SHA256: dc7ab2e7ed26554a11da51a184e95b01e685b1a2f99c7fc77d54d5966530bf60

Analysis

In Figure 1. Below, we check if the file is packed with PeID and Detect-it-Easy. The entropy is low and both applications state that the file is unpacked.

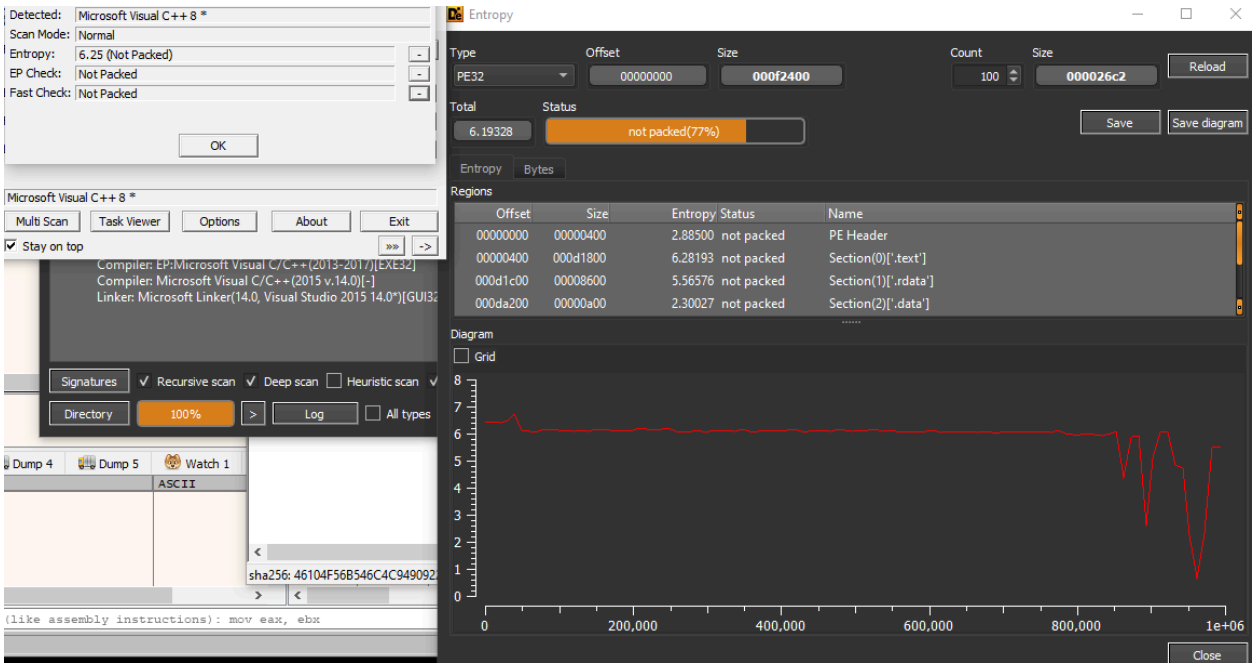


Figure 1. Checking if File is Packed

The next steps taken were to look at the file version and file details. We take a look at the strings as well to find any suspicious strings in the file.

