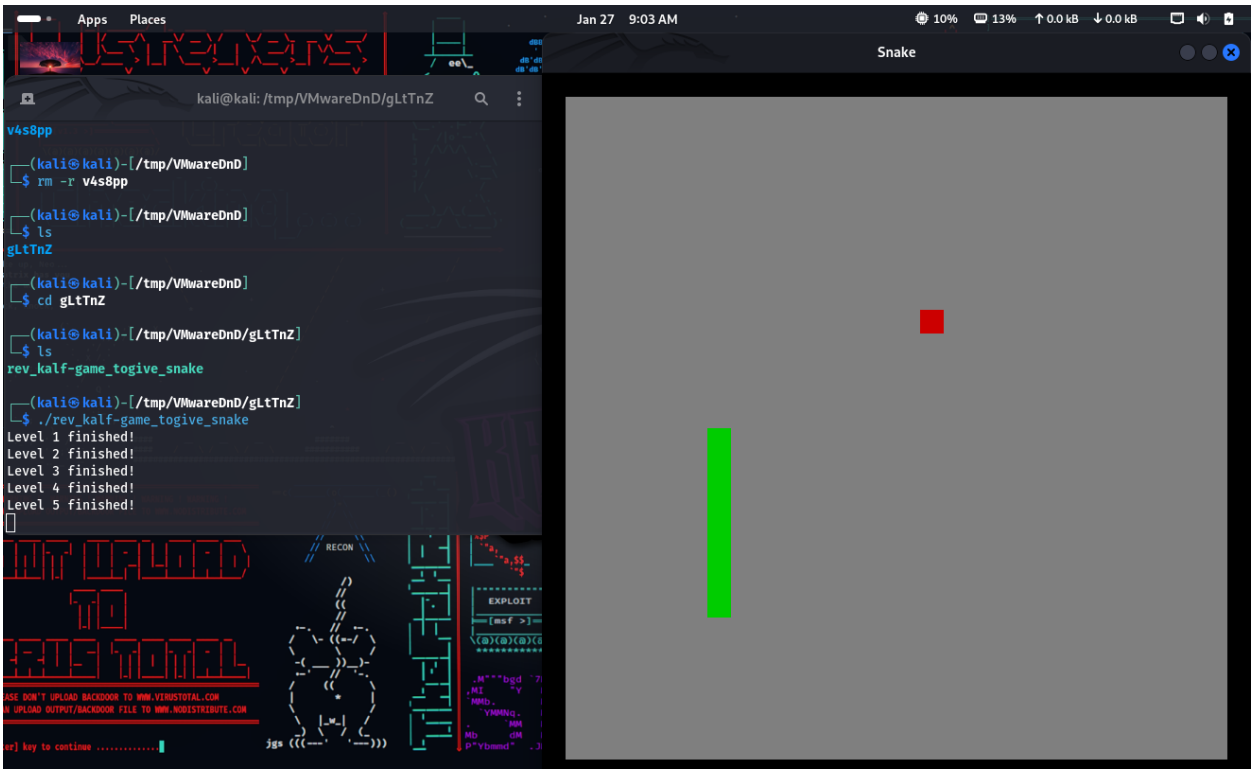
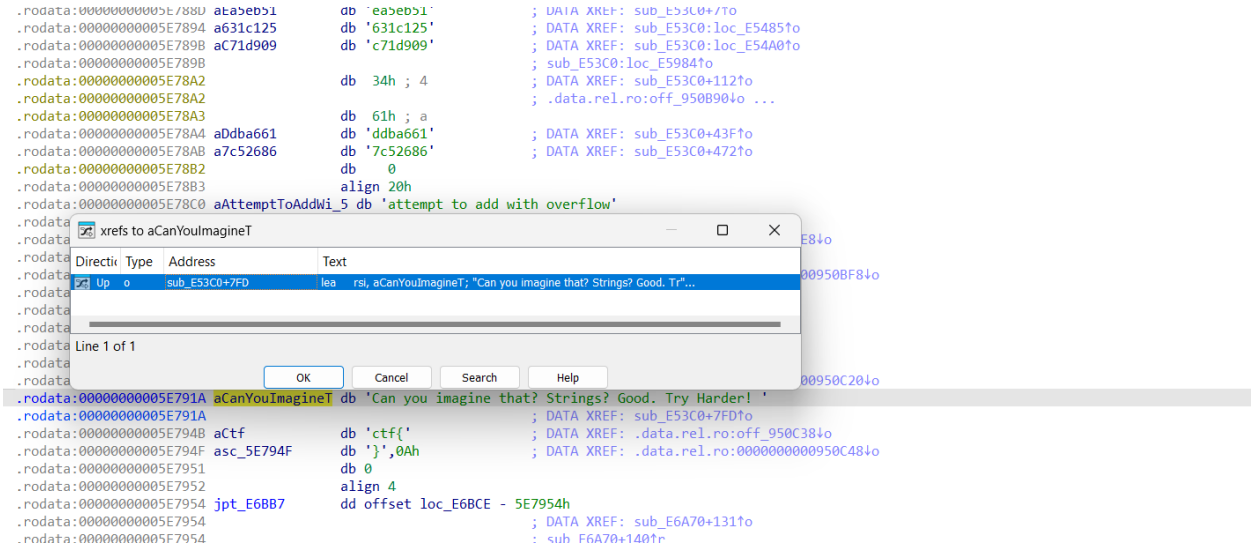
After opening it in IDA and seeing the countless functions, I decided to run the program, to get an understanding of what it does.



