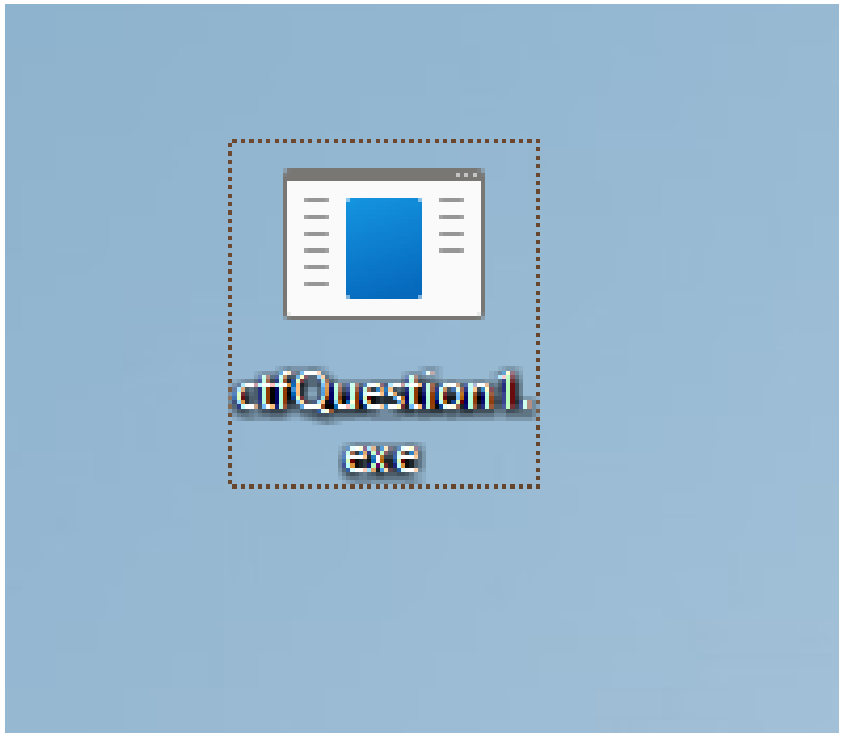
Patch Me If You Can

Hey, check out this annoying program I found! It just spams the screen with some weird art. The author must think they're clever, hiding a flag in here somewhere.

I took a quick look. The code is a mess, and I have a feeling it might even modify itself at runtime.To get the flag, you'll probably have to find a hidden key and manually slot it into the right place in memory while the program is running.

Think you can handle it? Good luck!

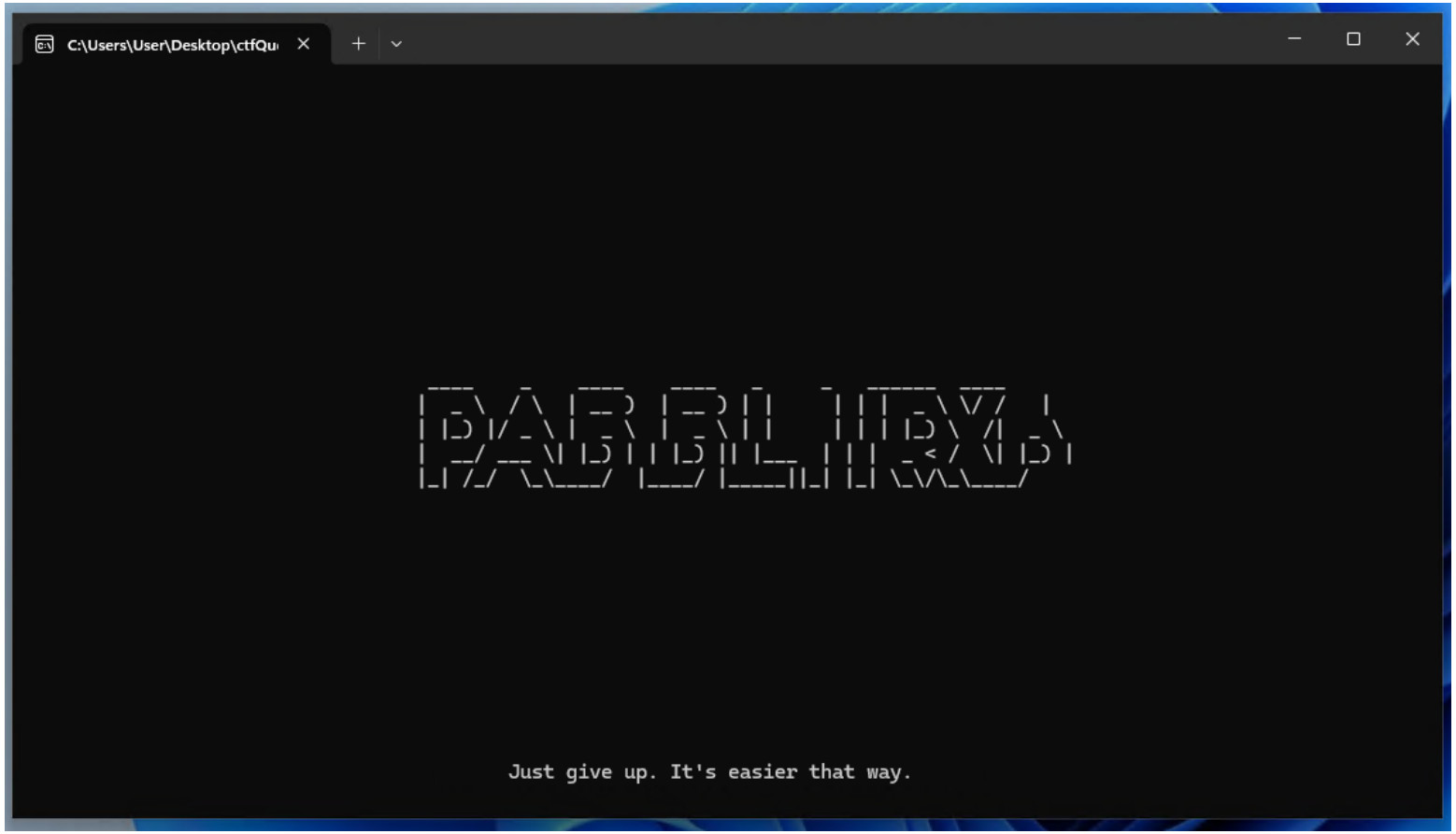


Just double click it to run.

Only shows this screen.

Looks like unlimited loop.

