里。。。去厕所面壁了

回头先看decrypt函数，`si`运行

这里将内存地址为ebp+0x8的内存区域的值赋值给eax寄存器

```
0x804865f <decrypt+7>: mov    eax,DWORD PTR [ebp+0x8]
```

打印：

```
gdb-peda$ x/ws $eax
0x8048aa8: U"v>>v
```

一脸懵逼，这些是个啥。。

```
gdb-peda$ x/286x $eax
0x8048aa8: 0x3a 0x14 0x00 0x00 0x36 0x14 0x00 0x00
0x8048ab0: 0x37 0x14 0x00 0x00 0x3b 0x14 0x00 0x00
0x8048ab8: 0x80 0x14 0x00 0x00 0x7a 0x14 0x00 0x00
0x8048ac0: 0x71 0x14 0x00 0x00 0x78 0x14 0x00 0x00
0x8048ac8: 0x63 0x14 0x00 0x00 0x66 0x14 0x00 0x00
0x8048ad0: 0x73 0x14 0x00 0x00 0x67 0x14 0x00 0x00
0x8048ad8: 0x62 0x14 0x00 0x00 0x65 0x14 0x00 0x00
0x8048ae0: 0x73 0x14 0x00 0x00 0x60 0x14 0x00 0x00
0x8048ae8: 0x6b 0x14 0x00 0x00 0x71 0x14 0x00 0x00
0x8048af0: 0x78 0x14 0x00 0x00 0x6a 0x14 0x00 0x00
0x8048af8: 0x73 0x14 0x00 0x00 0x70 0x14 0x00 0x00
0x8048b00: 0x64 0x14 0x00 0x00 0x78 0x14 0x00 0x00
0x8048b08: 0x6e 0x14 0x00 0x00 0x70 0x14 0x00 0x00
0x8048b10: 0x70 0x14 0x00 0x00 0x64 0x14 0x00 0x00
0x8048b18: 0x70 0x14 0x00 0x00 0x64 0x14 0x00 0x00
0x8048b20: 0x6e 0x14 0x00 0x00 0x7b 0x14 0x00 0x00
0x8048b28: 0x76 0x14 0x00 0x00 0x78 0x14 0x00 0x00
0x8048b30: 0x6a 0x14 0x00 0x00 0x73 0x14 0x00 0x00
0x8048b38: 0x7b 0x14 0x00 0x00 0x80 0x14 0x00 0x00
0x8048b40: 0x00 0x00 0x00 0x00 0x53 0x00 0x00 0x00
0x8048b48: 0x75 0x00 0x00 0x00 0x63 0x00 0x00 0x00
0x8048b50: 0x63 0x00 0x00 0x00 0x65 0x00 0x00 0x00
0x8048b58: 0x73 0x00 0x00 0x00 0x73 0x00 0x00 0x00
0x8048b60: 0x21 0x00 0x00 0x00 0x20 0x00 0x00 0x00
0x8048b68: 0x57 0x00 0x00 0x00 0x65 0x00 0x00 0x00
0x8048b70: 0x6c 0x00 0x00 0x00 0x63 0x00 0x00 0x00
0x8048b78: 0x6f 0x00 0x00 0x00 0x6d 0x00 0x00 0x00
0x8048b80: 0x65 0x00 0x00 0x00 0x20 0x00 0x00 0x00
0x8048b88: 0x62 0x00 0x00 0x00 0x61 0x00 0x00 0x00
0x8048b90: 0x63 0x00 0x00 0x00 0x6b 0x00 0x00 0x00
0x8048b98: 0x21 0x00 0x00 0x00 0x0a 0x00 0x00 0x00
0x8048ba0: 0x00 0x00 0x00 0x00 0x41 0x00 0x00 0x00
0x8048ba8: 0x63 0x00 0x00 0x00 0x63 0x00 0x00 0x00
0x8048bb0: 0x65 0x00 0x00 0x00 0x73 0x00 0x00 0x00
0x8048bb8: 0x73 0x00 0x00 0x00 0x20 0x00 0x00 0x00
0x8048bc0: 0x64 0x00 0x00 0x00 0x65 0x00
```

跟ASCII码对应起来就是flag：`9447{you_are_an_international_mystery}`

这题考察的是字符串的观察？。。。我还以为跟段错误有关