XCTF新人区逆向Write up合集

原创

yjy1231 于 2021-04-10 11:10:33 发布 89 收藏

XCTF攻防世界

open_source

这道题直接给了源码，读就行了，一开始以为文件名是32位hash，但是计算的话太大，也无法解密，所以突破口还是hash本身的计算，从之前的判断可以推出，其实(second % 17)就是等于8。

算出来还要转为16进制。

logmein

也是考的阅读c的能力，拖到ida里面F5反编译，简单读一下，突破口应该在判断语句里面。

```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
  size_t v3; // rsi
  int i; // [rsp+3Ch] [rbp-54h]
  char s[36]; // [rsp+40h] [rbp-50h]
  int v6; // [rsp+64h] [rbp-2Ch]
  __int64 v7; // [rsp+68h] [rbp-28h]
  char v8[8]; // [rsp+70h] [rbp-20h]
  int v9; // [rsp+8Ch] [rbp-4h]

  v9 = 0;
  strcpy(v8, ":\"AL_RT^L*.?+6/46");
  v7 = 28537194573619560LL;
  v6 = 7;
  printf("Welcome to the RC3 secure password guesser.\n", a2, a3);
  printf("To continue, you must enter the correct password.\n");
  printf("Enter your guess: ");
  __isoc99_scanf("%32s", s);
  v3 = strlen(s);
  if ( v3 < strlen(v8) )
    sub_4007C0();
  for ( i = 0; i < strlen(s); ++i )
  {
    if ( i >= strlen(v8) )
      sub_4007C0();
    if ( s[i] != (char)(*((_BYTE *)&v7 + i % v6) ^ v8[i]) )
      sub_4007C0();
  }
  sub_4007F0();
}
```


处理之后提示数太大了，乍看一眼*(_BYTE *)不太理解什么意思，但是感觉long整数和字符串长度差不多，可以合理推测，可能需要对long型的数据一个字节一个字节的处理。

结果是我看错了，括号其实对应错了，(*((_BYTE *)&v7 + i % v6)，(_BYTE *)&v7是一个整体，实现的功能也是从long中依次取一个字节的数。

首先要知道long在机器中是如何存储的，才知道怎么取出，在汇编中，一般是小端存储，所以按地址取数是从低位开始往高位取，一个字节可以存储两个16进制数，所以一次取两个数，这也解释i % 7 的原因。

最开始想用模运算取数，还停留在10进制中，但是python实现16进制模运算有一个坑就是，会把%识别成格式化的符号，所以这了采用`x>>(8^i)&0xFF`来取数据。

最后16进制结果要转为ascii码。

这道题花了一点时间，我用python处理进制的转换非常的不熟悉，真的头大。

这篇文章中对*(_BYTE *)的解释：

```
This actually has two distinctive parts, evaluated one after the other.

First, (_BYTE *) casts a value or a register to be a byte pointer. This is similar to assigning the value to a C
 variable which is defined as byte *.

Second, * dereferences the address and retrieves the value in that address, value is assumed to be of the type o
f the pointer, in our case _BYTE.
```

拆解出来就是先将值转换为字节指针，之后再找地址对应值。

我觉得自己的解法太麻烦了，最后看了一下其他人的，有的v7反汇编出来就是一个字符串而不是long型，但是这个估计一开始会想不到小端存储，把v7换成字符串处理确实方便很多。

insanity

先用ida看了一下，反编译出来的结果很简单，看一看有没有什么有效的字符信息。直接看到一个像flag的东西，我开始只提交了字符，没加前面数字，还好再尝试提交了一次，不然又掉坑里面了。

getit

ida反编译main函数，感觉核心内容是对"/tmp/flag.txt"文件的写入，fputc做的就是写入操作，另外关注s和t两个参数，从长度来看，感觉又是和logmein一样的套路，但是简单很多，也没什么大小端。

但是总是提交不对，我还以为第二步文件写入打乱顺序了写入，可能要动态调试因为太麻烦了准备看一下到底怎么回事，结果是ida里面字符串只显示了`harifCTF{}`，掉了一个S，ghidra里面可以正常显示出，我不知道为什么，我差点骂出世界上最脏的脏话。不过这道题确实可以动态分析跑出来，更快一点。

python-trade

这一道题是python反编译，`pip install uncompyle`，用uncompyle反编译出来直接是python代码了。

这里有一个点就是，base64解码出来的结果是bytes。

re1

反编译之后看到strcmp，v9和v5，v5是用户输入，突破口就是v9了，顺着找过去是一串16进制数，分成了两段，直接16进制转ascii码看一下，是一串像flag的东西，但是顺序是反的，从程序逻辑来看就是flag，不知道考点是啥。

