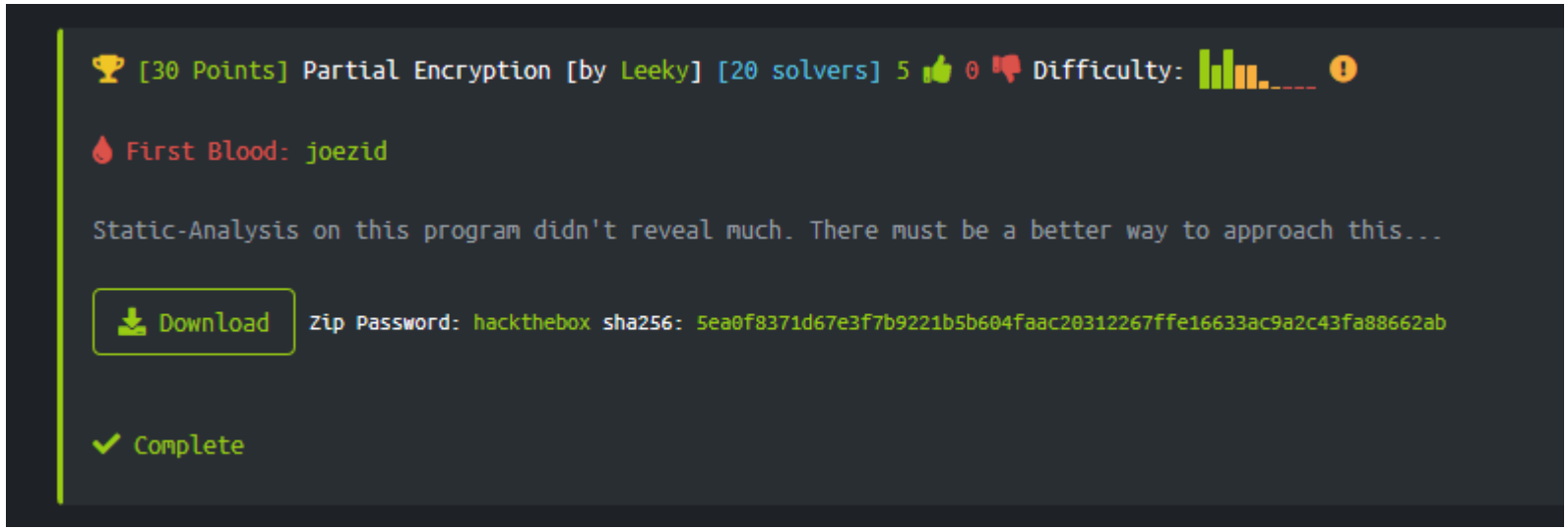
PartialDecryption – HackTheBox Reverse challenge



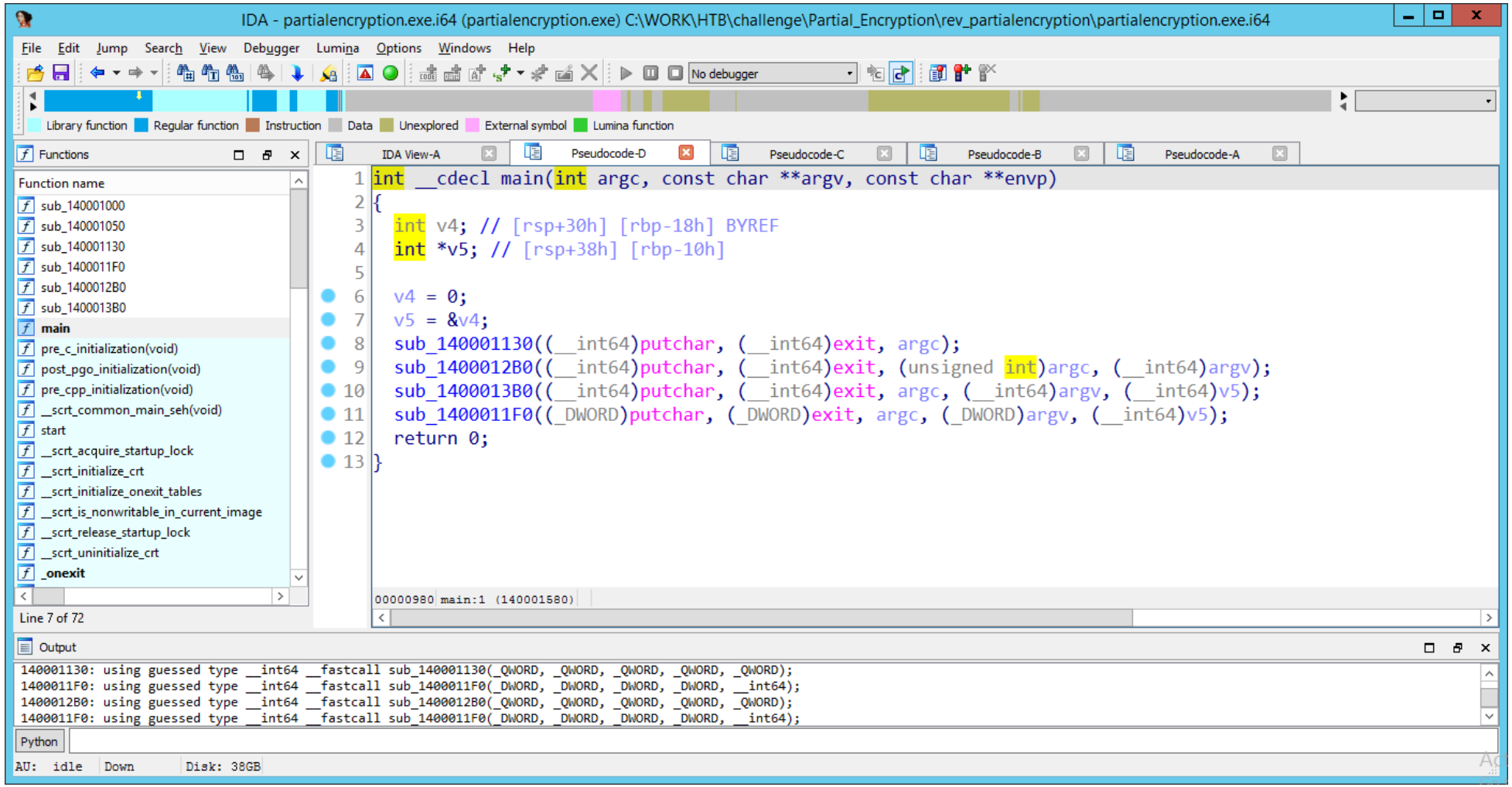
https://www.hackthebox.com/home/challenges/Reversing

If we run the sample, we see usage string

```
>partialencryption.exe
./chal <flag>
```

In case providing some input get “Nope” as answer.

Open the sample in IDA and decompile it



Function Main has 4 functions:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+30h] [rbp-18h] BYREF
  int *v5; // [rsp+38h] [rbp-10h]

  v4 = 0;
  v5 = &v4;
  sub_140001130((__int64)putchar, (__int64)exit, argc); // 1st function check argc
  sub_1400012B0((__int64)putchar, (__int64)exit, (unsigned int)argc, (__int64)argv); // do smth
  sub_1400013B0((__int64)putchar, (__int64)exit, argc, (__int64)argv, (__int64)v5); // print smth
  sub_1400011F0((_DWORD)putchar, (_DWORD)exit, argc, (_DWORD)argv, (__int64)v5);   // free memory
  return 0;
}
```

The first function just if argc != 2 print usage and exit:

```
void __fastcall sub_140001130(__int64 a1, __int64 a2, int a3)
{
  void (__fastcall *v3)(__int64, __int64); // [rsp+30h] [rbp-18h]
  void (__fastcall *v4)(__int64, __int64); // [rsp+38h] [rbp-10h]

  if ( a3 != 2 )
  {
    v3 = (void (__fastcall *)(__int64, __int64))sub_140001050((__int64)&unk_140004000, 112);
    v3(a1, a2);
    VirtualFree(v3, 0i64, 0x8000u);
    v4 = (void (__fastcall *)(__int64, __int64))sub_140001050((__int64)&unk_140004110, 48);
    v4(a1, a2);
    VirtualFree(v4, 0i64, 0x8000u);
  }
}
```

The second function does the cycle, 22 times:

```
void __fastcall sub_1400012B0(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  int i; // [rsp+20h] [rbp-38h]
