Malware Analysis
Fall 2015 — Project 3

Maxfield Chen

November 23, 2015

Executive Summary

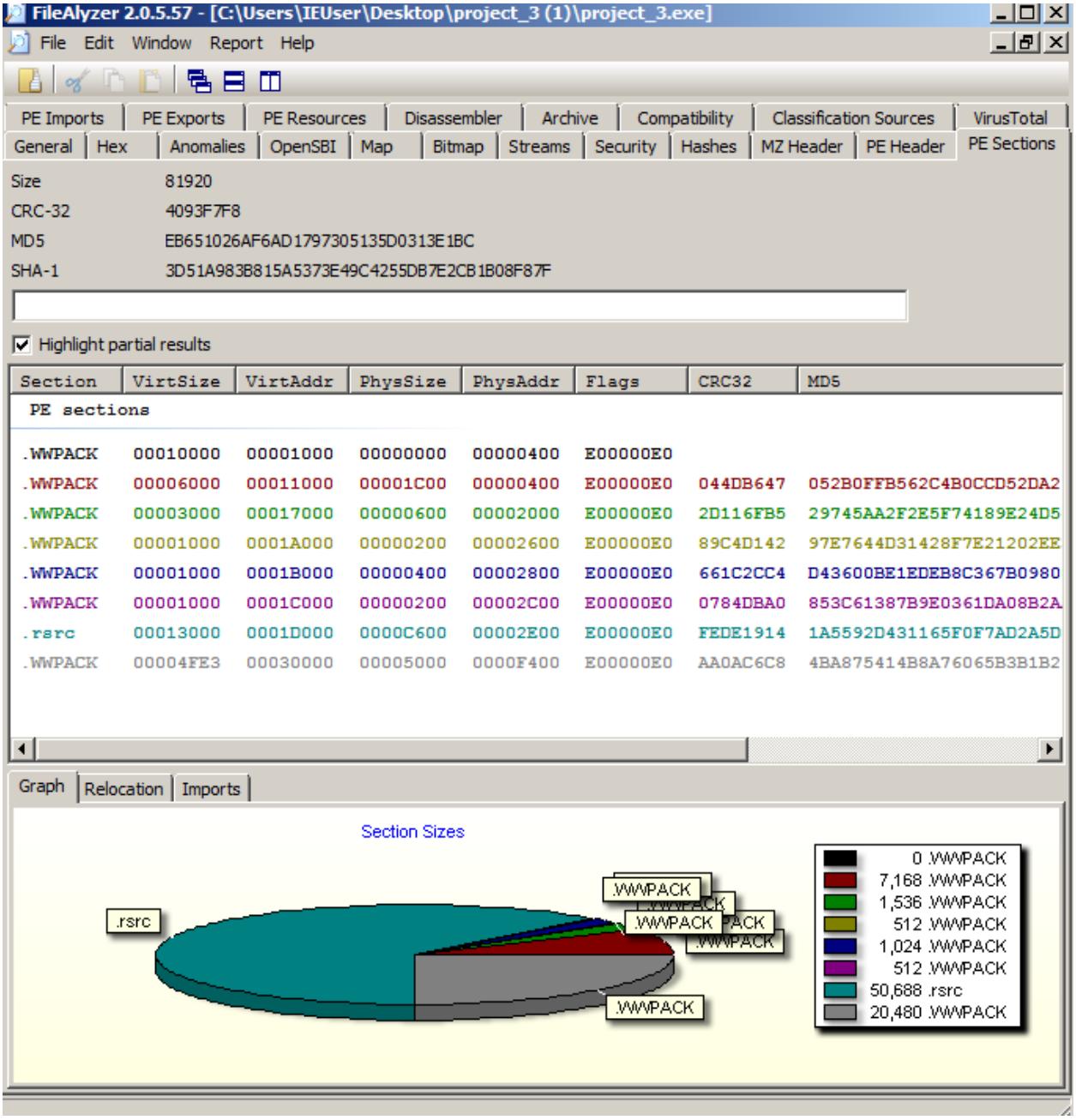
1. This was a triple packed version of freesol with an additional cd key prompt at the end. The first two layers were solved by searching for a function indicated by static analysis(Lock Resource) to find OEP and then dumping the resulting alloced program directly from memory. This dumped program was ImpRec'd. To find second OEP we used run until usercode to be dropped into unpacked code. From there it was simple to locate OEP and fix imports.