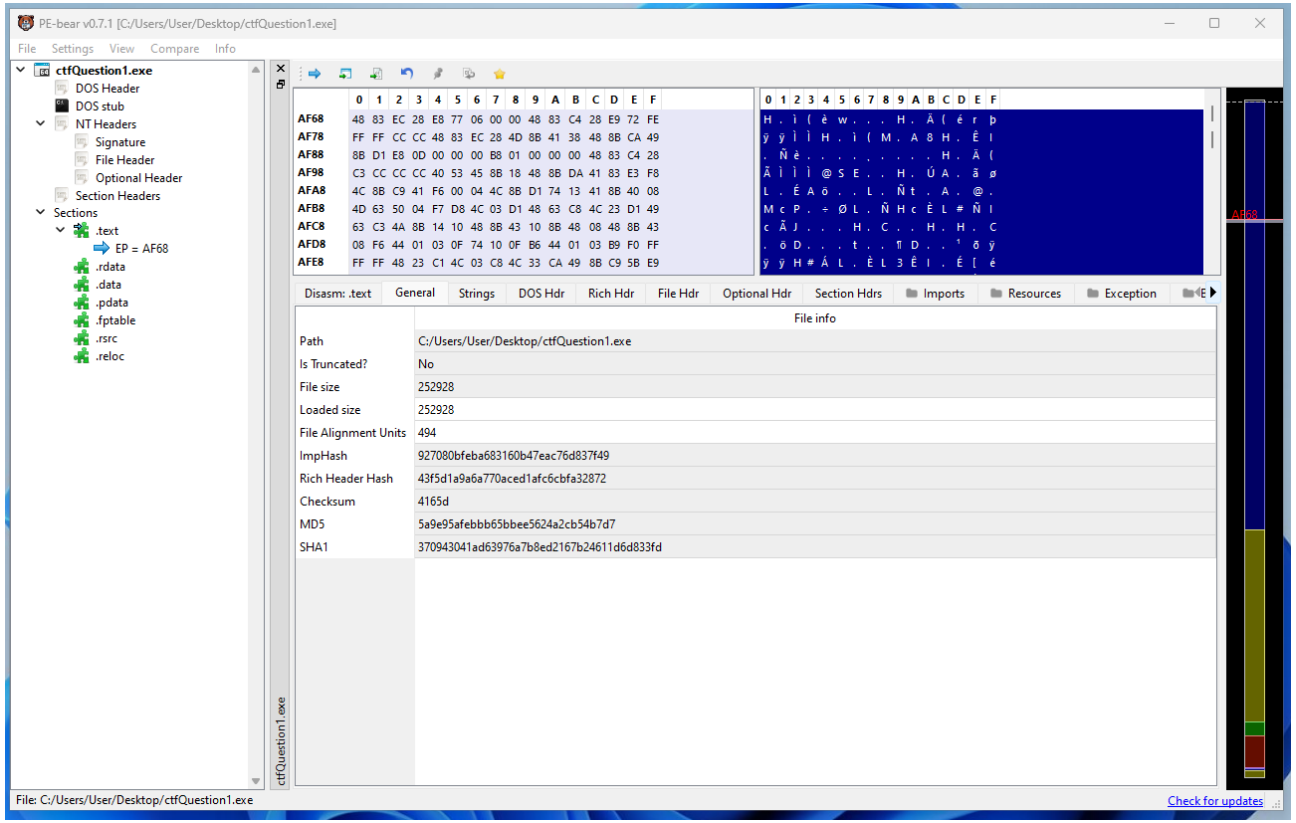
Nothing happens.

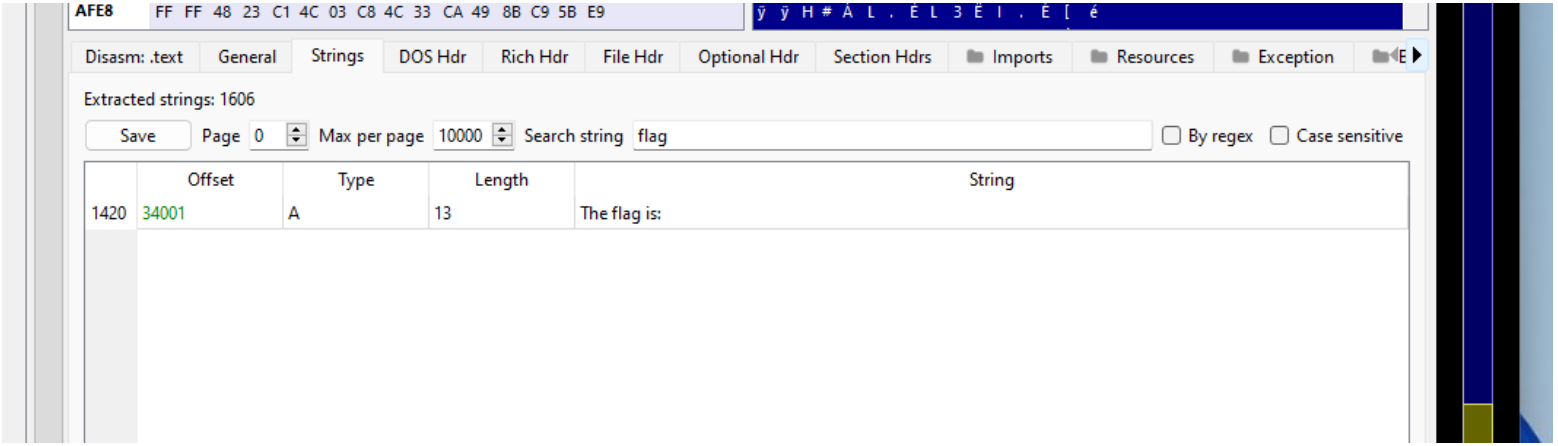


We have to see its meta data with PEbear first.

Looks like just exe console application.

Couldn’t find any flag in Strings view.