  void (__fastcall *v5)(__int64, __int64); // [rsp+38h] [rbp-20h]
  void (__fastcall *v6)(__int64, __int64); // [rsp+40h] [rbp-18h]

  for ( i = 0; i < 22; ++i )
  {
    if ( !*(_BYTE *)(*(_QWORD *)(a4 + 8) + i) )
    {
      v5 = (void (__fastcall *)(__int64, __int64))sub_140001050((__int64)&unk_140004070, 64);
      v5(a1, a2);
      VirtualFree(v5, 0i64, 0x8000u);
      v6 = (void (__fastcall *)(__int64, __int64))sub_140001050((__int64)&unk_140004110, 48);
      v6(a1, a2);
      VirtualFree(v6, 0i64, 0x8000u);
    }
  }
}
```

The third function do some output and the fourth function just free allocated memory

Let consider the second function closer

```
LPVOID __fastcall sub_140001050(__int64 a1, int a2)
{
  __m128i v2; // xmm0
  __m128i v3; // xmm0
  __m128i v4; // xmm0
  int i; // [rsp+20h] [rbp-38h]
  DWORD flOldProtect; // [rsp+24h] [rbp-34h] BYREF
  LPVOID lpAddress; // [rsp+28h] [rbp-30h]
  __m128i v9; // [rsp+30h] [rbp-28h] BYREF
  __m128i v10; // [rsp+40h] [rbp-18h] BYREF

  lpAddress = VirtualAlloc(0i64, a2, 0x1000u, 4u);
  for ( i = 0; i < a2 / 16; ++i )
  {
    v2 = _mm_cvtsi32_si128((char)i);
    v3 = _mm_unpacklo_epi8(v2, v2);
    v9 = _mm_shuffle_epi32(_mm_unpacklo_epi16(v3, v3), 0);
    v4 = _mm_loadu_si128((const __m128i *)(a1 + 16i64 * i));
    v10 = v4;
    *(double *)v4.m128i_i64 = sub_140001000(&v10, &v9);
    *((__m128i *)lpAddress + i) = v4;
  }
  VirtualProtect(lpAddress, a2, 0x20u, &flOldProtect);
  return lpAddress;
}
```

It allocates memory and do something there

Let us check what is happening there closer

Obviously, it is some AES operations.

