[GWCTF 2019]xxor-buuctf

原创

Le0nard404 于 2022-03-14 18:25:09 发布 4607 收藏

分类专栏： CTF每日做题 文章标签： 反编译 信息安全 安全

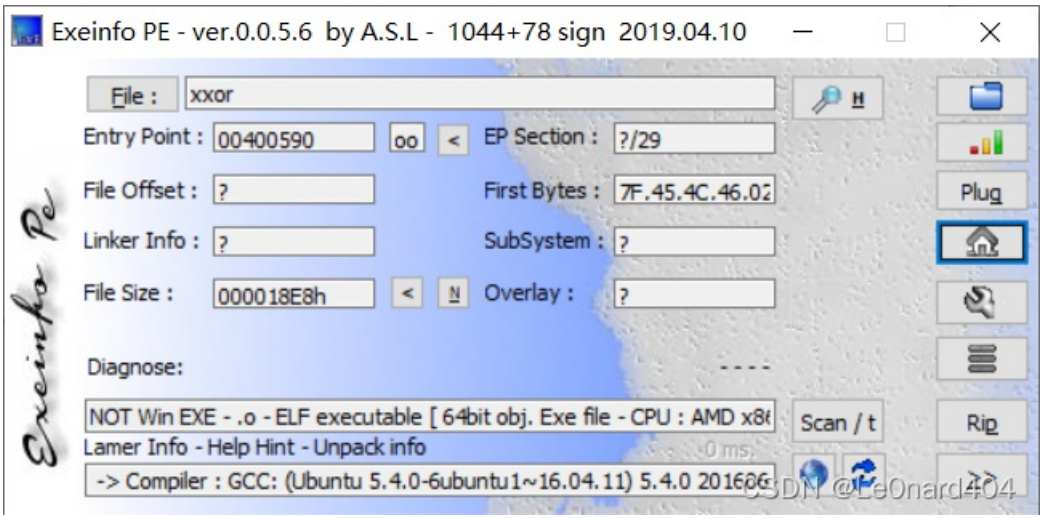
本文链接：https://blog.csdn.net/Lenard404/article/details/123480584

CTF每日做题 专栏收录该内容

8 篇文章 0 订阅

订阅专栏

buuctf做题记录



查壳

64位无壳

分析

IDA反编译main函数

```
puts("Let us play a game?");
puts("you have six chances to input");
puts("Come on!");
v6[0] = 0LL;
v6[1] = 0LL;
v6[2] = 0LL;
v6[3] = 0LL;
v6[4] = 0LL;
for ( i = 0; i <= 5; ++i )
{
  printf("%s", "input: ");
  a2 = (char **)((char *)v6 + 4 * i);
  __isoc99_scanf("%d", a2);
}
v7[0] = 0LL;
v7[1] = 0LL;
```

```
v7[2] = 0LL;
v7[3] = 0LL;
v7[4] = 0LL;
for ( j = 0; j <= 2; ++j )
{
  dword_601078 = v6[j];
  dword_60107C = HIDWORD(v6[j]);
  a2 = (char **)&unk_601060;
  sub_400686(&dword_601078, &unk_601060);
  LODWORD(v7[j]) = dword_601078;
  HIDWORD(v7[j]) = dword_60107C;
}
if ( (unsigned int)sub_400770(v7, a2) != 1 )
{
  puts("NO NO NO~ ");
  exit(0);
}
puts("Congratulation!\n");
```


结构还是很清晰的，`sub_400686`是加密函数，`sub_400770`是对加密后字符串进行比较，即验证函数。

查看**sub_400770**

```
__int64 __fastcall sub_400770(_DWORD *a1)
{
  __int64 result; // rax

  if ( a1[2] - a1[3] == 0x84A236FFLL
    && a1[3] + a1[4] == 0xFA6CB703LL
    && a1[2] - a1[4] == 0x42D731A8LL
    && *a1 == 0xDF48EF7E
    && a1[5] == 0x84F30420
    && a1[1] == 0x20CAACF4 )
  {
    puts("good!");
    result = 1LL;
  }
  else
  {
    puts("Wrong!");
    result = 0LL;
  }
  return result;
}
```