而且这道题在macos下会乱码。

game

这道题感觉程序比较大，简单用ida看了一下，感觉没什么有效信息，所以还是运行玩一玩游戏，感觉这道题可能用动态分析工具比较好，现在是考虑修改内存把灯全部点亮。

小总结

做了好几道题，简单题基本上还是读C，要熟悉16进制的各种操作，而且本质就是通过加密算法写对应的解密算法。

simple-unpack

给了提示是脱壳相关的，之前电脑里面下了upx，查了一下指令，upx -d 直接脱壳，感觉我 -h不太会看，应该要学一学。

Hello, CTF

这道题，F5反编译之后直接阅读代码，很好理解，就是输入和字符串“437261636b4d654a757374466f7246756e”比较，但是输入字符串长度应该为17，被比较的字符串长度为35，我当时就像，啊，这是溢出吗，没什么想法，然后到处看了看，无意间回到题目描述

> 菜鸡发现Flag似乎并不一定是明文比较的

然后我悟了，16进制转ascii，flag就出来了，怎么说呢，一周多没做题了，感觉脑子就像生锈了一样，不太敏锐了，在思考如果没有题目描述，我解这道题要多久。

之前连做六七道题的时候，对有些套路还是有点感觉的，很快就能get到，到了这周，我就恨自己是块木头。

no-strings-attached

这道题做的太自闭了，看了题目描述，说运行就可以拿flag，所以全心全意想着怎么运行起来，linux下的32位elf文件，好不容易安装32位的lib库，运行还报错，整个人都傻了。

```
sudo apt-get install lib32ncurses5
```

最后还是回归ida，结果就是一个普通套路的加解密，两个数组操作，浪费大量时间想把它跑起来的我深深叹了一口气。

maze

一开始直接IDA静态分析，F5反编译之后，重点看了一下main函数，感觉有点不太对，因为直接靠用户输入进行规则判定，唯一有参考价值的字符串asc_601060看不出什么名堂，感觉静态分析不太可靠。难道是按规则穷举暴力破解吗？后面仔细观察if语句里面的判断条件，看了一下出现的十进制数对应的ascii码（其实是因为读了一半读吐了，为了避免入门到放弃另寻出路。

```
79 - O
111 - o
46 - .
48 - 0
35 - #
```

参考了一下IDA动态调试的博客

把几个判断跳转的地方，特别是跳转到label_22要退出的地方下了断点，试一试`nctf{00000000000000000}`，发现可以单步。弄了半天无发现，倒是某一篇博客里面提到，想要练手可以试一试xctf的track2，这就意味着，这道题应该不是动态调试。

又回到main函数，但是判断语句太多，括号什么的不好看，装了一个高亮括号的hexlight插件，但是感觉使用感一般。

之后就是分析-推翻-分析-推翻，搞到自闭的循环，最后回到题目描 述，菜鸡走出迷宫s m de，结合asc_601060，一共64位，那么弄成8*8矩阵看看什么情况，就真•迷宫呗，不过‘maze’这个词本身也有迷宫的意思，哎，这就是英语不好的代价吗。

```
..******
*...*..*
***.*.**
**..*.**
*..*#..*
**.***.*
**.....*
********
```

那么['0','o','O','.']也就是四个方向，走出迷宫的路径就是flag，开始在分析v9的时候（ida一开始显示成v10，类型是__int64），总是不知道到底是个什么，SHIDWORD之前也没见过，推测是不是计数器，但是知道是迷宫之后，猜测可能是二维数组。

查资料之后发现这个issue里面有详细解释，SHIDWORD其实就是取v9的高位，存储的是列，v9低位是行，这个从最后一句判断是不是找到出口了可以看出来。

```
if ( asc_601060[8 * (signed int)v9 + SHIDWORD(v9)] != 35 )
```

那么再看和四个方向相关的函数，加法的就是□，□，减法的就是□，□；再看具体传入函数的参数，`(_DWORD *)&v9 + 1`和`(int *)&v9`，传入的是地址，+1也比较好理解，因为小端存储，高位放在高地址里面，那么显然`&v9 + 1`就是`SHIDWORD(v9)`。加上自减操作，所以O对应的就是左，以此类推。

```
0  o  . 0
左 右 上 下
```

