XCTF逆向题-no-strings-attached

原创

轻描时光 于 2021-11-25 01:08:04 发布 156 收藏
文章标签： flag python unctf

下载，IDA打开，找到main函数：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setlocale(6, &locale);
  banner();
  prompt_authentication();
  authenticate();
  return 0;
}
```


每个函数点开看一下，发现authenticate最复杂：

```
void authenticate()
{
  int ws[8192]; // [esp+1Ch] [ebp-800Ch]
  wchar_t *s2; // [esp+801Ch] [ebp-Ch]

  s2 = decrypt(&s, &dword_8048A90);
  if ( fgetws(ws, 0x2000, stdin) )
  {
    ws[wcslen(ws) - 1] = 0;
    if ( !wcscmp(ws, s2) )
      wprintf(&unk_8048B44);
    else
      wprintf(&unk_8048BA4);
  }
  free(s2);
}
```


看了下unk_8048B44和BA4，分别提示成功和失败，看来关键就在ws和s2上。

注意到有一句s2 = decrypt(&s, &dword_8048A90);

打开decrypt，发现是一个简单的置换函数，第一个入参逐个字节减第二个入参，第二个入参循环。

```
wchar_t *__cdecl decrypt(wchar_t *s, wchar_t *a2)
{
  size_t v2; // eax
  signed int v4; // [esp+1Ch] [ebp-1Ch]
  signed int i; // [esp+20h] [ebp-18h]
  signed int v6; // [esp+24h] [ebp-14h]
  signed int v7; // [esp+28h] [ebp-10h]
  wchar_t *dest; // [esp+2Ch] [ebp-Ch]

  v6 = wcslen(s);
  v7 = wcslen(a2);
  v2 = wcslen(s);
  dest = (wchar_t *)malloc(v2 + 1);
  wcscpy(dest, s);
  while ( v4 < v6 )
  {
    for ( i = 0; i < v7 && v4 < v6; ++i )
      dest[v4++] -= a2[i];
  }
  return dest;
}
```


回到前一个函数，看一下给decrypt的两个入参都是什么，然后写python：

```
x = [0x3A, 0x36, 0x37, 0x3b, 0x80, 0x7A, 0x71, 0x78, 0x63, 0x66, 0x73, 0x67, 0x62, 0x65, 0x73, 0x60, 0x6B,

y = [1, 2,3,4,5]

i = 0
r = ''
for a in x:
  a -= y[i]
  i = (i+1) % 5
  r += chr(a)

print(r)
```

运行得到结果：

```
┌──(.venv)(rock㉿a6a4bfb8ac4c)-[~/ctf]
└─$ /home/rock/ctf/.venv/bin/python /home/rock/ctf/1/tmp.py
9447{you_are_an_international_mystery}
```

然后去看了下别人的writeup，发现是用gdb解，今天太困了，下次学一下。

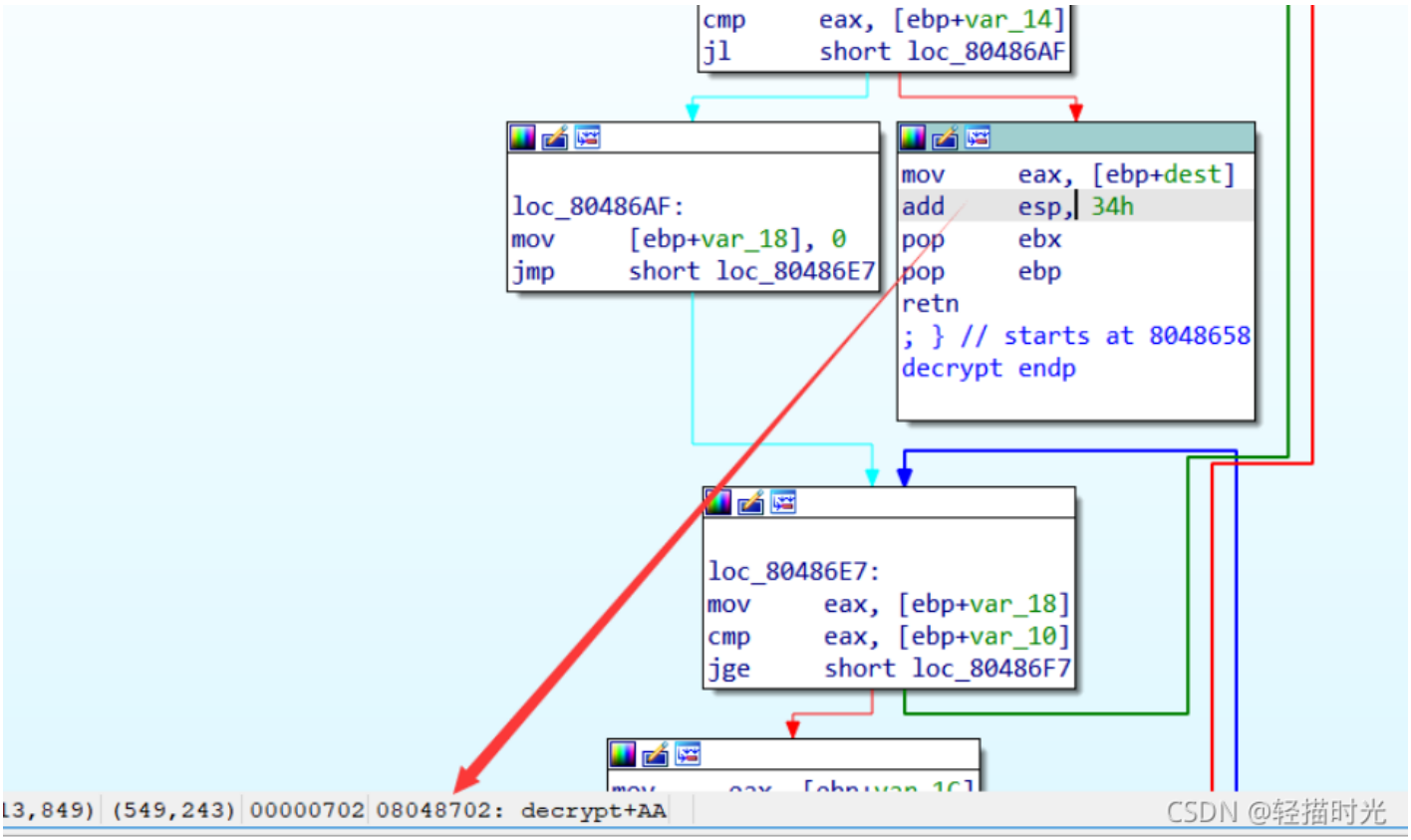
方法二（使用gdb）：

还是decrypt这张图