So, we have a snake game, which counts the number of red dots we eat by finished levels. Maybe there's something more in the logic of the game itself, so the next thing to do is investigate the function which controls the level counting. But where is it?

I opened the fila in IDA and went to the strings view (it is noteworthy that by investigating the program from main I couldn't understand a thing so we need to find the function by string). From string view I went into the rodata, to see what's in there and try to cross-reference to a function.

After looking around a bit, I saw that a lot of important-looking strings pointed to the same function:



sub_E53C0 seems to be what I was looking for. Let's have a look!

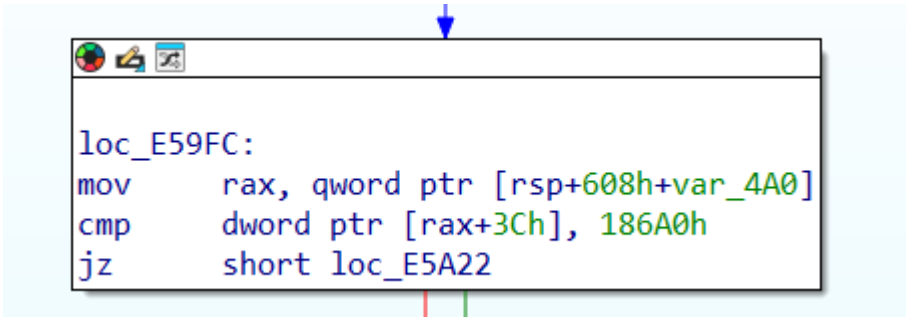
After a bit of analysis, I understood 2 things:

1. Inside this function the flag is probably being built. It's too hard to make out what happens exactly, so I'm not going to try to make the flag myself.

2. The only interesting part (in terms of what doesn't normally happens) can be seen here:

```
if ( *(_DWORD *)(a1 + 60) == 100000 )
```

In assembly, that instruction converts to this:



From here, I understand that the flag finishes building only if that value is in rax, so the next thing in order is to see what that value signifies. By making sure rax has that value we may be able to recover the flag.

Since PIE is enabled, we have aslr to deal with, so getting to that point is going to be a bit of a pain.

After breaking in the main function with gdb and running, we can run vmmap to see what is going on in the vm's memory:

```
0x555555400000     0x555555b4d000 r-xp   74d000      0 /home/kali/.cache/vmware/drag_and_drop/gLtTnZ/rev_kalf-game_togive_snake
0x555555d4d000     0x555555d87000 r--p    3a000 74d000 /home/kali/.cache/vmware/drag_and_drop/gLtTnZ/rev_kalf-game_togive_snake
0x555555d87000     0x555555d8d000 rw-p     6000 787000 /home/kali/.cache/vmware/drag_and_drop/gLtTnZ/rev_kalf-game_togive_snake
```

The program starts at **0x555555400000**. The function starts at **0xe53c0**. The instruction is at **+644** inside of the function. As a result, the break command we need is **break *0x555555400000 + 0xe53c0 + 0x644**.

After continuing, the game starts. After collecting a red dot, it stops as it reaches that function. My initial assumption was that **sub_E53C0** only executes after the game is over, but it seems I made a wrong assumption there.

```
pwndbg>  x/wx $rax+0x3C
0x7fffffffd984: 0x00000001
```

So it's 1…. Hmmm…..

I continue and collect another red dot:

```
pwndbg> x/wx $rax+0x3C
0x7fffffffd984: 0x00000002
```

So… I think it's safe to conclude at that offset the number of red dots (or levels passed) is stored.

So… **set {int}($rax+0x3C) = 0x186A0**

After we type continue a message pops up in gdb, telling us the flag.

Made with love by: AndreiCat