符合条件就返回1，用z3进行一个求解：

```
from z3 import *
s = IntVector('s',6)
solver = Solver()
solver.add(s[0]== 0xdf48ef7e)
solver.add(s[2]-s[3] == 0x84a236ff)
solver.add(s[3]+s[4]== 0xfa6cb703)
solver.add(s[2]-s[4] == 0x42d731a8)
solver.add(s[5] == 0x84f30420)
solver.add(s[1]== 0x20caacf4)


flag = []
if solver.check() == sat:
    m = solver.model()
    for i in s:
        flag.append(m[i])
    print(flag)
f = ''
```

得到解：

```
[3746099070,
 550153460,
3774025685,
1548802262,
2652626477,
2230518816]
```

这就是经过加密后我们应该得到的字符串，需要注意的是他们**非常大**。

查看sub_400686函数

```
__int64 __fastcall sub_400686(unsigned int *a1, _DWORD *a2)
{
  __int64 result; // rax
  unsigned int v3; // [rsp+1Ch] [rbp-24h]
  unsigned int v4; // [rsp+20h] [rbp-20h]
  int v5; // [rsp+24h] [rbp-1Ch]
  unsigned int i; // [rsp+28h] [rbp-18h]

  v3 = *a1;
  v4 = a1[1];
  v5 = 0;
  for ( i = 0; i <= 0x3F; ++i )
  {
    v5 += 1166789954;                                   // 二进制
    v3 += (v4 + v5 + 11) ^ ((v4 << 6) + *a2) ^ ((v4 >> 9) + a2[1]) ^ 0x20;
    v4 += (v3 + v5 + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 0x10;
  }
  *a1 = v3;
  result = v4;
  a1[1] = v4;
  return result;
}
```


逻辑很简单，逆向过来先处理v4,然后v3,最后v5即可。

先查看a2数组是什么：

```
data:0000000000601060 unk_601060      db    2
data:0000000000601061                 db    0
data:0000000000601062                 db    0
data:0000000000601063                 db    0
data:0000000000601064                 db    2
data:0000000000601065                 db    0
data:0000000000601066                 db    0
data:0000000000601067                 db    0
data:0000000000601068                 db    3
data:0000000000601069                 db    0
data:000000000060106A                 db    0
data:000000000060106B                 db    0
data:000000000060106C                 db    4
data:000000000060106D                 db    0
data:000000000060106E                 db    0
data:000000000060106F                 db    0
data:000000000060106F _data           ends
```


写脚本这里注意，需要创建的是`_int64`数组。

> Int64是有符号 64 位整数数据类型，相当于C++中的long long、 C# 中的 long 和 SQL Server 中的 bigint，表示值介于 -2^63 (-9,223,372,036,854,775,808) 到2^63-1(+9,223,372,036,854,775,807)之间的整数。

```
#include <iostream>
using namespace std;

int main()
{
    __int64 a[6] = { 3746099070, 550153460, 3774025685, 1548802262, 2652626477, 2230518816 };
    unsigned int a2[4] = { 2,2,3,4 };
    unsigned int v3, v4;
    int v5;
    for (int j = 0; j <= 4; j += 2) {
        v3 = a[j];
        v4 = a[j + 1];
        v5 = 1166789954*0x40;
        for (int i = 0; i <= 0x3F; ++i) {
            v4 -= (v3 + v5 + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 0x10;
            v3 -= (v4 + v5 + 11) ^ ((v4 << 6) + *a2) ^ ((v4 >> 9) + a2[1]) ^ 0x20;
            v5 -= 1166789954;
        }
        a[j] = v3;
        a[j + 1] = v4;
    }
    //小端排序
    for (int i = 0; i < 6; ++i) {
        cout << *((char*)&a[i] + 2) << *((char*)&a[i] + 1) <<  * ((char*)&a[i]);
    }
    return 0;
}
```

得到flag

`flag{re_is_great!}`

小记

输出关于高16位和低16位，`hiword\loword`，需要补充知识，复建。