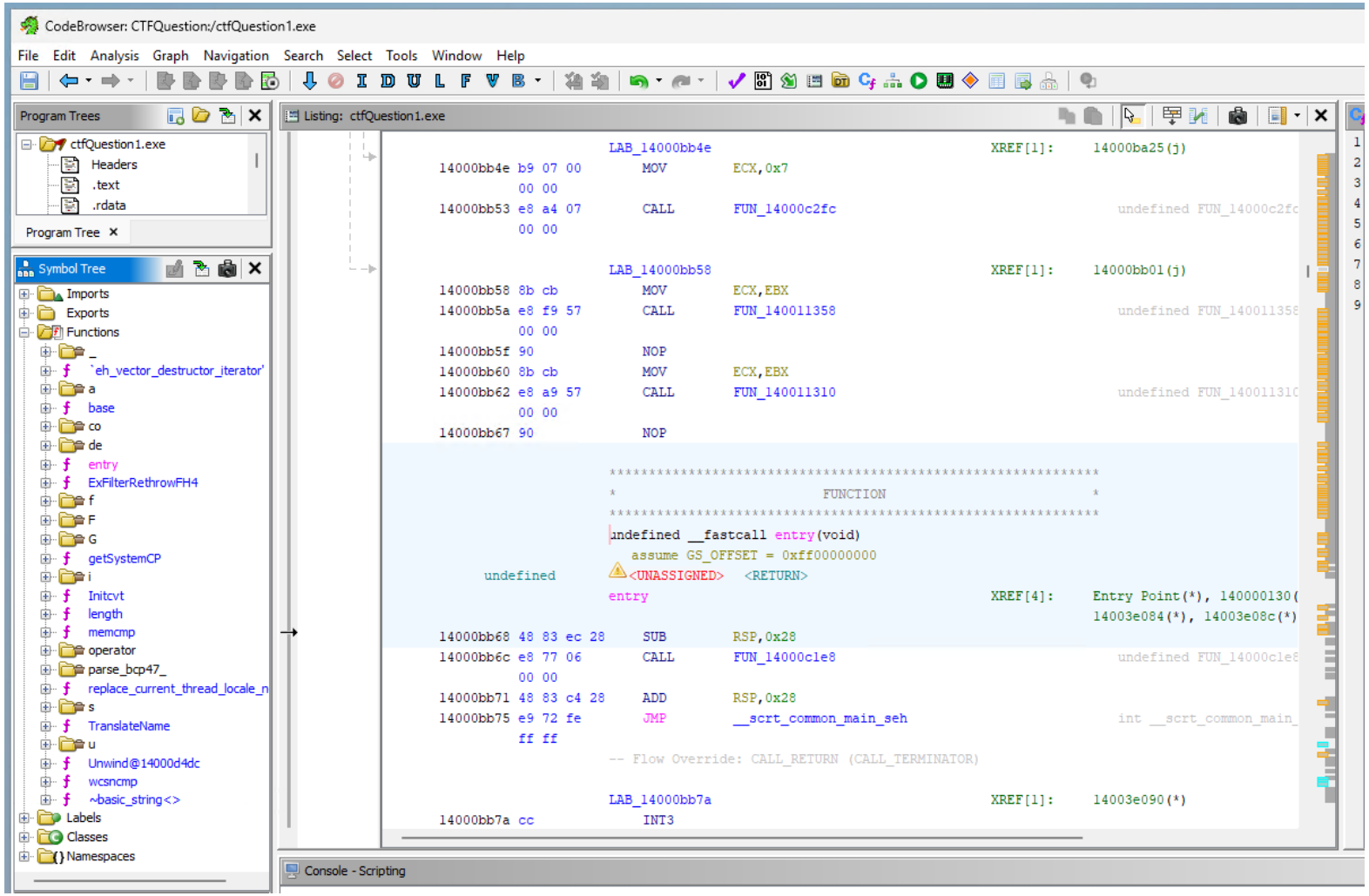


Find main function

Static analyze with Ghidra.

There is library functions other than entry point.

We need to find main function under entry point.



Decompile: entry - (ctfQuestion1.exe)

```
1
2 void entry(void)
3
4 {
5   FUN_14000c1e8();
6   __scrt_common_main_seh();
7   return;
8 }
9
```

Line 5 is just default code.

Look into __scrt_common_main_seh

```
1
2 /* WARNING: Function: _guard_dispatch_icall replaced with injection: guar
3 /* Library Function - Single Match
4    int __cdecl __scrt_common_main_seh(void)
5
6   Library: Visual Studio 2019 Release */
7
8 int __cdecl __scrt_common_main_seh(void)
9
10 {
11   bool bVar1;
12   bool bVar2;
13   int iVar3;
14   undefined8 uVar4;
15   undefined8 uVar5;
16   longlong *plVar6;
17   ulonglong uVar7;
18   ulonglong *puVar8;
19   uint uVar9;
20   undefined8 unaff_RBX;
21   undefined8 in_R9;
22   undefined1 uVar10;
23
24   uVar9 = (uint)unaff_RBX;
25   uVar4 = FUN_14000bc38(1);
26   if ((char)uVar4 == '\0') {
27     FUN_14000c2fc(7);
28   }
29   else {
30     bVar1 = false;
31     uVar10 = 0;
32     uVar4 = __scrt_acquire_startup_lock();
33     uVar9 = (uint)CONCAT71((int7)((ulonglong)unaff_RBX >> 8),(char)uVar4)
34     if (DAT_14003be40 != 1) {
```

```
48     __scrt_release_startup_lock((char)uVar4);
49     plVar6 = (longlong *)FUN_14000c2e0();
50     if ((*plVar6 != 0) && (uVar7 = FUN_14000bd00((longlo
51       (*(code *)*plVar6)(0,2,0,in_R9,uVar10);
52     }
53     puVar8 = (ulonglong *)FUN_14000c2e8();
54     if ((*puVar8 != 0) && (uVar7 = FUN_14000bd00((longlo
55       FUN_14001131c(*puVar8);
56     }
57     FID_conflict:_get_initial_narrow_environment();
58     FUN_1400152a4();
59     FUN_14001529c();
60     uVar4 = FUN_140003b20();
61     uVar9 = (uint)uVar4;
62     bVar2 = FUN_14000c450();
63     if (bVar2) {
64       if (!bVar1) {
65         _cexit();
66       }
67       __scrt_uninitialize_crt(true,'\0');
68       return uVar9;
69     }
70     goto LAB_14000bb58;
71   }
72 }
73 FUN_14000c2fc(7);
74 LAB_14000bb58:
75 FUN_140011358(uVar9);
76 FUN_140011310(uVar9);
77 FUN_14000c1e8();
78 iVar3 = __scrt_common_main_seh();
79 return iVar3;
```

The line 60 FUN_140003b20 looks like main since its located between default codes above and below.

```
undefined8 FUN_140003b20(void)

{
  FUN_140003360();
  FUN_140002ee0();
  FUN_140003a60();
  FUN_140003fa0(&DAT_14003b3e0,0x140034ca8);
  FUN_140004b00(&DAT_14003b620);
  return 0;
}
```

This is main function.

It calls 5 functions.

```
void FUN_140003360(void)

{
  DAT_14003ca74 = 0x54;
  DAT_14003ca73 = 0x50;
  DAT_14003ca72 = 0x59;
  DAT_14003ca71 = 0x52;
  DAT_14003ca70 = 0x43;
  DAT_14003ca6f = 0x45;
  DAT_14003ca6e = 0x44;
  DAT_14003ca6d = 0x5f;
  DAT_14003ca6c = 0x4f;
  DAT_14003ca6b = 0x54;
  DAT_14003ca6a = 0x5f;
  DAT_14003ca69 = 0x59;
  DAT_14003ca68 = 0x45;
  DAT_14003ca67 = 0x4b;
  DAT_14003ca66 = 0x5f;
  DAT_14003ca65 = 0x52;
  DAT_14003ca64 = 0x45;
  DAT_14003ca63 = 0x54;
  DAT_14003ca62 = 0x53;
  DAT_14003ca61 = 0x41;
  DAT_14003ca60 = 0x4d;
  DAT_14003ca5f = 0x5f;
  DAT_14003ca5e = 0x53;
  DAT_14003ca5d = 0x49;
  DAT_14003ca5c = 0x5f;
  DAT_14003ca5b = 0x53;
  DAT_14003ca5a = 0x49;
  DAT_14003ca59 = 0x48;
  DAT_14003ca58 = 0x54;
  DAT_14003ca75 = 0;
  return;
}
```

First one is just defining strings but does nothing.

