solution for *elfZ crackme 2* by sghctoma

.intro

First of all, I want to thank elfZ for this crackme. This is exactly the kind of stuff a newbie like me can learn from. I've never encountered the SetUnhandledExceptionFilter trick before, so I had to do a little digging to be able to find the magic word. Digging means learning which is always a good thing to do, so I've enjoyed this crackme a lot.

One more thing: I know it's 6 years old now, and probably lots of people solved it, but it taught me something new. That's why I've decided to write a tutorial about how I solved it. Besides, two of the three submitted solutions use SoftICE, and one of them simply grabs all referenced strings and bruteforces the magic word. In my solution I will show how to do deal with this kind of trick in OllyDbg.

.start

I've loaded the target in Olly, and started it to see what it does. A dialog box appeared, asking for a string. I gave it one, pressed the "check!" button, and the program stopped at 0x0040106B. Hmm, what the frak? The instruction at that address writes EAX to 0x00401113.

At least, it tries to write there. That address is in the .text section, which is non-writeable, so an access violation is generated here. And it is generated on purpose, by the creator of the crackme! The crackme runs fine outside of Olly, so the exception generated by the access violation has to be handled somewhere. It is obvious, that this handler does not run while debugging the program in Olly (because it crashes).

.thetrick

OK, this was a start, but I had no clue, how exception handling actually works (which is a shame btw, because I've used exception handling in my C++ programs, but never thought about how it works on Windows). So I've decided to look further in the code first, and later make some search on exception handling. Well, there is some sort of string compare at 0x00401084, and just after that, there is a JE, which jumps over the badboy message. After that there are several GetDlgItems and ShowWindows. Near the end, at 0x00401108 there is something interesting: a SetUnhandledExceptionFilter API call. A quick search on MSDN reveals this:

SetUnhandledExceptionFilter Function

Enables an application to supersede the top-level exception handler of each thread of a process.

After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the unhandled exception filter, that filter will call the exception filter function specified by the *lpTopLevelExceptionFilter* parameter.

Syntax

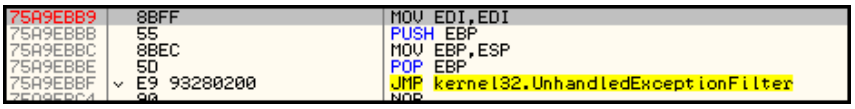
```
LPTOP_LEVEL_EXCEPTION_FILTER WINAPI SetUnhandledExceptionFilter(
  __in  LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter
);
```

After reading the info from MSDN, it was obvious why the program crashes inside Olly, and why it works outside the debugger. The author superseded the top-level exception handler with his/her (sorry elfZ, I have no clue if you are a man or a woman ☺) own function, but this function is called only if the process is not being debugged.

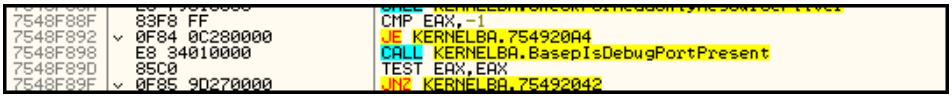
The one and only parameter of the SetUnhandledExceptionFilter API is the address of the exception handler function. I've followed the address (with CTRL+G), and there was definitely something there that looked like a function. I did not know what the values of registers are when this piece of code runs, so I could not determine what it does exactly. I have put a breakpoint on the function's entry point just in case.

I've read lots of things about exception handling while I was trying to solve this crackme, and I've learned something useful: the decision about the process is being debugged or not takes place in Kernel32.dll's UnhandledExceptionFilter function. I've also read that inside UnhandledExceptionFilter, the NtQueryInformationProcess function makes the actual decision.

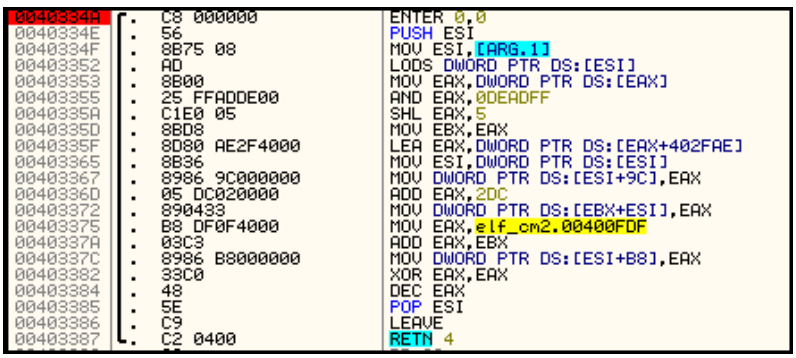
I fired up PEditor to determine the RVA of UnhandledExceptionFilter. I've found that the RVA is 0x0007EBB9. I went back to Olly, pressed ALT+M (Memory map), and I have found that Kernel32.dll is loaded at address 0x75A20000. 0x75A20000 + 0x0007EBB9 = 0x75A9EBB9, so UnhandledExceptionFilter should be at that address. I switched to Kernel32.dll in Olly (CTRL+E, and double click on the dll), and went to 0x75A9EBB9. There was a function epilogue and a JMP there:



The JMP lead me to another JMP, and that JMP lead me where I wanted to go. It turned out, that the function is in KernelBase.dll, btw. OK, as I have said, the decision about the process being debugged or not is in the hand of the NtQueryInformationProcess function. I started to look for it, but I did not find anything like that. I could think of two possible things about the missing NtQueryInformationProcess. Either I am at the wrong place or Windows 7 Beta1 handles this thing differently than XP (everything I've read about exception handling was quite old, so maybe Vista or even XPSP3 does this differently, too). I looked around a bit, and I found an interesting call:



That CALL BasepIsDebugPortPresent looked promising, so I put a hardware breakpoint on it. I let the crackme run, gave it a string, and pushed the "check!" button. Olly stopped on the hardware bp. I stepped over the instruction (F8), so I landed on the TEST EAX, EAX. The value of EAX was 0x00000001, which is indeed means that the process is being run in the context of a debugger. I changed it to 0x00000000 and let the program run. Olly paused at the address 0x0040334A. Remember that address? Yepp, it was the one that the author used with the SetUnhandledExceptionFilter API. That meant that I was at the good place! It looked like this:



I stepped through the code, and found that the LEA at 0x0040335F loads the address of the string "magic" into EAX. I quickly fired up another instance of the crackme, and tried "magic" as the magic word, and it was the right one!

.end

Thank you for reading this tutorial. It is a little bit long I think, but my intention was to explain everything I did, so maybe somebody somewhere can learn from this stuff. I really enjoyed solving this crackme, I hope you enjoyed my tut, as well ☺

Best regards,

sghctoma /*sghctoma@gmail.com*/
January 21, 2009