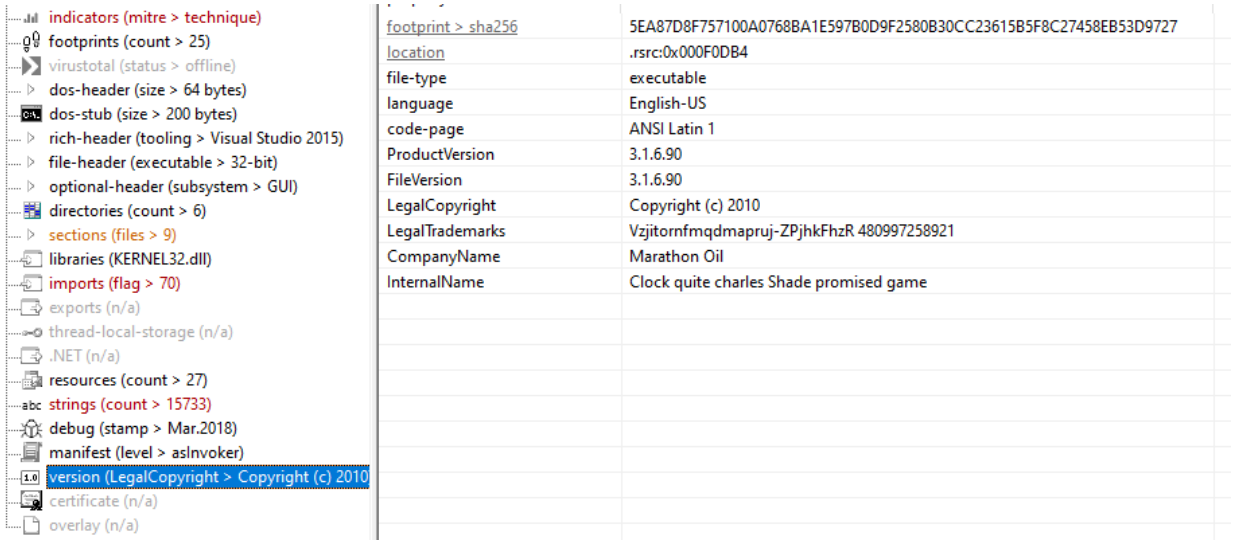


Figure 2. File Versions

In the next Figure below. I took screenshots of some of the strings and imports to gather information on what the file is presumably capable of. It is not included in Figure 4. But IsDebuggerPresent is also included in the imports.

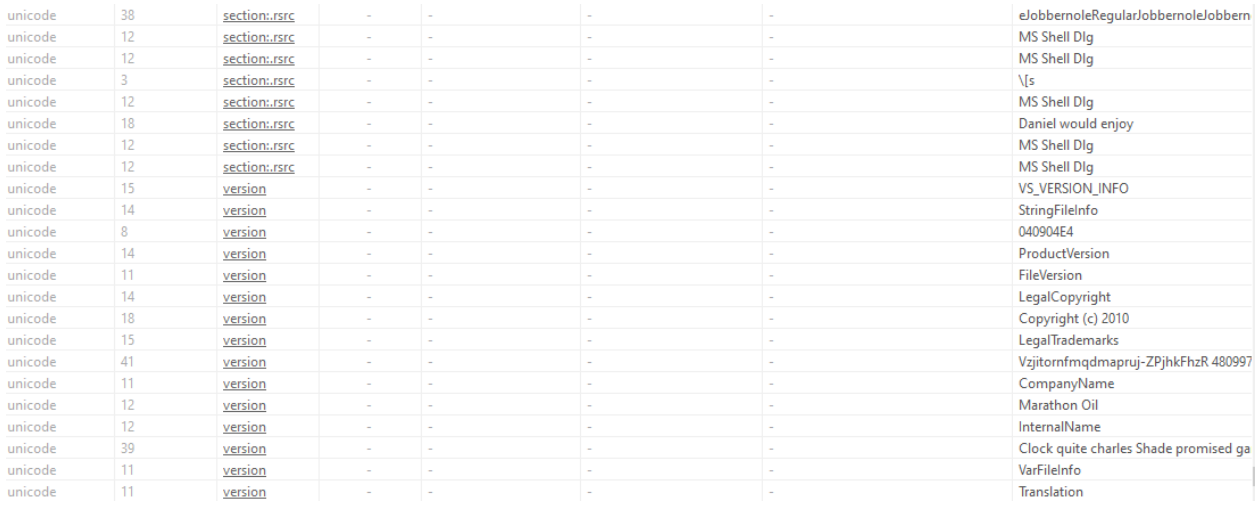


Figure 3. Example of Included Strings.