Usage Instructions

1. Fire up Ollydbg and load your cryptic game into the debugger. Let's get you playing!
2. Run the stageOne script from the Plugins -¿ OllyScript -¿ load script menu.
3. A message will pop up with further instructions, but here they are again reiterated:
   (a) In the memory window, press control g and enter the address indicated by the first argument on the stack.
   (b) select the first byte at that address and then press control g again and search for the previous address + F600
   (c) Binary copy those bytes. Open HXD and binary paste. Save this as stageTwo.exe
4. Now just run the stageTwo script using the same path as above.
5. Dump Process from here, set modify to FF3EEF.
6. Repair imports in ImpRec using 5438 as your offset and auto settings for the rest.
7. Run the program.
8. ???
9. Fill my bank account with your newly uncluttered and freesol time traveling mind..

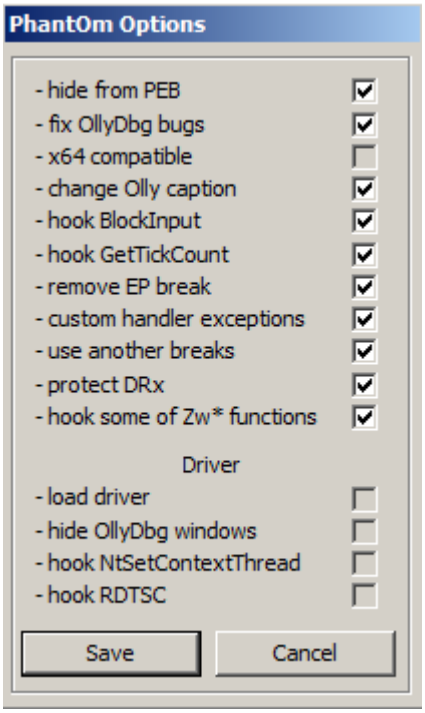
Troubles / Process

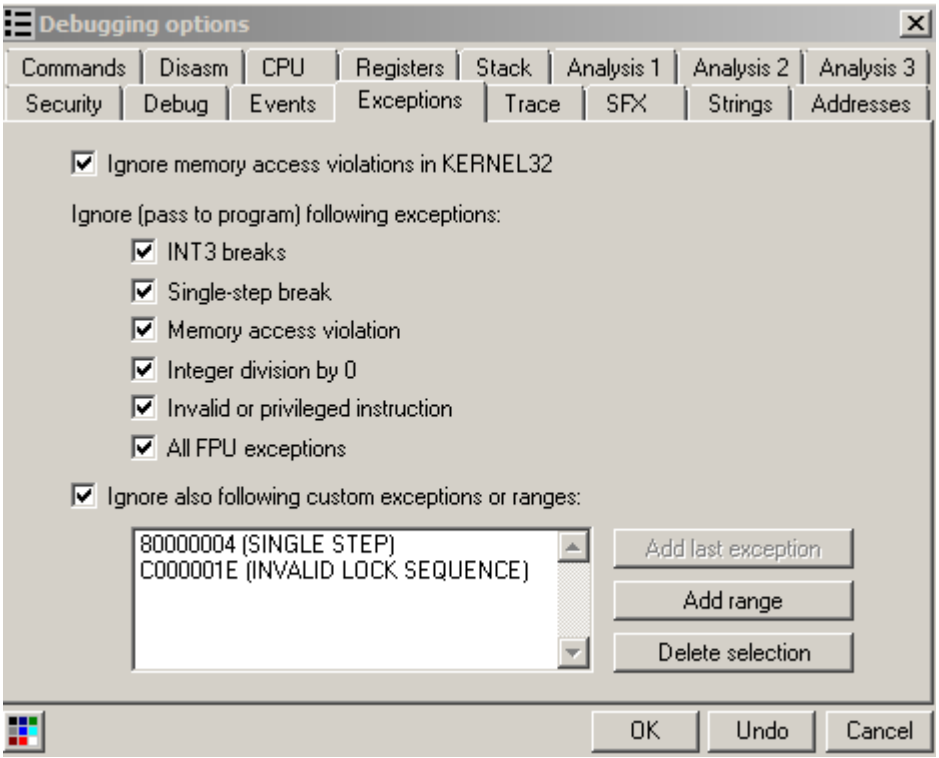
1. I started with some static analysis of the project3.exe file. The PE sections were all scrambled and I found ImportLibrary and getProcAddress as imports which suggested dynamic import construction. I also saw that there was an executable resource which could be dropped. I tried dumping it right away and found it was just gibberish, not a real program. I decided to come back to this mystery after some more analysis.

2. 

