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".

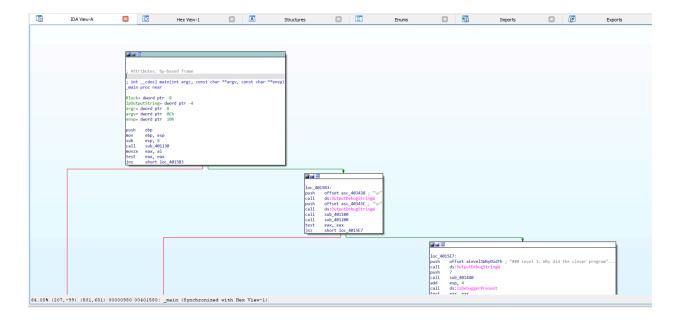


Image 2: Ida GUI (main view).

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.

```
Lext:0008165A loc_8165A:
.text:0008165A loc_8165A:
.text:0008165A push offset aGoodJobHereSYo ; "### Good job! Here's your flag:\n"
.text:0008165F call ds:OutputDebugStringW
.text:0008166A call ds:OutputDebugStringW
.text:0008166A call ds:OutputDebugStringW
                                       .text:00081658 jmp
                                                                                                         ds:OutputDebugStringW
ecx, [ebp+]pOutputString]
ecx
ds:OutputDebugStringW
offset asc_83700; "\n"
ds:OutputDebugStringW
                                                                                  .text:0008166A call
text:00081670 mov
.text:00081673 push
.text:00081674 call
.text:00081674 call
.text:0008167F call
.text:00081685 push
.text:0008168A call
                                                                                                          offset aNoteTheFlagCou; "### (Note: The flag could become corrup".
                                                                                                         edx, [ebp+Block], edx
eax, [ebp+Block] eax
;
j_j_free
                                                                                  .text:00081693 mov
                                                                                  .text:00081696 mov
.text:00081699 push
.text:0008169A call
                                                                                                                          ; Block
                                                                                  .text:0008169F add
                                                                                                         esp, 4
0% (2762,1230) (591,211) 00000A90 00081690: _main+110 (Synchronized with EIP)
315B0 00 00 00 68 38 34 08 00 FF 15 08 30 08 00 68 3C
                                                                                             ...h84.....0..h<
315C0 34 08 00 FF 15 08 30 08 00 E8 E2 FB FF FF E8 2D 4....0.....
315D0 FC FF FF 85 C0 75 10 68 40 34 08 00 <mark>FF 15 08 30</mark>
                                                                                           .....h@4.....0
315E0 08 00 E9 BB 00 00 00 68 C0 34 08 00 FF 15 08 30 .....h.....0
315F0 08 00 6A 07 F8 47 FF FF FF 83 C4 04 FF 15 14 30
                                                                                            ..i.....0
09E0 000815E0: _main+60
Output
nugged application message: ### Level 1: Why did the clever programmer become a gardener? Because they discovered t
nugged application message: ### Good job! Here's your flag:
nugged application message: ### ~~~
ougged application message: picoCTF{d3bug_f0r_th3_Win_0x100_e6c390e2}
nugged application message:
```

Image 4: flag printed.