攻防世界 reverse 进阶 10 Reverse Box

转载

deyou0823 于 2019-09-01 12:31:00 发布 223 收藏
文章标签： python
原文链接：http://www.cnblogs.com/DirWang/p/11441963.html

攻防世界中此题信息未给全，题目来源为[TWCTF-2016：Reverse] Reverse Box

网上有很多wp是使用gdb脚本，这里找到一个本地还原关键算法，然后再爆破的

https://www.megabeets.net/twctf-2016-reverse-reverse-box/

[TWCTF-2016：Reverse] Reverse Box Writeup

标准

Shak的客座文章。

> **挑战描述**
> $ ./reverse_box $ {FLAG}
> 95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4a re

---

这个挑战是一个二进制文件，它需要一个参数然后吐出一个字符串。我们得到二进制文件的输出，用于运行它。让我们开始这一点。

```
./reverse_box 0000
28282828
```

二进制可能会打印替换每个字符的十六进制值。同样清楚的是，它是每个角色的唯一值。我们必须处理某种替代密码。在使用完全相同的参数再次运行它之后，我们得到不同的输出，因此替换也以某种方式随机化。

跳转到二进制文件，我们有两个重要的函数，我命名为main和calculate。

From Hex-Rays Decompiler

C

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@7
  int v4; // ecx@7
  size_t i; // [sp+18h] [bp-10Ch]@4
  int v6; // [sp+1Ch] [bp-108h]@4
  int v7; // [sp+11Ch] [bp-8h]@1

  v7 = *MK_FP(__GS__, 20);
  if ( argc <= 1 )
  {
    printf("usage: %s flag\n", *argv);
    exit(1);
  }
  calculate((int)&v6);
  for ( i = 0; i < strlen(argv[1]); ++i )
    printf("%02x", *((_BYTE *)&v6 + argv[1][i]));
  putchar('\n');
  result = 0;
  v4 = *MK_FP(__GS__, 20) ^ v7;
  return result;
}
```

主要功能非常简单。它负责检查我们是否使用参数运行二进制文件，调用calculate函数，最后打印结果。查看结果打印格式，我们可以看到结果是十六进制格式。

接下来我们将处理计算功能。

From Hex-Rays Decompiler

C

```
int __cdecl calculate(int a1)
{
  unsigned int seed; // eax@1
  int v2; // edx@4
  char v3; // al@5
  char v4; // ST1B_1@7
  char v5; // al@8
  int v6; // eax@10
  int v7; // ecx@10
  int v8; // eax@10
  int v9; // ecx@10
  int v10; // eax@10
  int v11; // ecx@10
  int v12; // eax@10
  int v13; // ecx@10
  int result; // eax@10
  char v15; // [sp+1Ah] [bp-Eh]@3
  char v16; // [sp+1Bh] [bp-Dh]@3
  char v17; // [sp+1Bh] [bp-Dh]@7
  int randomNum; // [sp+1Ch] [bp-Ch]@2

```

```
22    seed = time(0);
23    srand(seed);
24    do
25      randomNum = (unsigned __int8)rand();
26    while ( !randomNum );
27    *(_BYTE *)a1 = randomNum;
28    v15 = 1;
29    v16 = 1;
30    do
31    {
32      v2 = (unsigned __int8)v15 ^ 2 * (unsigned __int8)v15;
33      if ( v15 >= 0 )
34        v3 = 0;
35      else
36        v3 = 27;
37      v15 = v2 ^ v3;
38      v4 = 4 * (2 * v16 ^ v16) ^ 2 * v16 ^ v16;
39      v17 = 16 * v4 ^ v4;
40      if ( v17 >= 0 )
41        v5 = 0;
42      else
43        v5 = 9;
44      v16 = v17 ^ v5;
45      v6 = *(_BYTE *)a1;
46      LOBYTE(v6) = v16 ^ v6;
47      v7 = (unsigned __int8)v16;
48      LOBYTE(v7) = __ROR1__(v16, 7);
49      v8 = v7 ^ v6;
50      v9 = (unsigned __int8)v16;
51      LOBYTE(v9) = __ROR1__(v16, 6);
52      v10 = v9 ^ v8;
53      v11 = (unsigned __int8)v16;
54      LOBYTE(v11) = __ROR1__(v16, 5);
55      v12 = v11 ^ v10;
56      v13 = (unsigned __int8)v16;
57      LOBYTE(v13) = __ROR1__(v16, 4);
58      result = v13 ^ v12;
59      *(_BYTE *)(a1 + (unsigned __int8)v15) = result;
60    }
61    while ( v15 != 1 );
62    return result;
63  }
```

它根本不像我们的主要直接，但基本上它随机化一个变量，做一些内存操作并返回一些随机值。现在，让我们尝试调试二进制文件而不完全理解计算。

二进制文件迭代输入中的字符并执行以下程序集以打印结果。

