Guilherme Alt Chagas Merklein

PicoCTF 2024 - Reverse Engineering

flag: picoCTF{d3bug_f0r_th3_Win_0x100_e6c390e2}

27/03/2024

WinAntiDbg0x100

In this challenge, we were present with two files, a windows executable and a file named config.bin. I opened the config.bin file and there were just random characters, which I suppose are gonna be used by the executable. I ran it on my terminal and the program printed a message telling me to launch it inside a debugger.

Before launching the debugger, I used the strings command on the executable, and it did print some suspicious strings, but none of them was saying anything useful, so I gave up and launched *IDA Freeware* to start analyzing the static code. After this, I loaded the executable to disassemble it.

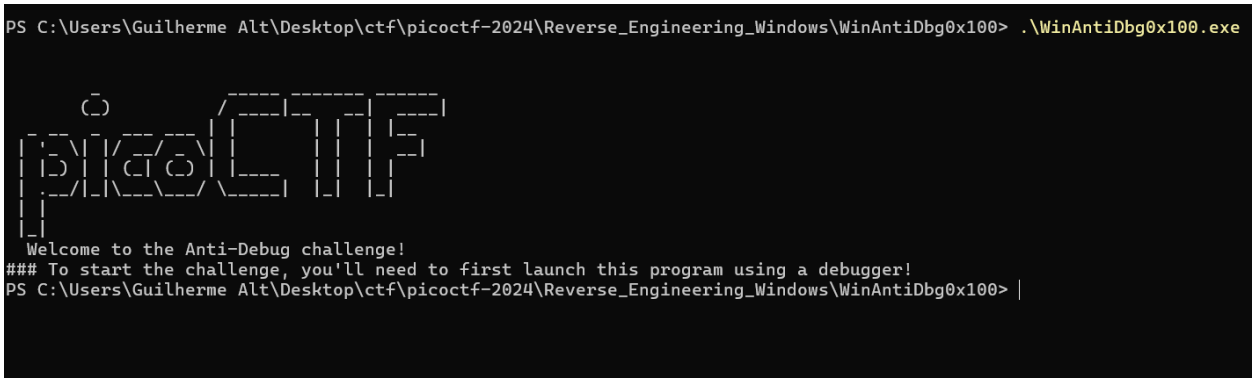

Image 1: running the executable on terminal.

IDA's main view opened on a disassembled graph view of the main function (image 2), so it would be easier to visualize program flow, like "ifs" and "elses".

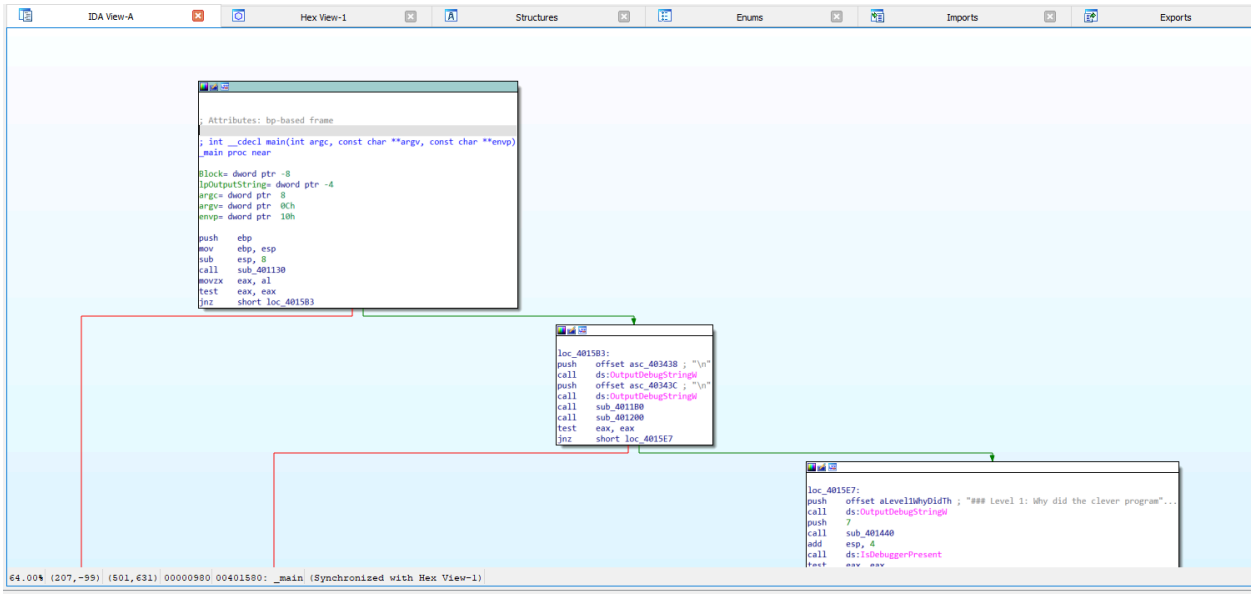

Image 2: Ida GUI (main view).

I took a look at the main function, since it was not so big. The first block already had a branch instruction. One of the branches would take me to a path that eventually prints "To start the challenge, you need to first launch this program using a debugger". The other branch would follow other steps.