个人觉得这道题一如以上来就莽的话蛮搞心态的，但是如果结合题目和描述各方面信息+经验多对这些敏感的话就其实简单。断断续续做了好几天，差点就做不下去了。不过也发现自己指针什么的全部忘光了，只是还是要经常用才记得住。

csaw2013reversing2

最后一题，目标就是吃午饭之前做完，再多时间就不做了，运行发现输出乱码，题目描述也是“听说运行就能拿到Flag，不过菜鸡运行的结果不知道为什么是乱码”。

先用ida简单看一下整个程序。IsDebuggerPresent() 第一直觉就是这个函数是反调试的相关的，后面查了一下，的确是。那就分析一下乱码的可能原因。推测sub_401000()函数就是一个还原flag的函数，但是因为直接退出了，所以没办法输出flag。

我最开始的想法就是通过Ollydbg动态调试，然后在他ExitProcess之前获取flag？主要在IsDebuggerPresent()和ExitProcess()下了断点，但是半天找不到字符串存储的位置，真的流泪了，Ollydbg我真的很不熟练，还是准备看源码。

重点关注sub_401000()函数。

```
unsigned int __fastcall sub_401000(int a1, int a2)
{
  int v2; // esi
  unsigned int v3; // eax
  unsigned int v4; // ecx
  unsigned int result; // eax

  v2 = dword_409B38;
  v3 = a2 + 1 + strlen((const char *)(a2 + 1)) + 1;
  v4 = 0;
  result = ((v3 - (a2 + 2)) >> 2) + 1;
  if ( result )
  {
    do
      *(_DWORD *)(a2 + 4 * v4++) ^= v2;
    while ( v4 < result );
  }
```

因为我不想花太多时间在这道题上面了，我个人觉得花太多时间做简单题，掉到一些坑里面，消磨斗志，真的得不偿失（来自maze的血的教训）。重点关注循环，循环里面做的就是一次取四个（因为是DWORD）然后依次异或。

result是循环次数，按理来说应该先分析v3，但是a2又是int型，又是const char *有点迷惑了，直接看，简化成`strlen((const char *)(a2 + 1)/4 +1`，这里大胆假设应该就是9次，一轮，所有的字符都异或一次。

解密的算法比较简单，直接依次和`0DDCCAABBh`抑或就完事了，但是发现结果居然不太对，我想该不会是result次数不对吧，但是后来发现，又是小段存储的问题，`0DDCCAABBh`异或的时候应该是`arr = [0xBB,0xAA,0xCC,0xDD]`，然后得到flag。

后来想看看这道题动态调试的解法，结果看到一个包含3种解法的wp，这就是我和别人的差距吗。准备写完都试一试，真的不错。

小结

这段时间，也算是磕磕绊绊、断断续续把xctf的新手逆向题做完了，我真的属于对逆向一窍不通的了。但是感觉还是有一些小技巧。

(1) 全局搜索字符串，感觉最简单的签到题应该就是这个水平

(2) 静态分析IDA F5反汇编，读C代码的能力

(3) 多关注字符串比较的地方，名字类似decrypt的就很有可能是对flag进行简单的加解密

(4) python要熟练使用ord()和chr()

(5) 16进制与ASCII码，可以敏感一点

(6) 知道小端存储

(7) 动态调试多注意jmp、call之类的跳转指令

(8) 脱壳目前只做了一个很简单的相关的，一般好像第一步就是先看程序有没有壳，我现在的知识也仅仅止步于upx -d

(9) ida的python终端做题的时候还是非常有用的，目前用的还停留在dump一些比较关键的数据，下面这个在csaw2013reversing2题目里面用到过的，应该没太大问题，有时候可能需要提的数据类型不太一样

```
```
adrr = 0x409B10
arr =[]
for i in range(36):
    arr.append(idc.Byte(adrr+i))
print(arr)
```
```

(10) 虽然勉勉强强的做完了XCTF里面逆向的简单题，但是感觉还是差很远，一个是有时候做题速度太慢了，二是相关软件使用非常不熟练，三是动态调试能力基本没有，还有很长的路要走，继续加油。

不知道为什么搬过来有序列表格式奇奇怪怪，只能用括号了。