This code is sequentially writing byte values to adjacent memory locations. By converting the hexadecimal values to their corresponding ASCII characters and arranging them in the correct order, we can decode the string being constructed.

Here is the breakdown of the decoding:

Memory Address	Hex Value	ASCII Character
DAT_14003ca58	0x54	T
DAT_14003ca59	0x48	H
DAT_14003ca5a	0x49	I
DAT_14003ca5b	0x53	S
DAT_14003ca5c	0x5f	_
DAT_14003ca5d	0x49	I
DAT_14003ca5e	0x53	S
DAT_14003ca5f	0x5f	_
DAT_14003ca60	0x4d	M
DAT_14003ca61	0x41	A

- Reading the characters from bottom to top (in order of memory address) and then the final two addresses, the decoded string is:
- **THIS_IS_MASTER_KEY_TO_DECRYPT**

Gemini decrypted this one shows the master key. Okey?

This one is second function that main called.

Looks the same as what I see on console.

Decompile: FUN_140002ee0 - (ctfQuestion1.exe)

```
/* WARNING: Removing unreachable block (ram,0x0001400032ff) */

void FUN_140002ee0(void)

{
  undefined1 auVar1 [16];
  undefined8 *puVar2;
  undefined8 uVar3;
  _String_val<> *p_Var4;
  longlong *plVar5;
  longlong lVar6;
  undefined1 auStack_298 [32];
  int local_278;
```

```
ulonglong local_28;

local_28 = DAT_14003a040 ^ (ulonglong)auStack_298;
local_248 = GetStdHandle(0xfffffff5);
GetConsoleCursorInfo(local_248,&local_78);
local_78.bVisible = 0;
SetConsoleCursorInfo(local_248,&local_78);
FUN_140003340((undefined1 *)local_70,0x18);
FUN_1400049d0(local_1d8,"  ____    _    ____    ____   _     _   _____  ____  ");
FUN_1400049d0(local_1b8," |  _ \\  / \\  | __ )  | __ ) | |     | | |  _ \\ \\/ /   | ");
FUN_1400049d0(local_198," | |_) |/ _ \\ |  _ \\  |  _ \\ | |     | | | |_) \\  /|  _ \\ ");
FUN_1400049d0(local_178," |  __/ ___ \\| |_) | | |_) || |___  | | |  _ < /  \\| |_) |");
FUN_1400049d0(local_158," |_| /_/  \\_\\____/  |____/ |_____||_| |_| \\_\\/\\_\\____/ ");
puVar2 = FUN_140004600(local_1f8,local_1d8,local_138);
plVar5 = local_218;
for (lVar6 = 0x10; lVar6 != 0; lVar6 = lVar6 + -1) {
  *(undefined1 *)plVar5 = *(undefined1 *)puVar2;
  puVar2 = (undefined8 *)((longlong)puVar2 + 1);
  plVar5 = (longlong *)((longlong)plVar5 + 1);
}
FUN_140004730(&local_274);
FUN_1400046b0(local_70,local_218);
`eh_vector_destructor_iterator'(local_1d8,0x20,5,~basic_string<>);
FUN_140003340((undefined1 *)local_40,0x18);
FUN_1400049d0(local_128,"Analyzing your keystrokes...");
FUN_1400049d0(local_108,"Still here? Don\'t you have better things to do?");
FUN_1400049d0(local_e8,"Bet you can\'t debug me.");
FUN_1400049d0(local_c8,"Just give up. It\'s easier that way.");
```

do {

....

} while(true);

Program is unlimited loop here. But those code is just showing strings on the console...

This one is third function that never called, because the program stacked on second function.

But why developer remain this function?

There is many logic inside and I don’t wanna take time to read it all.

Since its never called then just run for debugging.

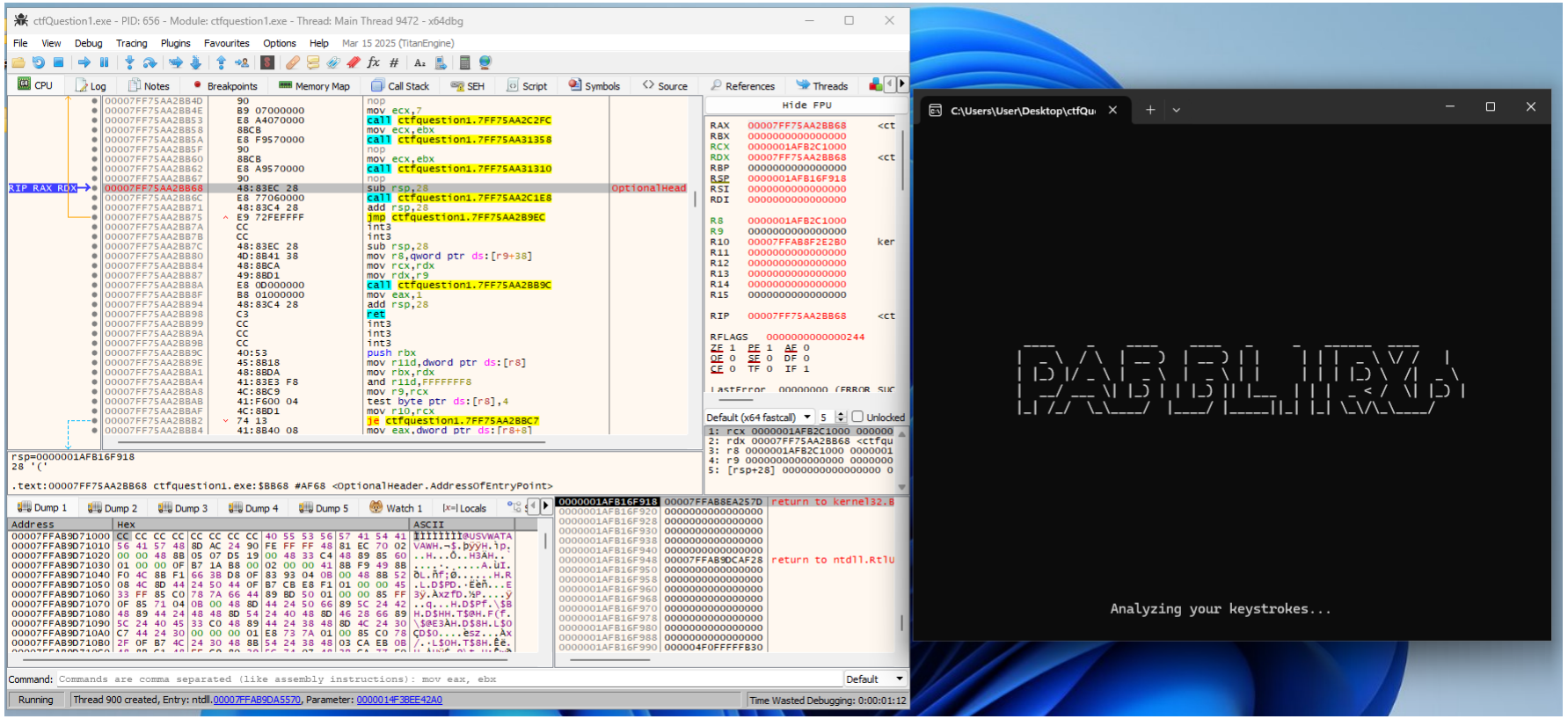
Decompile: FUN_140003a60 - (ctfQuestion1.exe)

```
void FUN_140003a60(void)

{
  undefined8 uVar1;
  longlong *plVar2;
  undefined1 auStack_58 [32];
  int local_38 [2];
  HANDLE local_30;
  longlong local_28;
  _CONSOLE_CURSOR_INFO local_20;
  ulonglong local_18;

  local_18 = DAT_14003a040 ^ (ulonglong)auStack_58;
  FUN_1400117b8(0x140034c60);
  FUN_140002ea0(0,0);
  local_30 = GetStdHandle(0xfffffff5);
  GetConsoleCursorInfo(local_30,&local_20);
  local_20.bVisible = 1;
  SetConsoleCursorInfo(local_30,&local_20);
  uVar1 = FUN_140003fa0(&DAT_14003b3e0,0x140034c68);
  FUN_140004ae0(uVar1,FUN_140004310);
  local_38[0] = 2;
  plVar2 = FUN_140003d60(&local_28,local_38);
  FUN_140004360(plVar2);
  FUN_140003930();
  FUN_14000b510(local_18 ^ (ulonglong)auStack_58);
  return;
}
```