```
__m128 __fastcall sub_140001000(const __m128i *a1, const __m128i *a2)
{
  _XMM0 = _mm_loadu_si128(a2);
  __asm { aeskeygenassist xmm0, xmm0, 0 }
  _XMM1 = _mm_loadu_si128(a2);
  __asm { aeskeygenassist xmm1, xmm1, 10h }
  _XMM1 = _mm_xor_si128(_mm_loadu_si128(a1), _XMM1);
  __asm { aesdeclast xmm1, xmm0 }
  return (__m128)_XMM1;
}
```

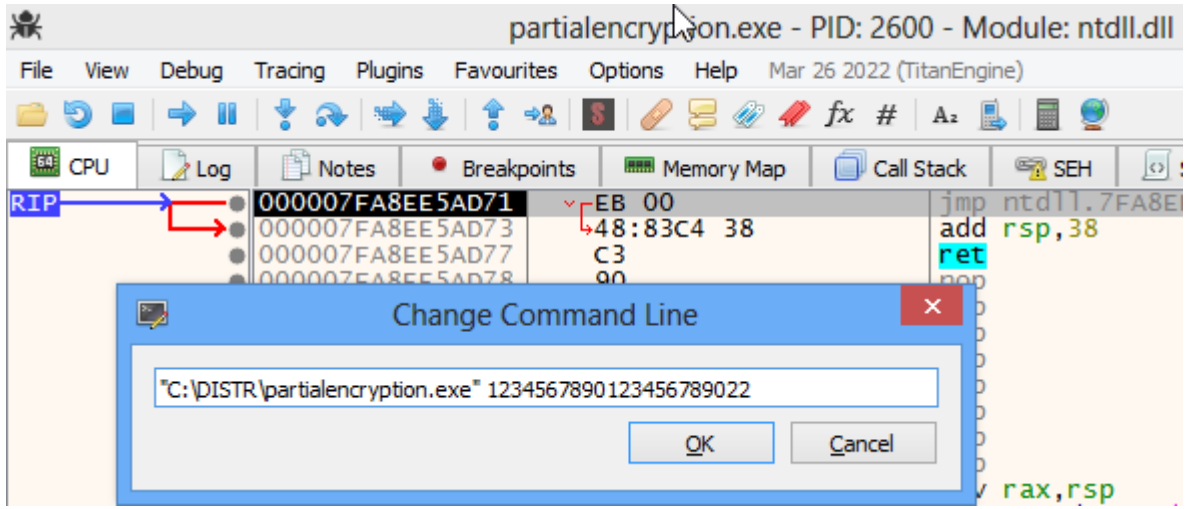
We consider 22 chars is the flag length and operations in allocated memory are decryption and comparing with the provided input as flag.

In case wrong flag we have Nope printed, in case correct flag we may expect some positive answer. We do not need the answer we need the flag.

To get the flag we can reverse encryption or just set break point on VirtualAlloc and VirtualFree.