3. I proceeded to dynamic analysis. After getting crashes and generally tripped up by anti-

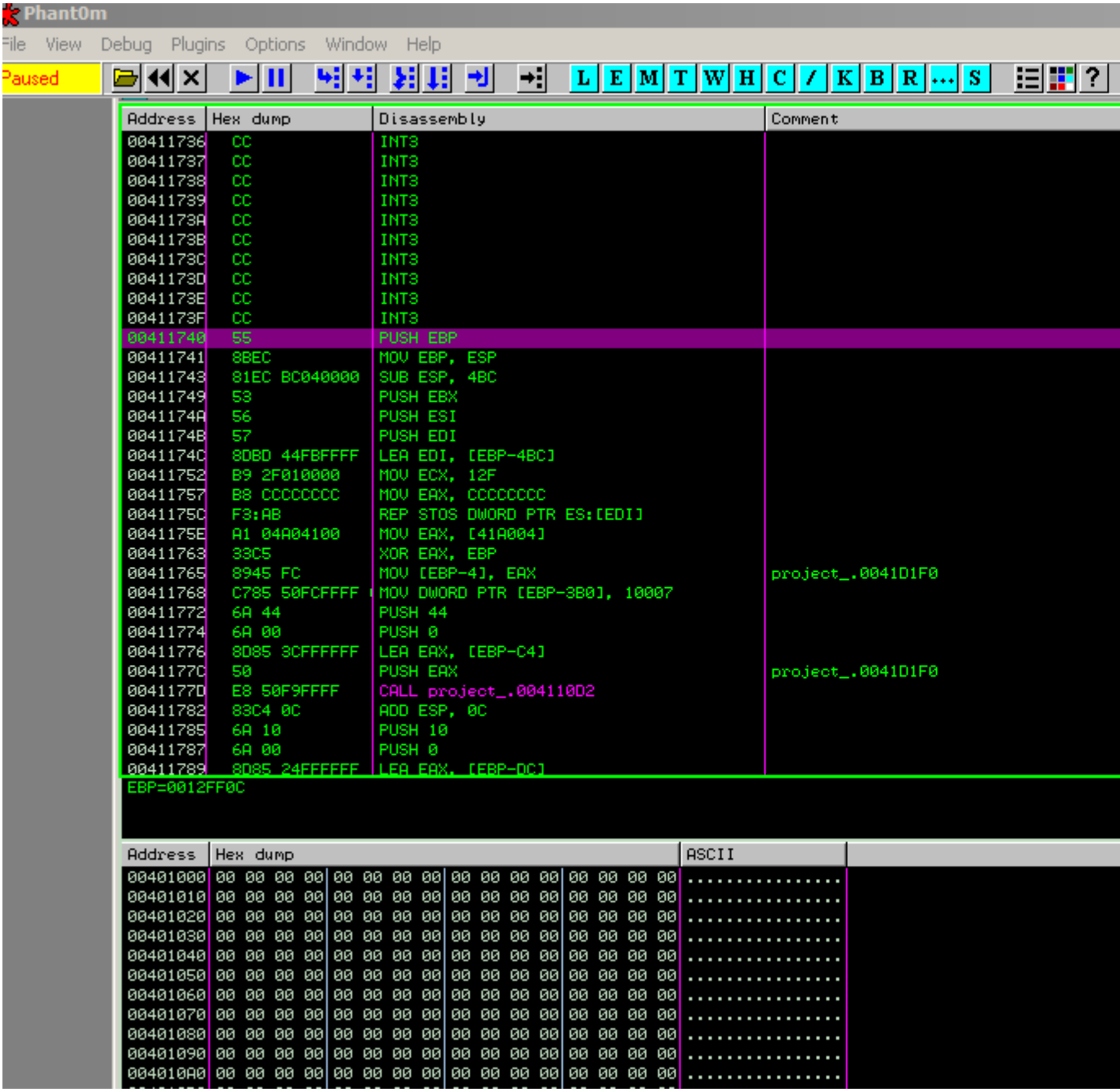
debugging settings, I adopted the following phantom and exception handling strategy:

4. 

5. 

6. I then decided to search for OEP by looking for API calls I knew would have to happen in the original program. Because there was an executable, I assumed that this program

would eventually write it out somewhere. I decided to place a hardware breakpoint on the LockResource call. Tracing up the callstack once this broke allowed me to find the first OEP!