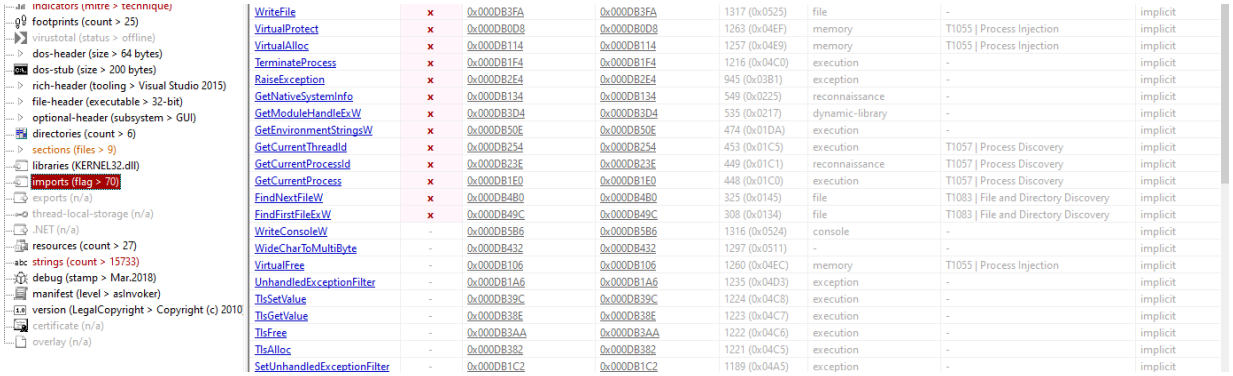


Figure 4. Imports of the File

The next steps of the investigation is to start dynamic analysis and open the file in x32dbg. Figure 5 below shows the breakpoints that I have chosen to set initially for this stage in the analysis.

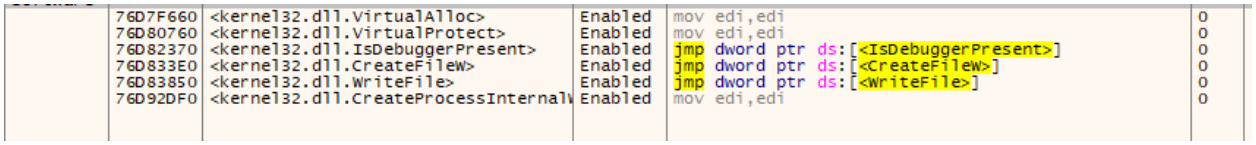


Figure 5. Breakpoints Set

Hitting VirtualAlloc the first time stores 00A80000 on EAX. We then follow this in dump and run again while watching where the memory is being allocated and monitoring what is being allocated as well.

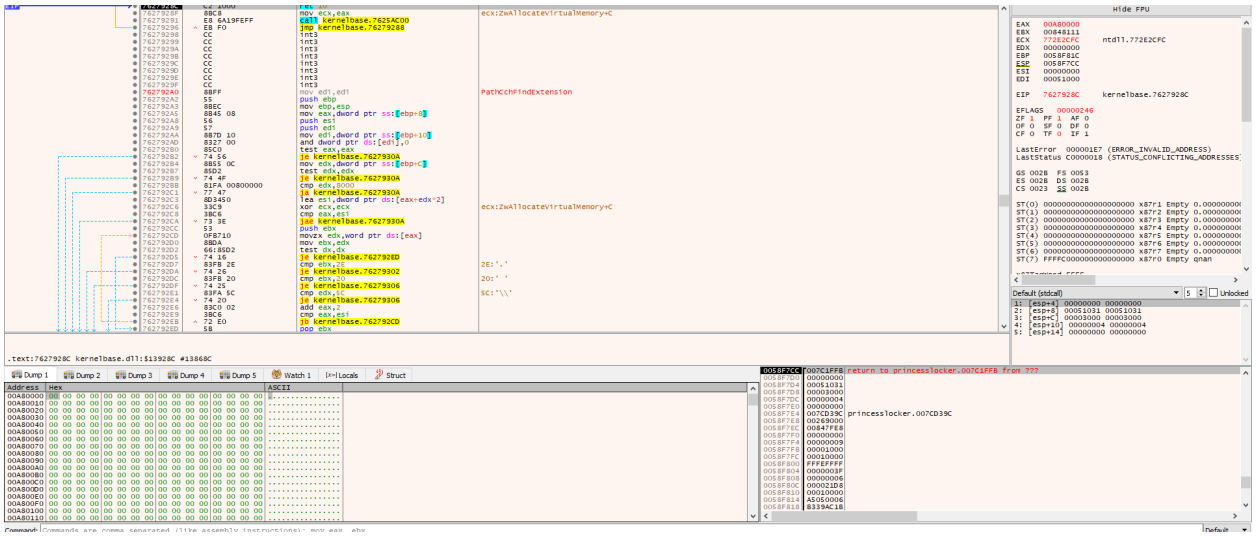


Figure 6. First VirtualAlloc Hit.