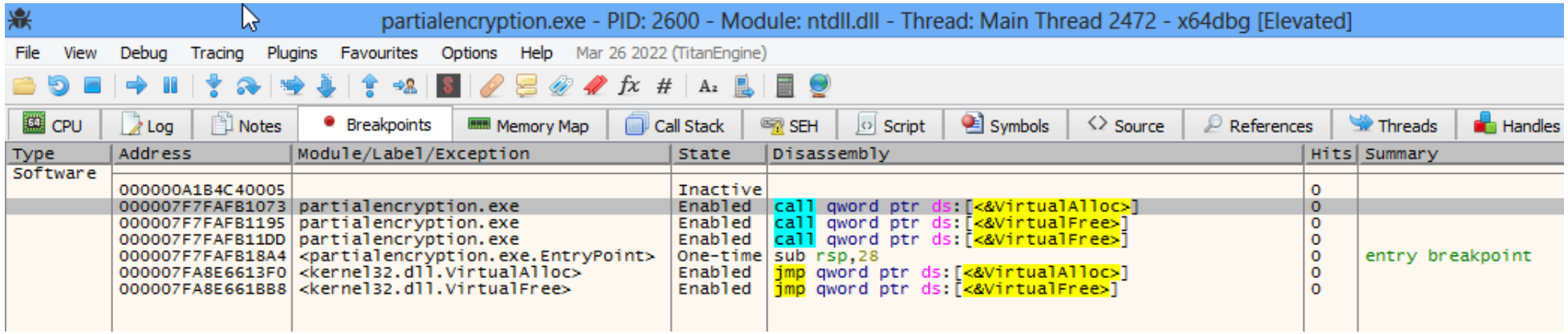
When hit VirtualAlloc we can notice the memory address for allocated area. When hit VirtualFree we can grab the content.

Open the sample in x64dbg and set Command line to have 22 chars input as flag

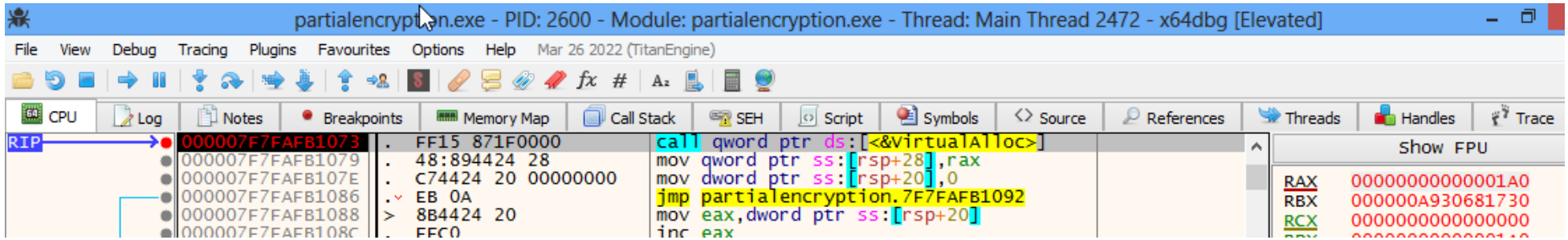


Set breakpoints on all VirtualAlloc and VirtualFree we can find in the code area

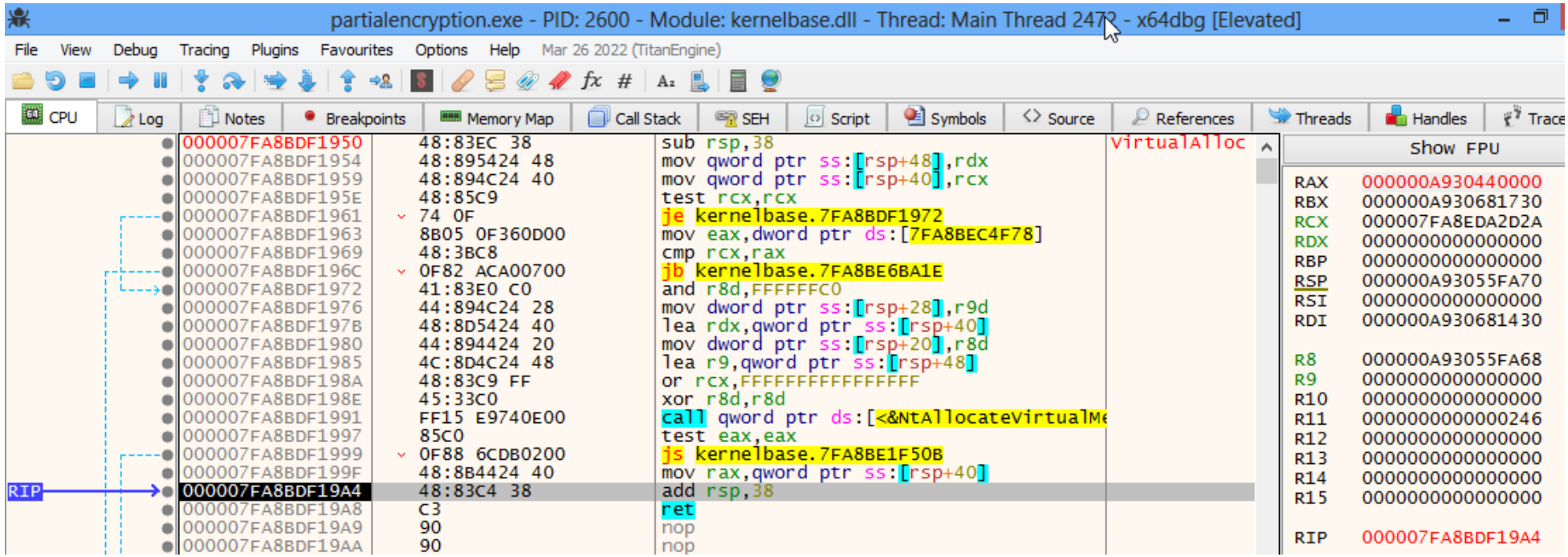
`SetBPX VirtualAlloc`



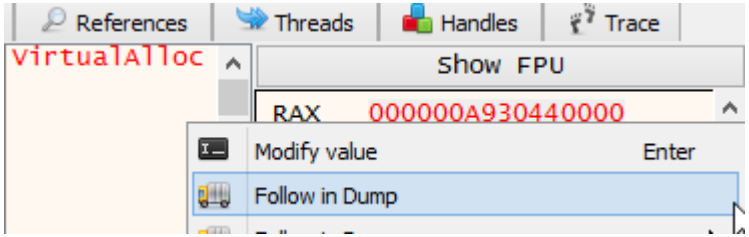
Run the binary and get it stopped at VirtualAlloc function



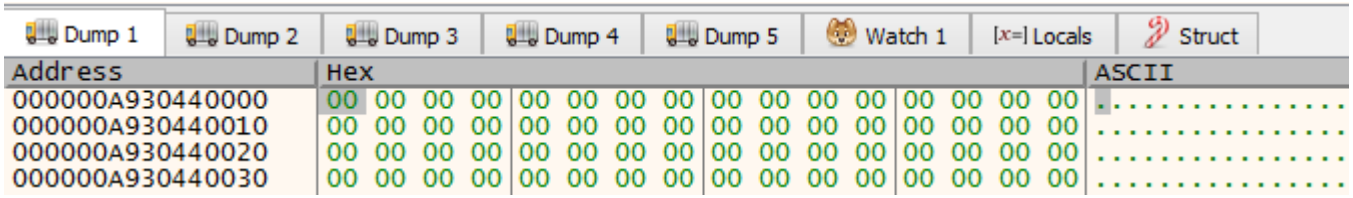
Step over until **ret** and **RAX** Register contains the memory address



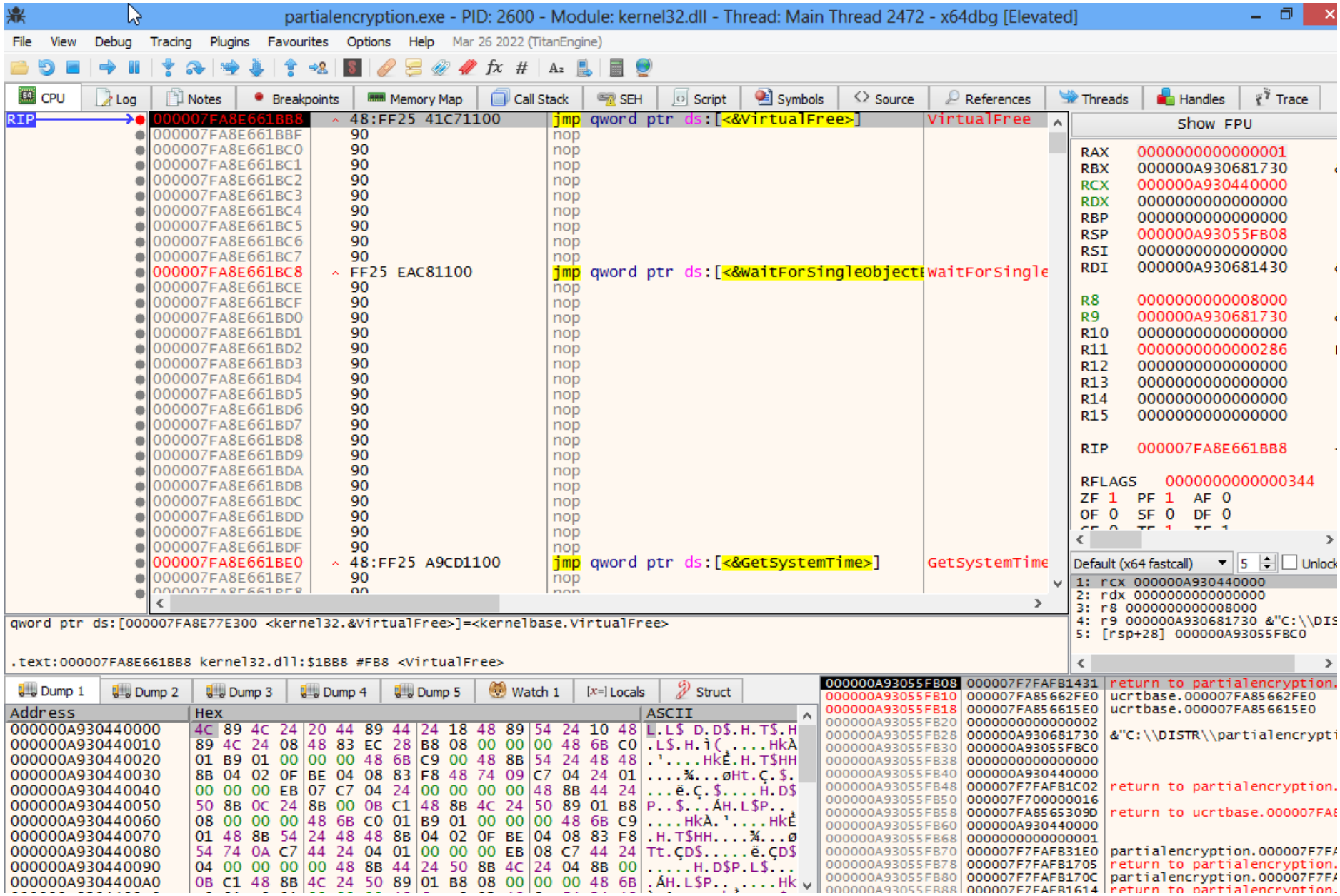
Follow RAX content in memory dump



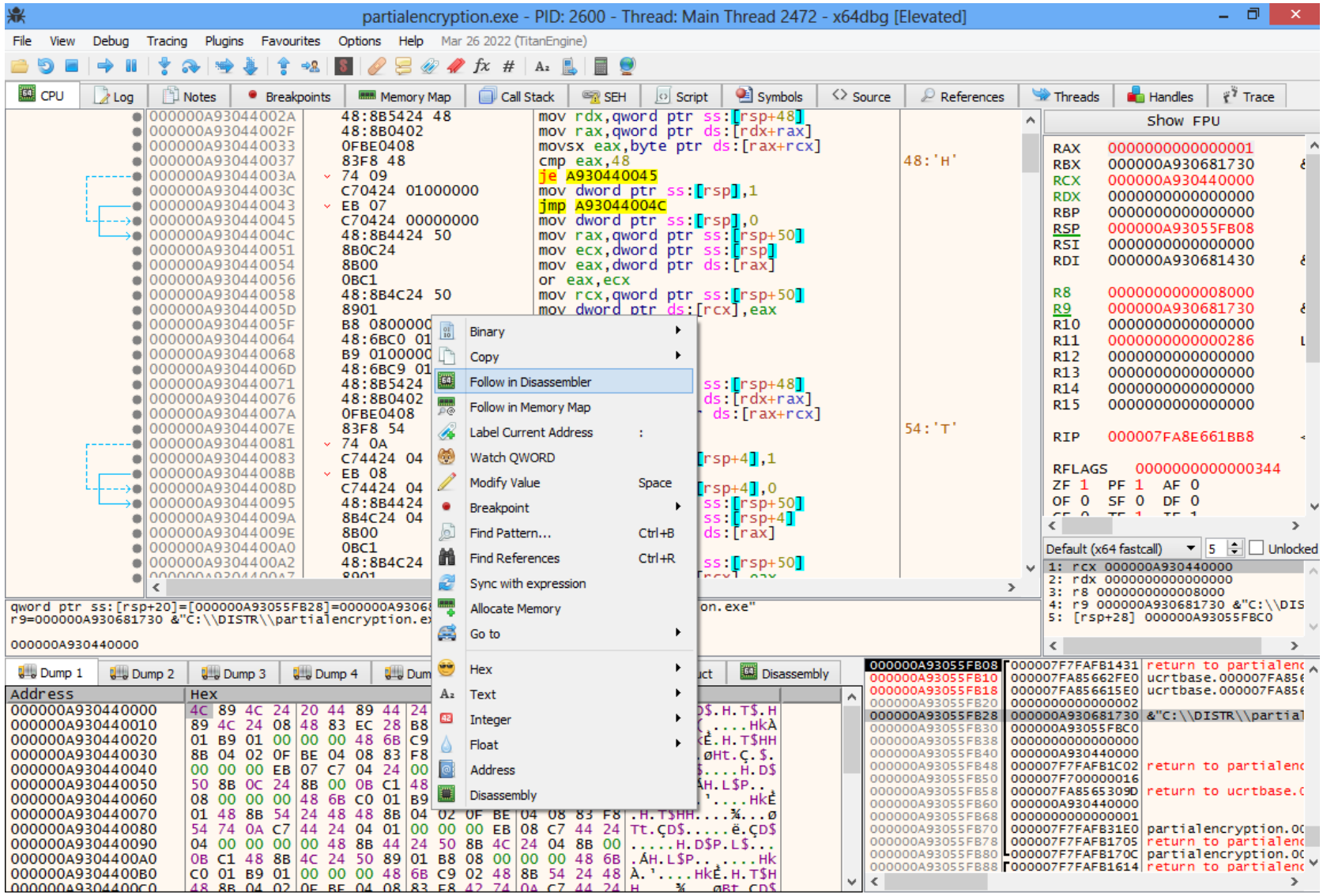
It will be filled with zeroes



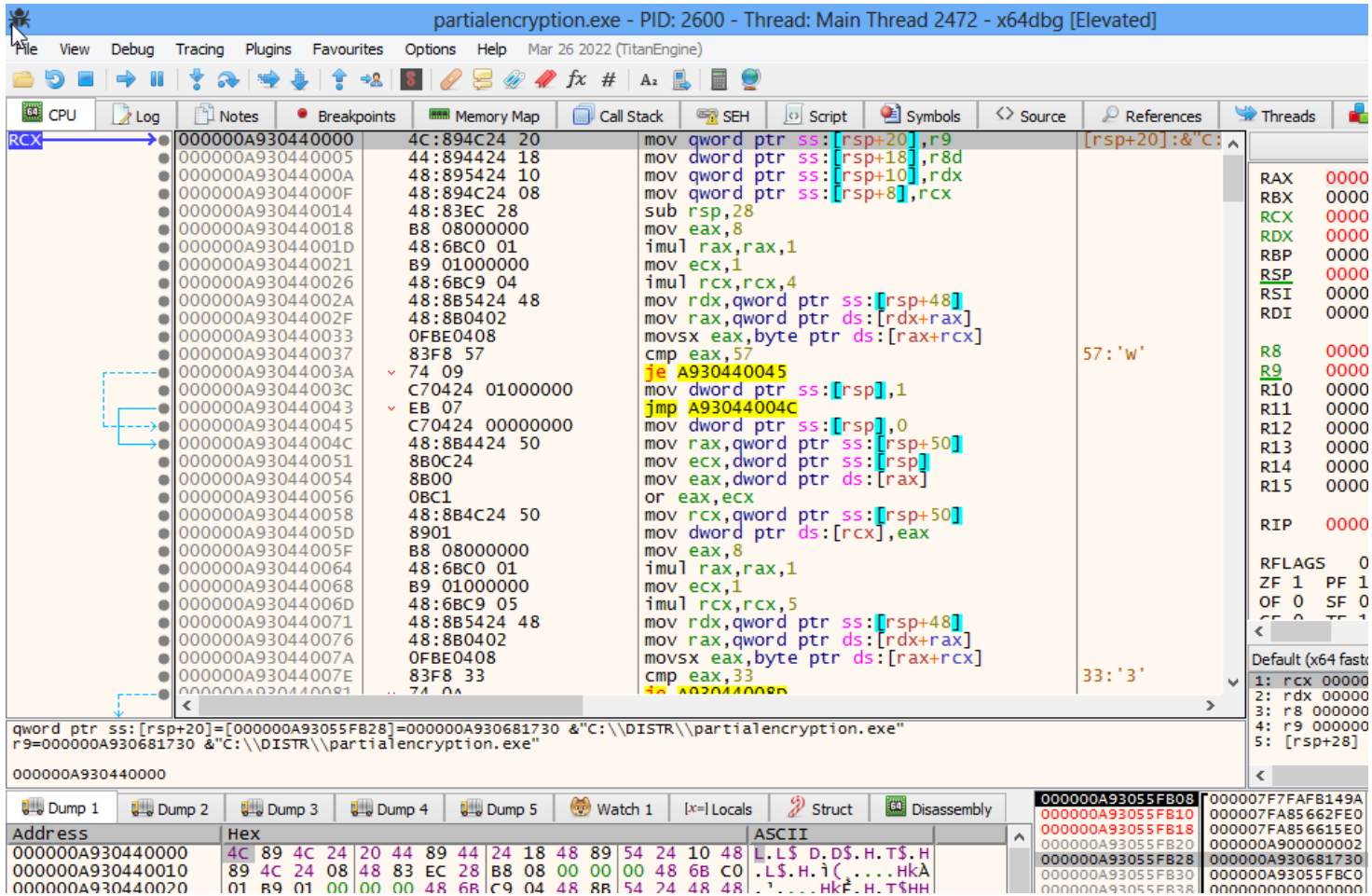
Then run the program until it hit VirtualFree


partialencryption.exe - PID: 2600 - Module: kernel32.dll - Thread: Main Thread 2472 - x64dbg [Elevated]