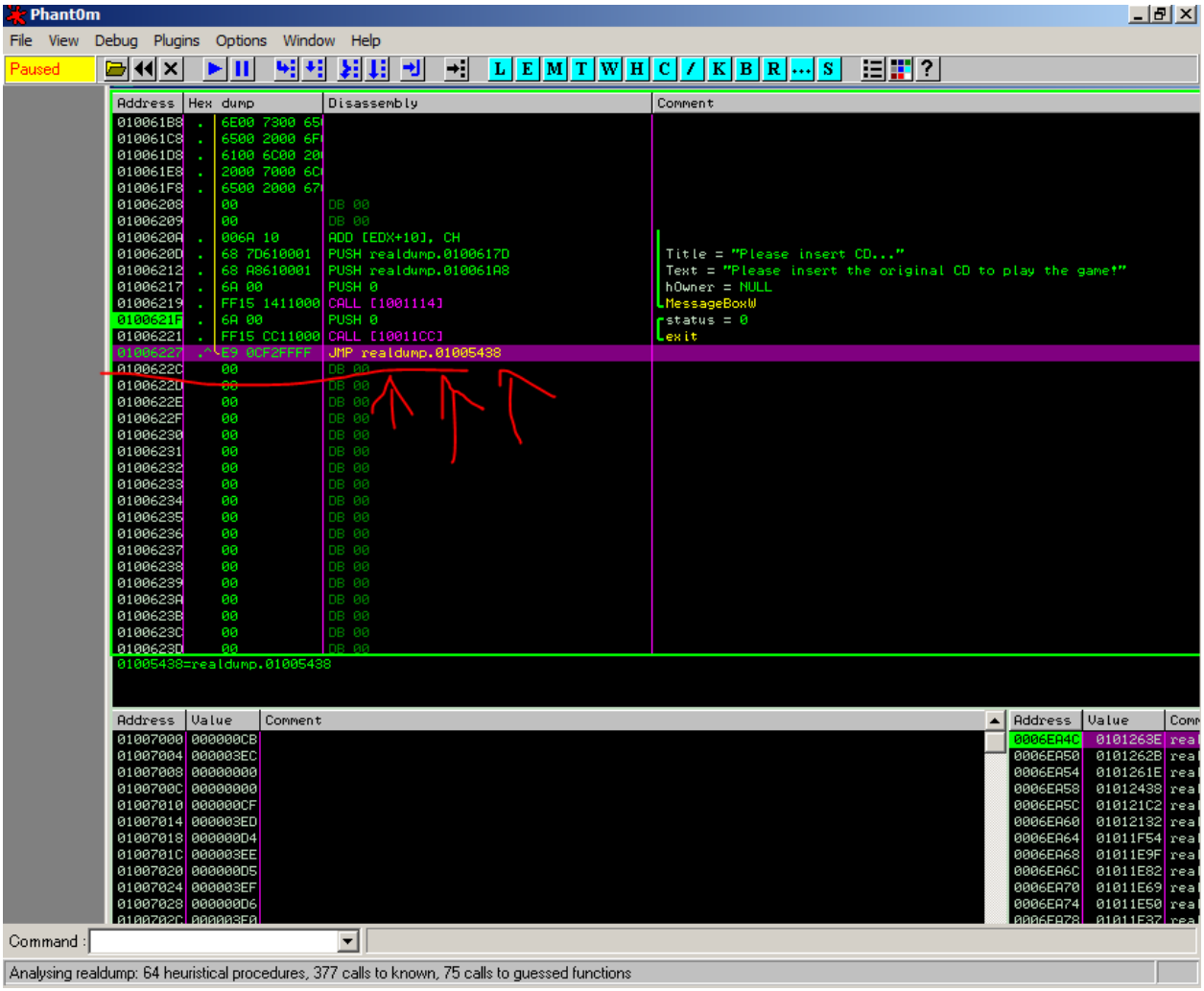
7. 

8. I threw this dump into IDA and found that the droppable was encrypted with a simple xor. Being lazy I broke on VirtualFree and simply dumped the program from memory.

9. Analysis of the resulting dump showed me it was packed with Yoda's Protector 1.03.2 and that it had a .aliens section :)

10. Throwing this new program into olly revealed a slew of gross anti-debugging tricks, from exceptions to sections of code which would completely freeze and or shutdown my VM.

11. Eventually I decided to completely ignore single stepping through this mess and simply run until usercode; knowing that the messagebox would help me out. This allowed me get the jump to the second OEP!



12.

13. It turned out that removing the cd check was as simple as changing the oep! To make sure I had it correct, I actually checked the starting bytes of an unpacked freeSol. Bytes matched up!