Just run it with x64dbg.

Need to make break point to see its flow.

For now, I just put break point for main function.



As I found main on Ghidra and its offset is 3b20.

So I just find it on x64dbg as well.

```
1
2  undefined8 FUN_140003b20(void)
3
4  {
5    FUN_140003360();
6    FUN_140002ee0();
7    FUN_140003a60();
8    FUN_140003fa0(&DAT_14003b3e0,0x140034ca8);
9    FUN_140004b00(&DAT_14003b620);
10   return 0;
11 }
12
```

The location of it is here!

Looks same as what I saw on ghidra, it calls 5 functions.

```
00007FF75AA23B1E   CC                 int3
00007FF75AA23B1F   CC                 int3
00007FF75AA23B20   48:83EC 28         sub rsp,28
00007FF75AA23B24   E8 37F8FFFF        call ctfquestion1.7FF75AA23360
00007FF75AA23B29   E8 B2F3FFFF        call ctfquestion1.7FF75AA22EE0
00007FF75AA23B2E   E8 2DFFFFFF        call ctfquestion1.7FF75AA23A60
00007FF75AA23B33   48:8D15 6E110300   lea rdx,qword ptr ds:[7FF75AA54CA8]     00007FF75AA5
00007FF75AA23B3A   48:8D0D 9F780300   lea rcx,qword ptr ds:[7FF75AA5B3E0]
00007FF75AA23B41   E8 5A040000        call ctfquestion1.7FF75AA23FA0
00007FF75AA23B46   48:8D0D D37A0300   lea rcx,qword ptr ds:[7FF75AA5B620]
00007FF75AA23B4D   E8 AE0F0000        call ctfquestion1.7FF75AA24B00
00007FF75AA23B52   33C0               xor eax,eax
00007FF75AA23B54   48:83C4 28         add rsp,28
00007FF75AA23B58   C3                 ret
00007FF75AA23B59   CC                 int3
00007FF75AA23B5A   CC                 int3
00007FF75AA23B5B   CC                 int3

R13   00C
R14   00C
R15   00C

RIP   00C

RFLAGS
ZF 1   PF
OF 0   SF
CF 0   TF

LastError

Default (x64 f
1: rcx 00
```

Since I just wanna run third function then, remove second function.

Can remove it by filling with NOPs.

Also made breakpoint on first function.

```
00007FF75AA23B1E   CC          int3
00007FF75AA23B1F   CC          int3
00007FF75AA23B20   48:83EC 28  sub rsp,28
00007FF75AA23B24   E8 37F8FFFF call ctfquestion1.7FF75AA23360
00007FF75AA23B29   E8 B2F3FFFF call ctfquestion1.7FF75AA22EE0
00007FF75AA23B2E   E8 2DFFFFFF call ctfquestion1.7FF75AA23A60
00007FF75AA23B33   48:8D15 6E110300 lea rdx,qword ptr ds:[7FF75AA54CA8]   00007FF75AA5
00007FF75AA23B3A   48:8D0D 9F780300 lea rcx,qword ptr ds:[7FF75AA5B3E0]
00007FF75AA23B41   E8 5A040000 call ctfquestion1.7FF75AA23FA0
00007FF75AA23B46   48:8D0D D37A0300 lea rcx,qword ptr ds:[7FF75AA5B620]
00007FF75AA23B4D   E8 AE0F0000 call ctfquestion1.7FF75AA24B00
00007FF75AA23B52   33C0        xor eax,eax
```

```
int3
sub rsp,28
call ctfquestion1.7FF75AA23360
call ctfquestion1.7FF75AA22EE0
call ctfquestion
lea rdx,qword pt
lea rcx,qword pt
call ctfquestion
lea rcx,qword pt
call ctfquestion
xor eax,eax
add rsp,28
ret
int3
int3
int3
```

Binary
Copy
Breakpoint
Follow in Dump
Follow in Disassembler
Follow in Memory Map
Graph G

Edit Ctrl+E
Fill... F
Fill with NOPs Ctrl+9
Copy Shift+C

```
R14 0000000000000000
R15 0000000000000000
RIP 00007FF75AA2BB68
Default (x64 fastcall) 5
1: rcx 0000001AFB2C1000
2: rdx 00007FF75AA2BB68 <cti
```

```
00007FF75AA23B20   48:83EC 28  sub rsp,28
00007FF75AA23B24   E8 37F8FFFF call ctfquestion1.7FF75AA23360
00007FF75AA23B29   90          nop
00007FF75AA23B2A   90          nop
00007FF75AA23B2B   90          nop
00007FF75AA23B2C   90          nop
00007FF75AA23B2D   90          nop
00007FF75AA23B2E   E8 2DFFFFFF call ctfquestion1.7FF75AA23A60
00007FF75AA23B33   48:8D15 6E110300 lea rdx,qword ptr ds:[7FF75AA54CA8]   000
00007FF75AA23B3A   48:8D0D 9F780300 lea rcx,qword ptr ds:[7FF75AA5B3E0]
00007FF75AA23B41   E8 5A040000 call ctfquestion1.7FF75AA23FA0
```