File View Debug Tracing Plugins Favourites Options Help Mar 26 2022 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
RIP
000007FA8E661BB8 48:FF25 41C71100 jmp qword ptr ds:[<&VirtualFree>] VirtualFree
000007FA8E661BBF 90 nop
000007FA8E661BC0 90 nop
000007FA8E661BC1 90 nop
000007FA8E661BC2 90 nop
000007FA8E661BC3 90 nop
000007FA8E661BC4 90 nop
000007FA8E661BC5 90 nop
000007FA8E661BC6 90 nop
000007FA8E661BC7 90 nop
000007FA8E661BC8 FF25 EAC81100 jmp qword ptr ds:[<&WaitForSingleObjectE WaitForSingle
000007FA8E661BCE 90 nop
000007FA8E661BCF 90 nop
000007FA8E661BD0 90 nop
000007FA8E661BD1 90 nop
000007FA8E661BD2 90 nop
000007FA8E661BD3 90 nop
000007FA8E661BD4 90 nop
000007FA8E661BD5 90 nop
000007FA8E661BD6 90 nop
000007FA8E661BD7 90 nop
000007FA8E661BD8 90 nop
000007FA8E661BD9 90 nop
000007FA8E661BDA 90 nop
000007FA8E661BDB 90 nop
000007FA8E661BDC 90 nop
000007FA8E661BDD 90 nop
000007FA8E661BDE 90 nop
000007FA8E661BDF 90 nop