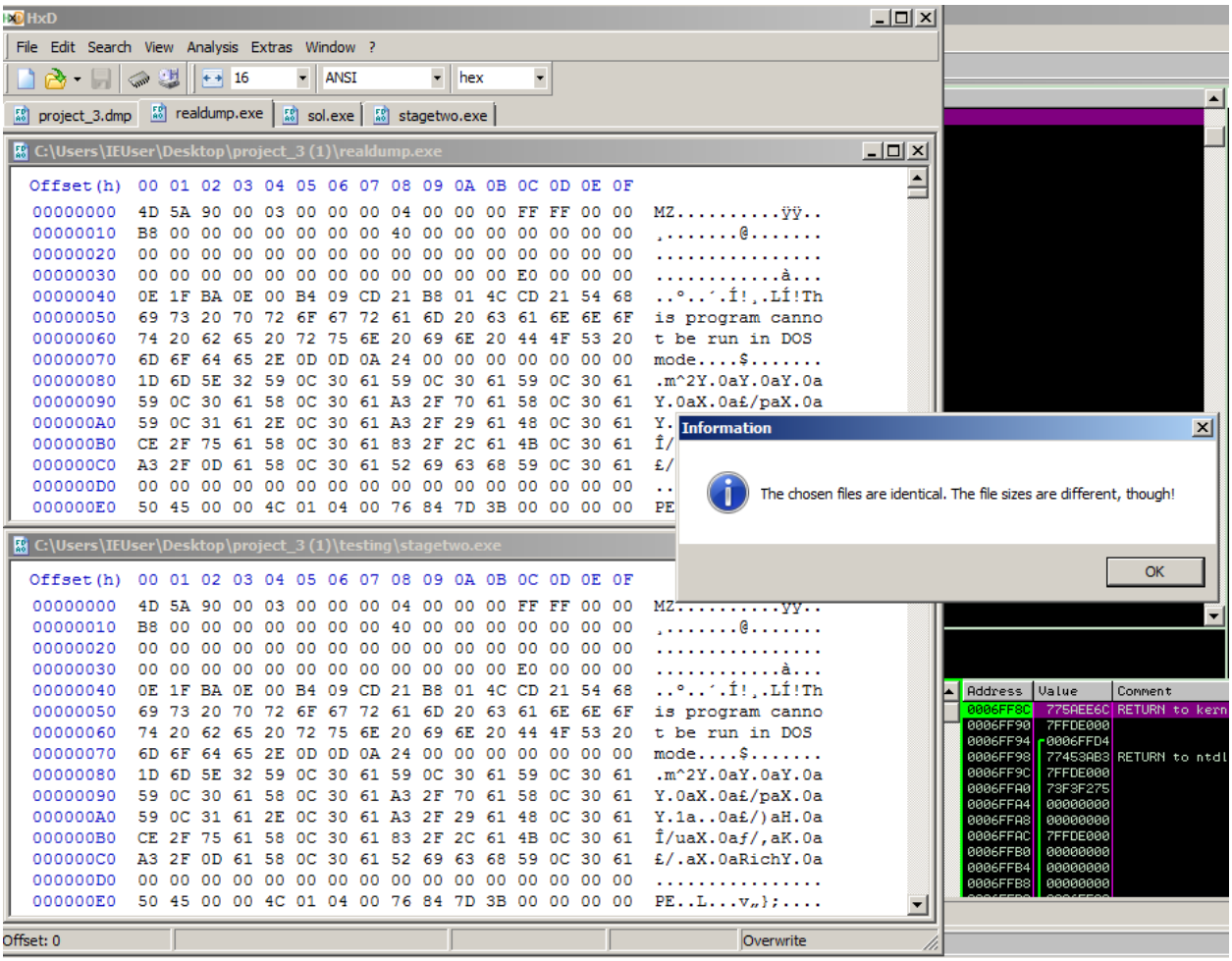
14. I dumped this OEP using ollydump and my newly found OEP(1005438).

15. Then it was a simple matter of using ImpRec to repair the import table using this OEP and suggested settings and the program ran flawlessly!

16. The final piece was adding a signature. I scrolled around the file until I found a code cave and then simply inserted messagebox code and placed the original first instruction at the end

followed by a jump to the second. I then replaced the original instruction with a jump to my code.

17. The last piece of trouble I ran into was during automation. I used ollyscript which ended up working nicely, except I couldn't find a good way to automate dumping from memory or ImpRec, so I had the user execute those manually. I also ran into a replication issue where upon dumping the pe in the first stage, I got a resulting executable which wouldn't run. Having followed the same instructions as before I ran a checksum with my working version:



18.

19. They were exactly the same! (FileSize difference is 0 bytes at the end, I guessed the first time, found exact bytes during automation) Yet when I ran the second version it locked up my whole machine on any kernel32 API call! I couldn't isolate the issue so i'm including my working verison of stagetwo in case the reviewer runs into a similar problem.