```
wchar_t *__cdecl decrypt(wchar_t *s, wchar_t *a2)
{
  size_t v2; // eax
  signed int v4; // [esp+1Ch] [ebp-1Ch]
  signed int i; // [esp+20h] [ebp-18h]
  signed int v6; // [esp+24h] [ebp-14h]
  signed int v7; // [esp+28h] [ebp-10h]
  wchar_t *dest; // [esp+2Ch] [ebp-Ch]

  v6 = wcslen(s);
  v7 = wcslen(a2);
  v2 = wcslen(s);
  dest = (wchar_t *)malloc(v2 + 1);
  wcscpy(dest, s);
  while ( v4 < v6 )
  {
    for ( i = 0; i < v7 && v4 < v6; ++i )
      dest[v4++] -= a2[i];
  }
  return dest;
}
```


稍加分析，可以看出dest其实就是flag。所以在ida里找一下最后一句return dest的地址：



打开gdb，设断点到0x8048702，然后run

```
rock@DESKTOP-610TC9B:~/ctf/1$ gdb -q 555
Reading symbols from 555...(no debugging symbols found)...done.
gdb-peda$ b *0x8048702
Breakpoint 1 at 0x8048702
gdb-peda$ r
Starting program: /mnt/z/Personal/Desktop/ctf-ground/555
Welcome to cyber malware control software.
Currently tracking 779070208 bots worldwide
[----------------------------------registers-----------------------------------]
EAX: 0x804f060 --> 0x39 ('9')
EBX: 0x7d ('}')
ECX: 0x1480
EDX: 0x7d ('}')
ESI: 0xf7fbc000 --> 0x1d7d8c
EDI: 0x0
EBP: 0xffff5268 --> 0xffffd298 --> 0xffffd2b8 --> 0x0
ESP: 0xffff5230 --> 0x804f060 --> 0x39 ('9')
EIP: 0x8048702 (<decrypt+170>:  add    esp,0x34)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[----------------------------------code-------------------------------------]
```


然后x/5sw $eax，看到了flag（大佬写的是6，我喜欢5这个数字）

```
gdb-peda$ x/5sw $eax
0x804f060:      U"9447{you_are_an_international_mystery}"
0x804f0fc:      U""
0x804f100:      U""
0x804f104:      U""
0x804f108:      U""
```


顺便放一下x指令的帮助：

```
gdb-peda$ help x
Examine memory: x/FMT ADDRESS.
ADDRESS is an expression for the memory address to examine.
FMT is a repeat count followed by a format letter and a size letter.
Format letters are o(octal), x(hex), d(decimal), u(unsigned decimal),
  t(binary), f(float), a(address), i(instruction), c(char), s(string)
  and z(hex, zero padded on the left).
Size letters are b(byte), h(halfword), w(word), g(giant, 8 bytes).
The specified number of objects of the specified size are printed
according to the format.  If a negative number is specified, memory is
examined backward from the address.

Defaults for format and size letters are those previously used.
Default count is 1.  Default address is following last thing printed
with this command or "print".
```

题做完了，说一下M1架构mac的坑。为了让arm架构的可以gdb x86程序，我费尽心思，查了无数个网站，但还是没找到方法。最终无奈，还是在windows本上做了这道题。