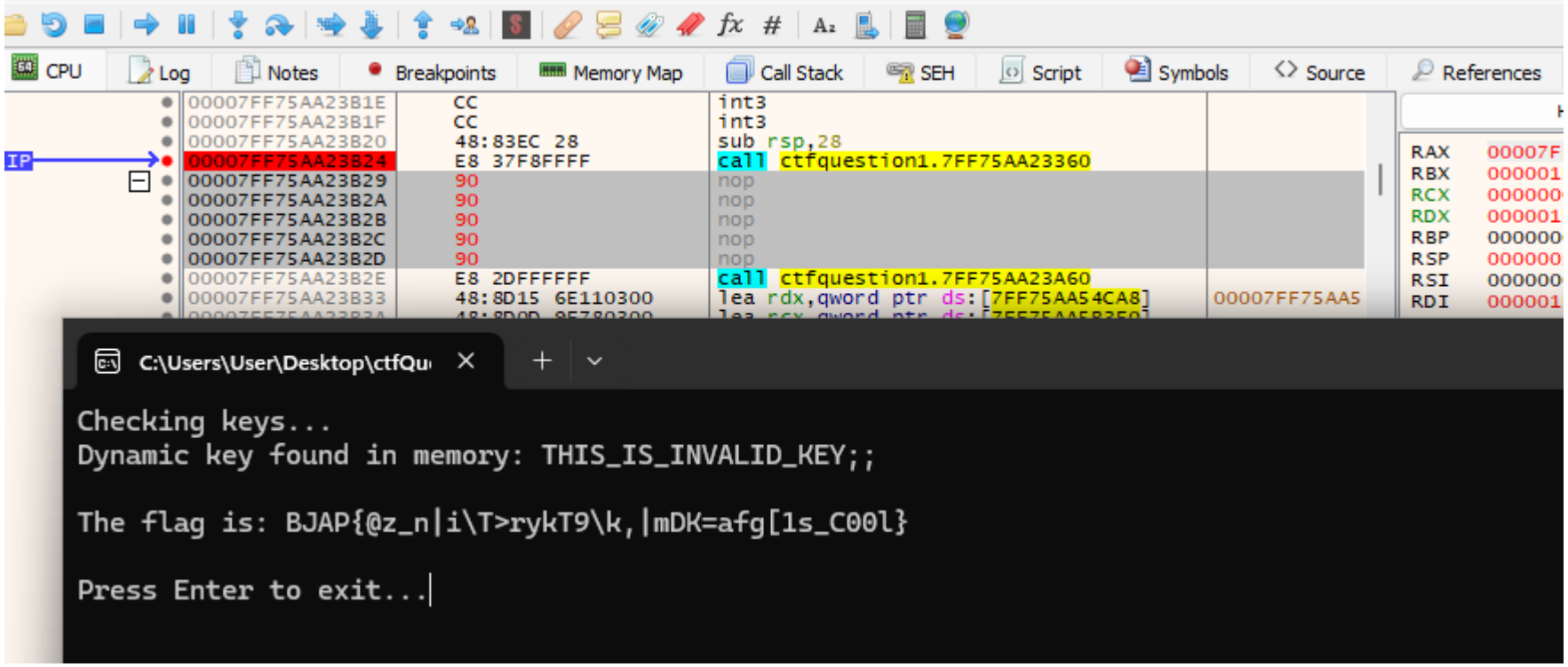
```
00007FF75AA23B1E   CC          int3
00007FF75AA23B1F   CC          int3
00007FF75AA23B20   48:83EC 28  sub rsp,28
00007FF75AA23B24   E8 37F8FFFF call ctfquestion1.7FF75AA23360
00007FF75AA23B29   90          nop
00007FF75AA23B2A   90          nop
00007FF75AA23B2B   90          nop
00007FF75AA23B2C   90          nop
00007FF75AA23B2D   90          nop
00007FF75AA23B2E   E8 2DFFFFFF call ctfquestion1.7FF75AA23A60
00007FF75AA23B33   48:8D15 6E110300 lea rdx,qword ptr ds:[7FF75AA54CA8]   00
00007FF75AA23B3A   48:8D0D 9F780300 lea rcx,qword ptr ds:[7FF75AA5B3E0]
00007FF75AA23B41   E8 5A040000 call ctfquestion1.7FF75AA23FA0
00007FF75AA23B46   48:8D0D D37A0300 lea rcx,qword ptr ds:[7FF75AA5B620]
```

The second function is executed correctly.

It shows flag but broken.

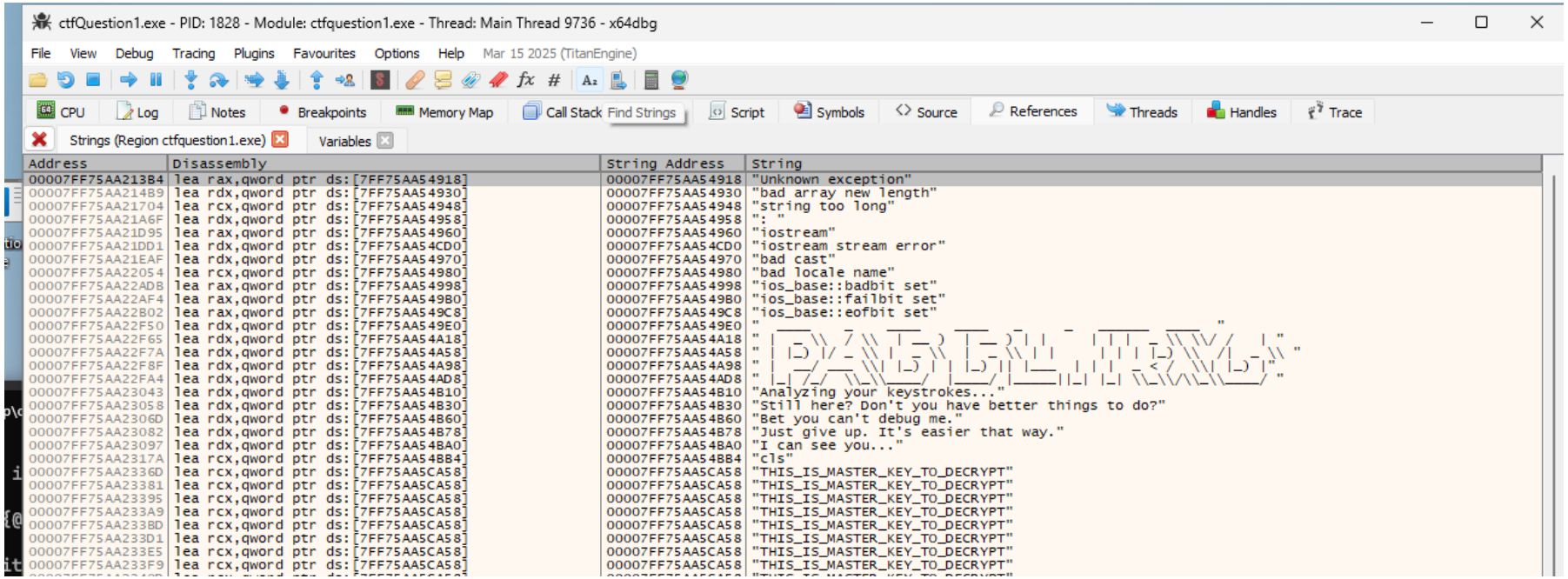
But It shows “THIS_IS_INVALID _KEY;;”