000007FA8E661BE0 48:FF25 A9CD1100 jmp qword ptr ds:[<&GetSystemTime>] GetSystemTime
000007FA8E661BE7 90 nop
Show FPU
RAX 0000000000000001
RBX 000000A930681730
RCX 000000A930440000
RDX 0000000000000000
RBP 0000000000000000
RSP 000000A93055FB08
RSI 0000000000000000
RDI 000000A930681430
R8 0000000000008000
R9 000000A930681730
R10 0000000000000000
R11 0000000000000286
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RIP 000007FA8E661BB8
RFLAGS 0000000000000344
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
Default (x64 fastcall) 5 Unlock
1: rcx 000000A930440000
2: rdx 0000000000000000
3: r8 0000000000008000
4: r9 000000A930681730 &"C:\\DIS
5: [rsp+28] 000000A93055FBC0
qword ptr ds:[000007FA8E77E300 <kernel32.&VirtualFree>]=<kernelbase.VirtualFree>
.text:000007FA8E661BB8 kernel32.dll:$1BB8 #FB8 <VirtualFree>
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
Address Hex ASCII
000000A930440000 4C 89 4C 24 20 44 89 44 24 18 48 89 54 24 10 48 L.L$ D.D$.H.T$.H
000000A930440010 89 4C 24 08 48 83 EC 28 B8 08 00 00 00 48 6B C0 .L$.H.ì(.....HkÀ
000000A930440020 01 B9 01 00 00 00 48 6B C9 00 48 8B 54 24 48 48 .¹....HkÉ.H.T$HH
000000A930440030 8B 04 02 0F BE 04 08 83 F8 48 74 09 C7 04 24 01 ....¾....øHt.Ç.$.
000000A930440040 00 00 00 EB 07 C7 04 24 00 00 00 00 48 8B 44 24 ...ë.Ç.$....H.D$
000000A930440050 50 8B 0C 24 8B 00 0B C1 48 8B 4C 24 50 89 01 B8 P..$....ÁH.L$P...
000000A930440060 08 00 00 00 48 6B C0 01 B9 01 00 00 00 48 6B C9 ....HkÀ.¹....HkÉ
000000A930440070 01 48 8B 54 24 48 48 8B 04 02 0F BE 04 08 83 F8 .H.T$HH....¾....ø
000000A930440080 54 74 0A C7 44 24 04 01 00 00 00 EB 08 C7 44 24 Tt.ÇD$.....ë.ÇD$
000000A930440090 04 00 00 00 00 48 8B 44 24 50 8B 4C 24 04 8B 00 .....H.D$P.L$...
000000A9304400A0 0B C1 48 8B 4C 24 50 89 01 B8 08 00 00 00 48 6B .ÁH.L$P.¸.....Hk
000000A93055FB08 000007F7FAFB1431 return to partialencryption.
000000A93055FB10 000007FA85662FE0 ucrtbase.000007FA85662FE0
000000A93055FB18 000007FA856615E0 ucrtbase.000007FA856615E0
000000A93055FB20 0000000000000002
000000A93055FB28 000000A930681730 &"C:\\DISTR\\partialencrypti
000000A93055FB30 000000A93055FBC0
000000A93055FB38 0000000000000000
000000A93055FB40 000000A930440000
000000A93055FB48 000007F7FAFB1C02 return to partialencryption.
000000A93055FB50 000007F700000016
000000A93055FB58 000007FA8565309D return to ucrtbase.000007FA8
000000A93055FB60 000000A930440000
000000A93055FB68 0000000000000001
000000A93055FB70 000007F7FAFB31E0 partialencryption.000007F7FA
000000A93055FB78 000007F7FAFB1705 return to partialencryption.
000000A93055FB80 000007F7FAFB170C partialencryption.000007F7FA

Follow in disassembler or disassembly the dump



As we can see the allocated memory before free contains HTB{ } chars that corresponds to the flag pattern. Repeat running the program until VirtualFree again



We get the first word of the flag. Run again until get the full flag

HTB{W3iRd_RUnT1m3_DEC}