I started to debug and set breakpoints to see which path the program flow was following by default. Since I was using a debugger, it did not take the path that tells me to launch the program on a debugger, so it went to the second block on image 2. This block also had a branch instruction. If I took the wrong path, it would eventually print the phrase "could not open the file config.bin…". Fortunately, this path wasn't taken, so I went to the third block (image 3),

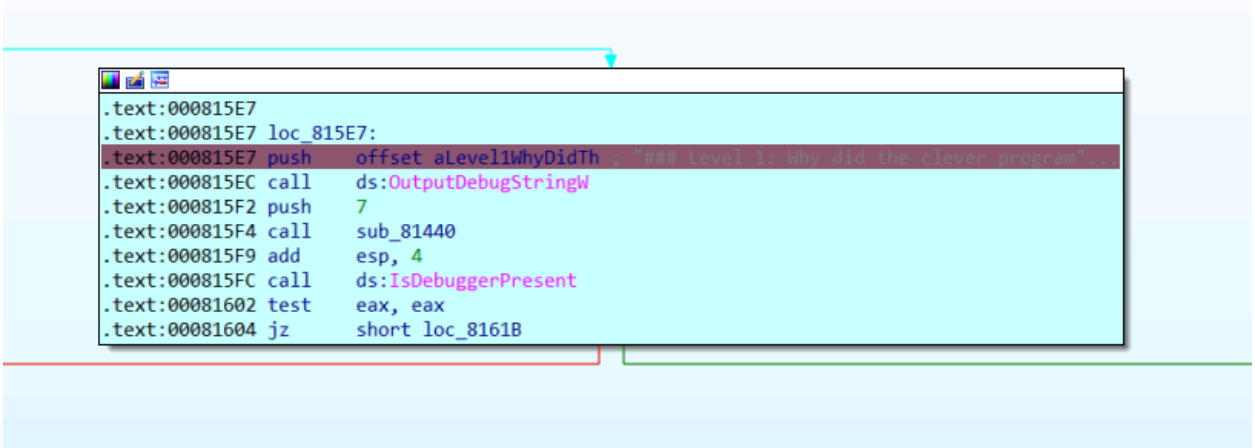

Image 3: Third block of execution flow.

I noticed that this block would call a function named "IsDebuggerPresent", so it could be a problem. However, in x86 based systems, the return value of a function call is usually stored in the eax register. Look at the last 3 assembly lines of this code. The first one calls the IsDebuggerPresent function. The second one does a bitwise AND between eax and itself, and sets the ZERO flag to 1 if this operation results in 0. The third line jumps to "short_loc_8161B" if the ZERO flag is 0.

I made a breakpoint at the jz instruction and saw that the program was about to jump to a location that would eventually print "Oops, the debugger was detected, try to bypass…", so I just manually set the ZERO flag to 1, and the program would now jump to the next block.

I kept doing the same thing, but that was the last problem, from now on all the paths the program took automatically were the correct ones. I set a breakpoint right after an instruction that said "Note: the flag could become corrupted if…", because it would apparently have already printed the flag by that time (see Image 4). I checked the IDA console and there was it.

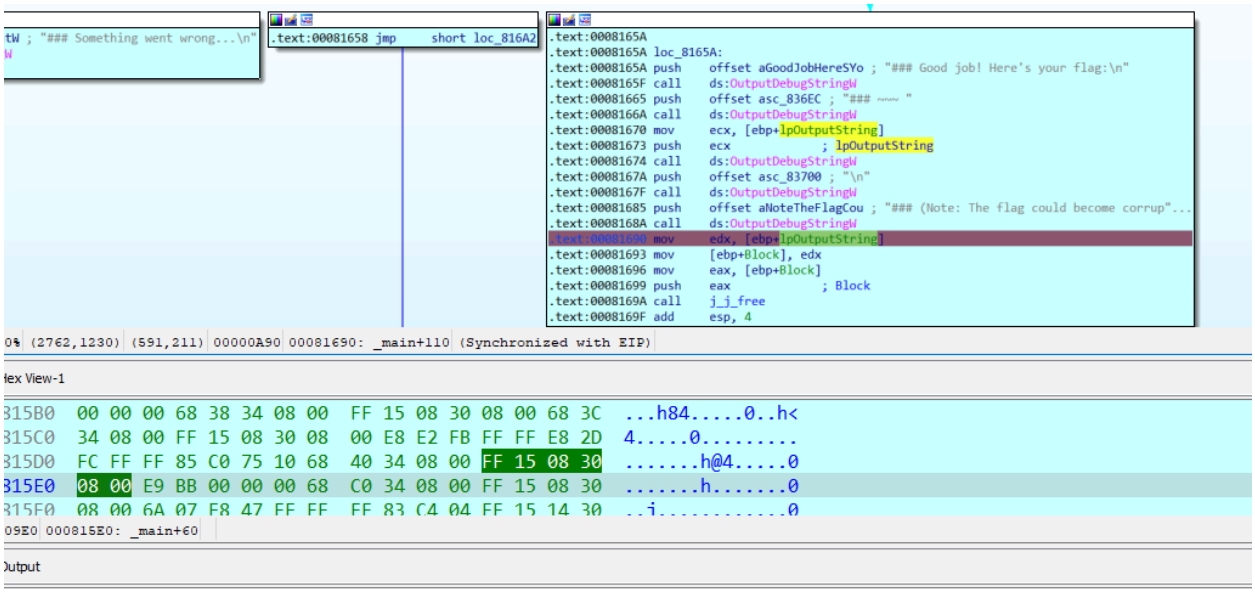

Image 4: flag printed.