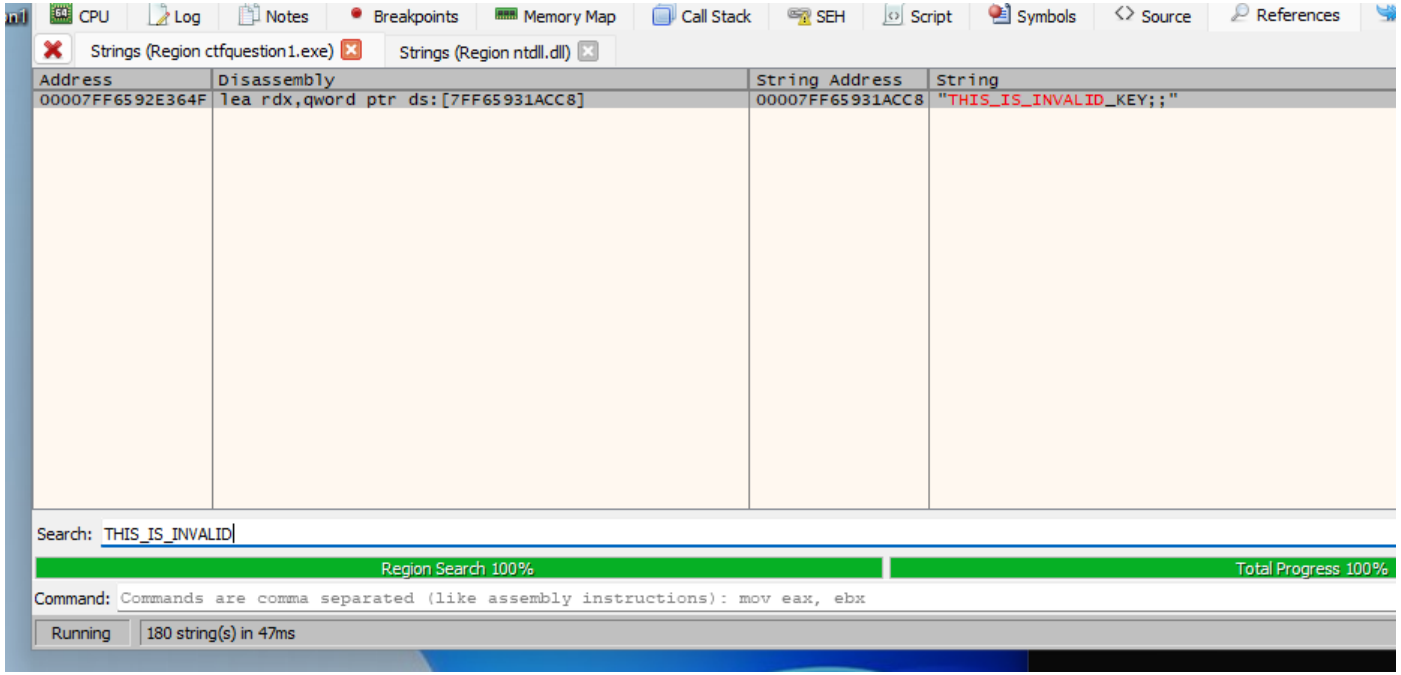
Maybe need to use correct master key that I got at beginning.



I can search strings on Find strings view.

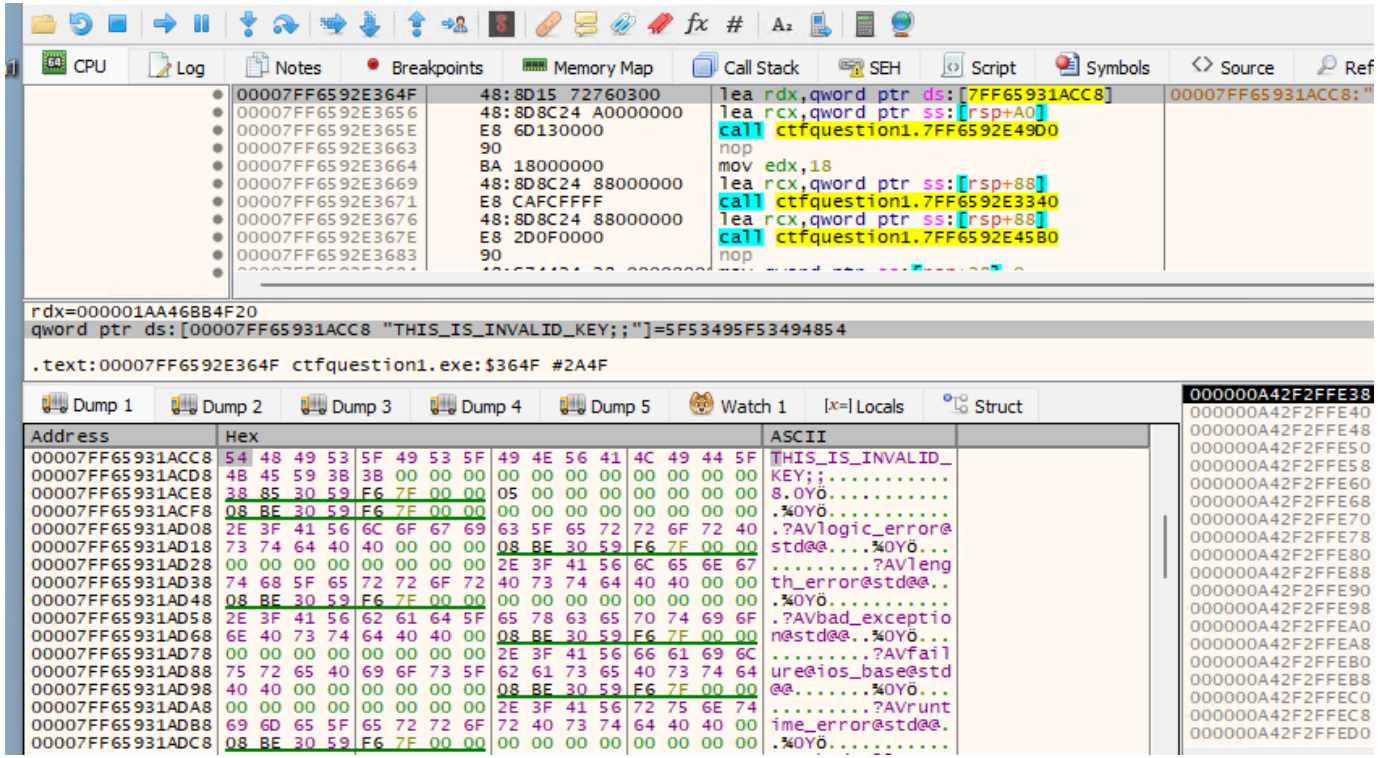
After searched it then the memory address of it is 00007FF6592E364F.

