Writeup of x64Lotto(reverse) in reversing.kr

原创

C0ss4ck 于 2018-01-26 23:44:01 发布 266 收藏
分类专栏： Reverse_of_CTF 文章标签： CTF reverse PRNG

此题风格诡异，有一种野生逆向的既视感（疯狂改跳转）

不扯别的，先下载附件。得到lotto.exe，无壳。接下来开始正式分析。

0x00 反编译代码逻辑

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // eax@1
  signed __int64 i; // rbx@2
  char v5; // r8@4
  int r; // edx@4
  signed __int64 k; // rcx@4
  _BYTE *v8; // rdx@10
  signed __int64 j; // rcx@10
  char v10; // al@11
  signed int d; // ecx@13
  __int16 *v12; // rdx@13
  __int16 v13; // ax@14
  __int16 v14; // ax@14
  int input; // [sp+40h] [bp-78h]@1
  int v17; // [sp+44h] [bp-74h]@1
  int v18; // [sp+48h] [bp-70h]@1
  int v19; // [sp+4Ch] [bp-6Ch]@1
  int v20; // [sp+50h] [bp-68h]@1
  int v21; // [sp+54h] [bp-64h]@1
  int newkey1; // [sp+58h] [bp-60h]@1
  int v23; // [sp+5Ch] [bp-5Ch]@1
  int v24; // [sp+60h] [bp-58h]@1
  int v25; // [sp+64h] [bp-54h]@1
  int v26; // [sp+68h] [bp-50h]@1
  int v27; // [sp+6Ch] [bp-4Ch]@1
  __int16 v28; // [sp+70h] [bp-48h]@10
  __int16 v29; // [sp+72h] [bp-46h]@10
  __int16 v30; // [sp+74h] [bp-44h]@10
  __int16 v31; // [sp+76h] [bp-42h]@10
  __int16 v32; // [sp+78h] [bp-40h]@10
  __int16 v33; // [sp+7Ah] [bp-3Eh]@10
  __int16 v34; // [sp+7Ch] [bp-3Ch]@10
  __int16 v35; // [sp+7Eh] [bp-3Ah]@10
  __int16 v36; // [sp+80h] [bp-38h]@10
  __int16 v37; // [sp+82h] [bp-36h]@10
  __int16 v38; // [sp+84h] [bp-34h]@10
```

```
  __int16 v39; // [sp+86h] [bp-32h]@10
  __int16 v40; // [sp+88h] [bp-30h]@10
  __int16 v41; // [sp+8Ah] [bp-2Eh]@10
  __int16 v42; // [sp+8Ch] [bp-2Ch]@10
  __int16 v43; // [sp+8Eh] [bp-2Ah]@10
  __int16 v44; // [sp+90h] [bp-28h]@10
  __int16 v45; // [sp+92h] [bp-26h]@10
  __int16 v46; // [sp+94h] [bp-24h]@10
  __int16 v47; // [sp+96h] [bp-22h]@10
  __int16 v48; // [sp+98h] [bp-20h]@10
  __int16 v49; // [sp+9Ah] [bp-1Eh]@10
  __int16 v50; // [sp+9Ch] [bp-1Ch]@10
  __int16 v51; // [sp+9Eh] [bp-1Ah]@10
  __int16 v52; // [sp+A0h] [bp-18h]@10
  __int16 v53; // [sp+A2h] [bp-16h]@10

  input = 0;
  v17 = 0;
  v18 = 0;
  v19 = 0;
  v20 = 0;
  v21 = 0;
  newkey1 = 0;
  v23 = 0;
  v24 = 0;
  v25 = 0;
  v26 = 0;
  v27 = 0;
  v3 = time64(0i64);
  srand(v3);
  do
  {
    wprintf(L"\n\t\tL O T T O\t\t\n\n");
    wprintf(L"Input the number: ");
    wscanf_s(L"%d %d %d %d %d %d", &input, &v17, &v18);
    wsystem(L"cls");
    Sleep(0x1F4u);
    i = 0i64;
    do
      *(&v21 + ++i) = rand() % 100;
    while ( i < 6 );
    v5 = 1;
    r = 0;
    k = 0i64;
    byte_7FF7E67335F0 = 1;
    while ( *(&newkey1 + k * 4) == *(&input + k * 4) )
    {
      ++k;
      ++r;
      if ( k >= 6 )
        goto LABEL_9;
    }
    v5 = 0;
    byte_7FF7E67335F0 = 0;
LABEL_9:
    ;
  }
  while ( r != 6 );
  v8 = key;
```

```
  v29 = 92;
  v28 = 184;
  v30 = 139;
  v33 = 184;
  v31 = 107;
  j = 0i64;
  v32 = 66;
  v34 = 56;
  v35 = 237;
  v36 = 219;
  v37 = 91;
  v38 = 129;
  v39 = 41;
  v40 = 160;
  v41 = 126;
  v42 = 80;
  v43 = 140;
  v44 = 27;
  v45 = 134;
  v46 = 245;
  v47 = 2;
  v48 = 85;
  v49 = 33;
  v50 = 12;
  v51 = 14;
  v52 = 242;
  v53 = 0;
  do
  {
    v10 = key[j - 1];
    j += 5i64;
    *(&v25 + j + 1) ^= (v10 - 12);
    *(&v26 + j) ^= (key[j - 5] - 12);
    *(&v26 + j + 1) ^= (key[j - 4] - 12);
    *(&v27 + j) ^= (key[j - 3] - 12);
    *(&v27 + j + 1) ^= (key[j - 2] - 12);
  }
  while ( j < 25 );
  if ( v5 )
  {
    d = 0;
    v12 = &v28;
    do
    {
      v13 = *v12;
      ++v12;
      v14 = d++ + (v13 ^ 0xF);
      *(v12 - 1) = v14;
    }
    while ( d < 25 );
    v53 = 0;
    wprintf(L"%s\n", &v28);
  }
  wprintf(L"\n", v8);
  return 1;
}
```

仔细观察，发现代码逻辑应该是：

1.输入6个整数

2.和伪随机数列前六位进行校验

3.校验成功后进入下一轮对伪随机数列的运算，得到输出结果；校验失败则无输出

0x01 动态调试

平时在Windows下调试都是针对32位程序的调试，可是这个程序是64位的，怎么操作呢？

（划重点！敲黑板！）

其实原理和远程调试elf文件一样——首先启动dbgsrv目录下的win64_remotex64.exe，显示：

```
IDA Windows 64-bit remote debug server(MT) v1.19. Hex-Rays (c) 2004-2015
Host DESKTOP-AMVLUJ3 (192.168.211.1): Listening on port #23946...
```

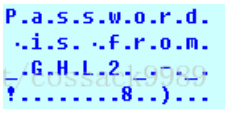
然后选择Remote Windows Debugger，并且Set as default debugger。之后进入process options，将hostname设置为127.0.0.1。

此处推荐一篇文章：**IDA debug x64 programs**

之后就可以开始正常远程动态调试了。

老套路，碰到这种校验相对单一的，只需要Patch跳转指令，一路跳到最终输出前的运算代码即可。

最后输出结果前在rdx寄存器中发现：



随后屏幕输出

Password is from_GHL2_-_!
大功告成