After hitting Run again; another VirtualAlloc is hit, this time allocating memory into 00A81000. The content of the allocated memory is interesting as there is now something that looks like shellcode as well as section headers somewhere below.

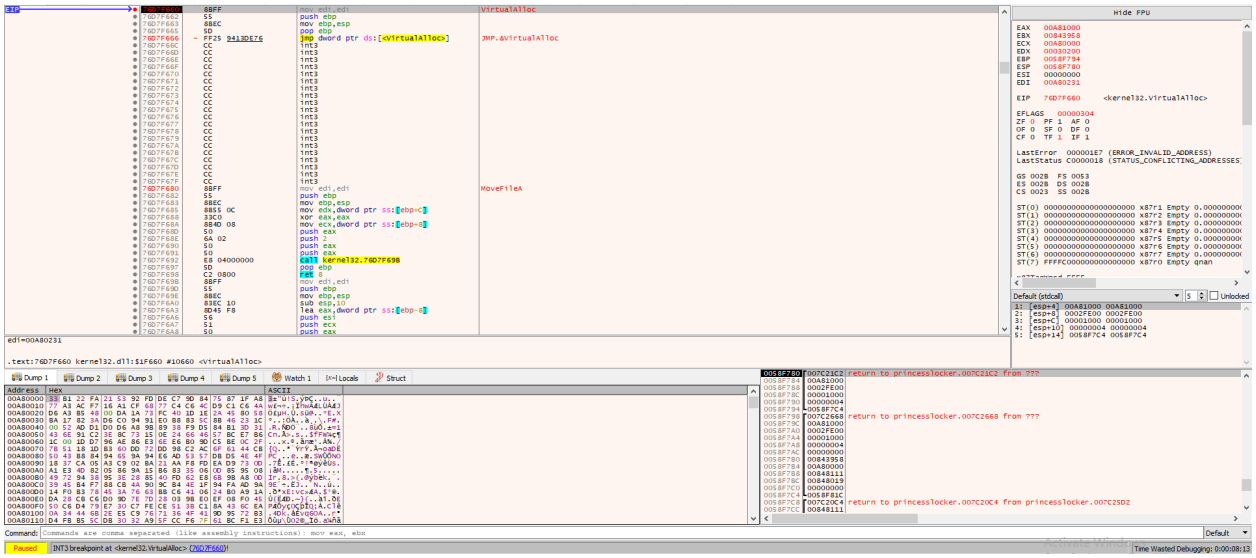
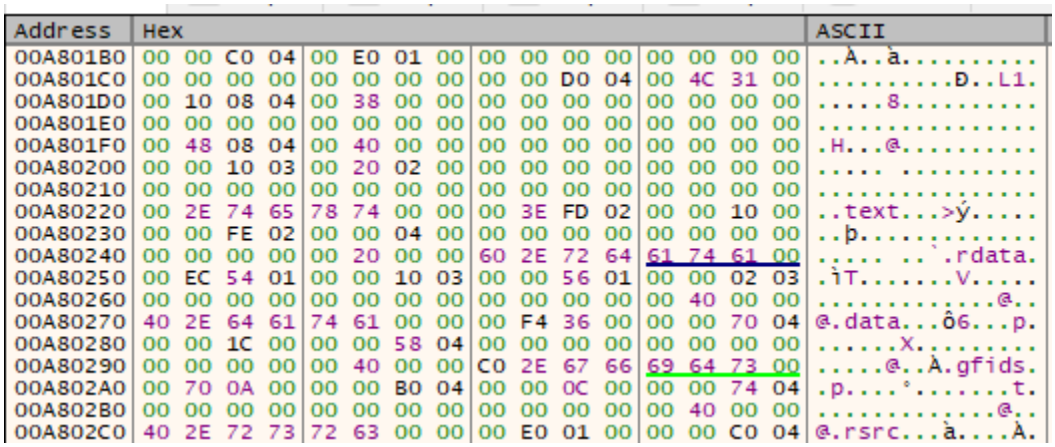