```
1  movzx   eax, byte ptr [esp+eax+1Ch]
2  movzx   eax, al
3  mov     [esp+4], eax
4  mov     dword ptr [esp], offset a02x ; "%02x"
5  call    _printf
```

到达这组指令后，eax寄存器保存了我们输入的字符。然后为printf函数传递的是来自eax位置（第1行）的[esp + 1Ch]中的数组的元素。所以这个数组保存我们的结果。让我们进一步研究它，看看它是什么。跳转到堆栈上的那个位置我们会遇到一个268 Bytes的十六进制数组，我们将把数组保存到一个名为*array*的二进制文件中供以后使用。

在运行二进制文件几次之后，我们可以理解该数组也是以某种方式随机化的。也许它与计算函数返回的随机值有关。再运行二进制文件几次，我们看到数组的第二个元素总是等于计算中的随机数。所以，二进制文件必须有一个基本数组，然后用随机值以某种方式进行操作。如果每次操作发生时，基本数组中的第二个元素必须为零，我们得到随机数。让我们试着理解二进制执行什么样的操作。我们将通过将二进制数组加载到python脚本来实现这一点，然后我们将通过对我们保存的数组元素执行假定操作来获取基数组，并使用我们执行二进制文件时的已知随机值，最后将该过程与另一个具有不同随机值的二进制运行进行比较以进行验证。第一个选项是将随机值添加到基础数组值。它适用于第二个元素，但将此过程与另一个二进制运行进行比较不起作用。第二个猜测是二进制文件使用基本数组元素对随机值进行异或运算，这也将允许第二个数组元素等于随机值。宾果，这个有效。现在我们所要做的就是使用我们保存的数组和该二进制运行的随机值来获取基数组。接下来我们要考虑的是标志结构，即TWCTF {...}。

```
#! /usr/bin/python
f = open(r"c:\megabeets\array", "rb")
buff = f.read(268)
random_value = 0x66
base_array = []

#Claculate the base array
for i in buff:
base_array.append(ord(i) ^ random value)

#The given output from the flag run, each value is seperated by -
flag_output = "95-ee-af-95-ef-94-23-49-99-58-2f-72-2f-49-2f-72-b1-9a-7a-af-72-e6-e7-76-b5-7a-ee-72-2f-e7-7a-b5-ad-9a-ae-b1-56-72-96-76-ae-7a-23-6d-99-b1-df-4a"
flag = ''
for c in flag_output:
flag.append(int('0x' + c , 16))

T_location = ord('T')
#XORing the T_location element in the base array with the output result in order to get the random value
flag_random_value = base_array[T_location] ^ flag[0]

#Manipulate the base array to create the array for the flag binary run
flag_array = []
for b in base_array:
flag_array.append(ord(b) ^ flag_random_value)

# Create a dictionary which maps an output hex value to an input character
dic{}
for i in range(0, 268):
dic[flag_array[i]] = chr(i)

#
for i in xrange(len(flag)):
print dic[flag[i]],
```

该flag为 ***TWCTF {5UBS717U710N_C1PH3R_W17H_R4ND0M123D_5-B0X}***。

python3

⊞ ⊟

```
ror = lambda val, r_bits, max_bits: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))

all_s_boxes = {}
for randomValue in range(256):
  results = [0 for i in range(256)]
  position_index = 3;
  some_value = 1;
  while position_index != 1:
    # Used to generate some_value
    v1 = 2 * some_value ^ some_value & 0xff
    v2 = (v1 ^ v1 * 4) &0xff
    v3 = (v2 ^ v2 * 16) & 0xff
    if ( v3 < 128 ):
      v5 = 0;
    else:
      v5 = 9;
    some_value = (v3 ^ v5) & 0xff;

    factor1 = some_value & 0xff
    factor2 = some_value ^ randomValue
    factor1_rotate_7bits = ror(factor1, 7,8)
    factor1_rotate_6bits = ror(factor1, 6,8)
    factor1_rotate_5bits = ror(factor1, 5,8)
    factor1_rotate_4bits = ror(factor1,4,8)
    result = factor2 ^ factor1_rotate_7bits ^ factor1_rotate_6bits ^ factor1_rotate_5bits ^
factor1_rotate_4bits

    results[position_index] = (result & 0xff)

    some_value = factor1
    v2 = (position_index ^ (2 * position_index));
    if position_index < 128:
      v3 = 0;
    else:
      v3 = 27;
    position_index = (v2 ^ v3) & 0xff
  all_s_boxes[randomValue] = results

startingFlag = "TWCTF"
startingTarget = "\x95\xee\xaf\x95\xef"
sbox_to_use = None

for key,sbox in all_s_boxes.items():
  for index,char in enumerate(startingFlag):
    if sbox[ord(char)] == ord(startingTarget[index]):
      sbox_to_use = sbox
      print("Found sbox at random value %d" % key)
      break

target =
"95eeaf95ef94234999582f722f492f72b19a7aaf72e6e776b57aee722fe77ab5ad9aaeb156729676ae7a236d99b1df4a"
print(''.join([chr(sbox_to_use.index(i)) for i in bytes.fromhex(target)]))
```

View Code

转载于:https://www.cnblogs.com/DirWang/p/11441963.html