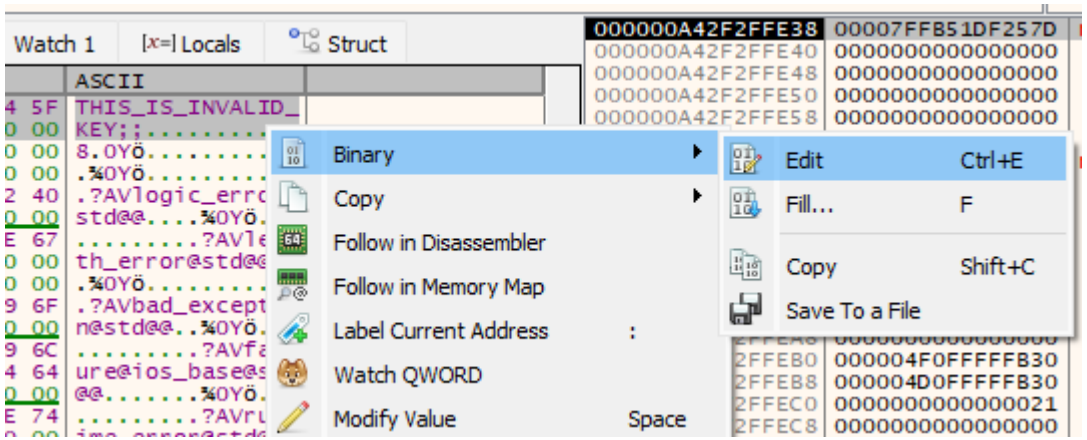




Since I find the location then change value to master key by editing binary.

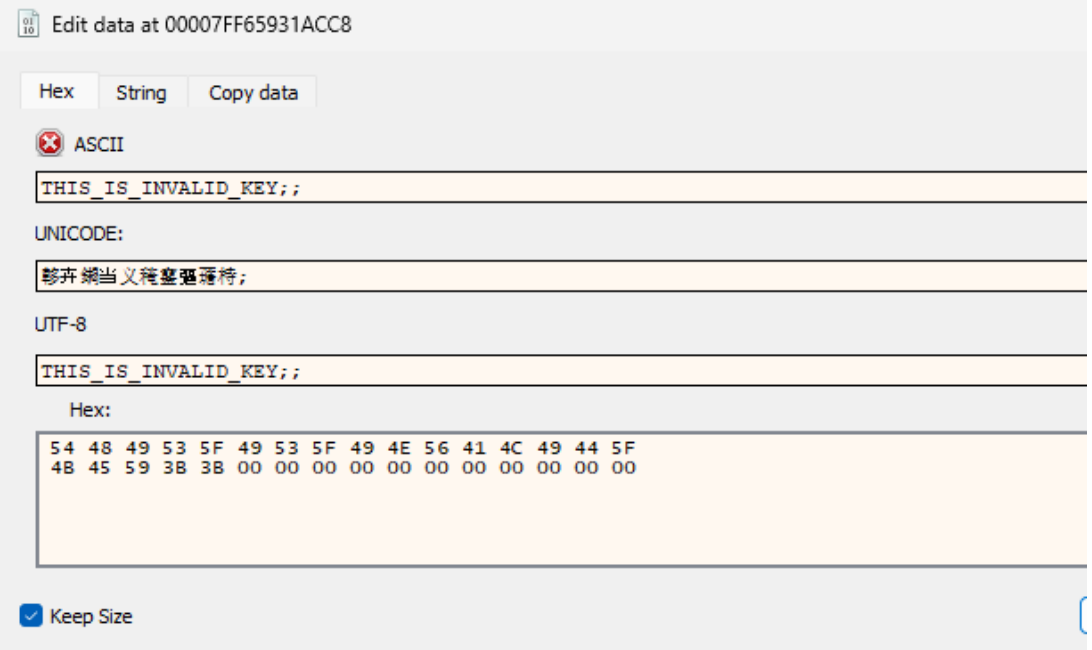
Need to double click the yellow holder “[00007FF65931ACC8]” to see it on memory dump window!!!

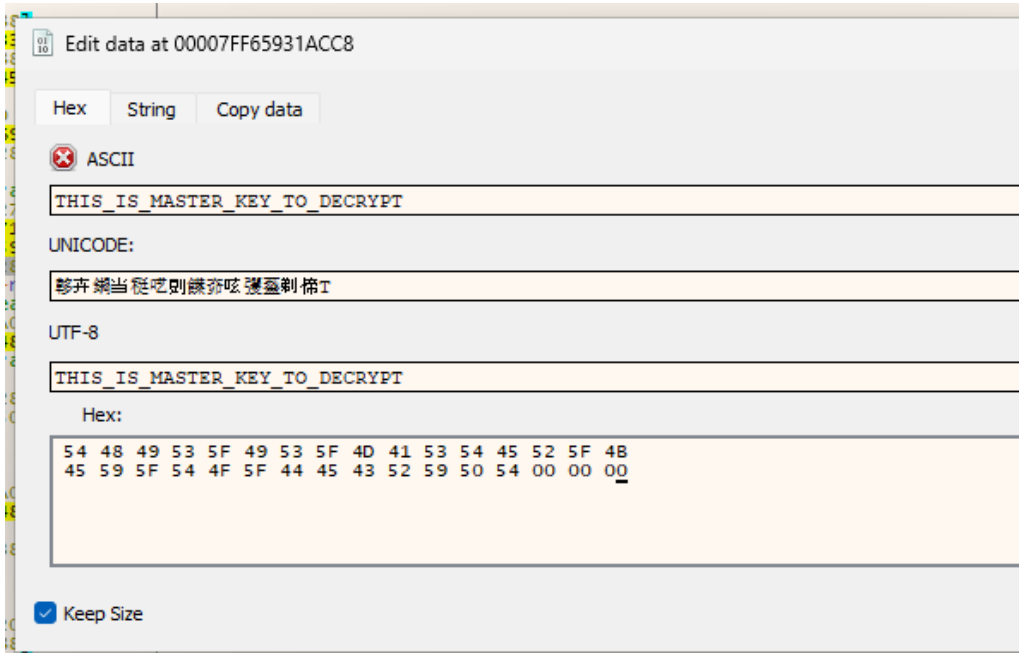




I changed it to correct master key.

Make sure to check “Keep Size” section.





Congratulation!

I got flag!!