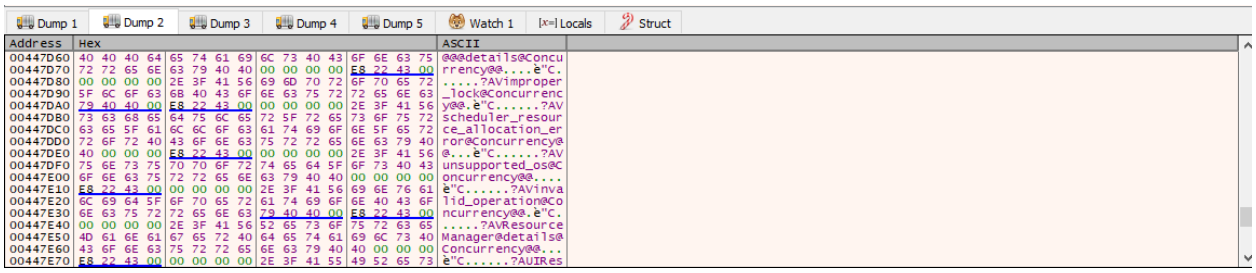
Figure 6. Data Stored in Memory



Address	Hex	ASCII
00A801B0	00 00 C0 04 00 E0 01 00 00 00 00 00 00 00 00 00	..À..à..........
00A801C0	00 00 00 00 00 00 00 00 00 00 D0 04 00 4C 31 00	..........Ð..L1.
00A801D0	00 10 08 04 00 38 00 00 00 00 00 00 00 00 00 00	.....8..........
00A801E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00A801F0	00 48 08 04 00 40 00 00 00 00 00 00 00 00 00 00	.H...@..........
00A80200	00 00 10 03 00 20 02 00 00 00 00 00 00 00 00 00	....
00A80210	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00A80220	00 2E 74 65 78 74 00 00 00 3E FD 02 00 00 10 00	..text...>ý.....
00A80230	00 00 FE 02 00 00 04 00 00 00 00 00 00 00 00 00	..þ..............
00A80240	00 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00	....`.rdata.
00A80250	00 EC 54 01 00 00 10 03 00 00 56 01 00 00 02 03	.ìT.......V.....
00A80260	00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00	.............@..
00A80270	40 2E 64 61 74 61 00 00 00 F4 36 00 00 00 70 04	@.data...ô6...p.
00A80280	00 00 1C 00 00 00 58 04 00 00 00 00 00 00 00 00	......X.........
00A80290	00 00 00 00 00 40 00 00 C0 2E 67 66 69 64 73 00	.....@..À.gfids.
00A802A0	00 70 0A 00 00 00 B0 04 00 00 0C 00 00 00 74 04	.p....°.......t.
00A802B0	00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00	.............@..
00A802C0	40 2E 72 73 72 63 00 00 00 E0 01 00 00 00 C0 04	@.rsrc...à....À.

Figure 7. Section Headers in Allocated Memory

Repeating this process of following VirtualAlloc until more data is allocated causes the process to interestingly self terminate. I redid the process and dumped up the memory before it self terminates since it seems to already have the data allocated in the address anyway and we'll see what happens.



Address	Hex	ASCII
00447D60	40 40 40 64 65 74 61 69 6C 73 40 43 6F 6E 63 75	@@@details@Concu
00447D70	72 72 65 6E 63 79 40 40 00 00 00 00 E8 22 43 00	rrency@@....è"C.
00447D80	00 00 00 00 2E 3F 41 56 69 6D 70 72 6F 70 65 72	.....?AVimproper
00447D90	5F 6C 6F 63 6B 40 43 6F 6E 63 75 72 72 65 6E 63	_lock@Concurrenc
00447DA0	79 40 40 00 E8 22 43 00 00 00 00 00 2E 3F 41 56	y@@.è"C......?AV
00447DB0	73 63 68 65 64 75 6C 65 72 5F 72 65 73 6F 75 72	scheduler_resour
00447DC0	63 65 5F 61 6C 6C 6F 63 61 74 69 6F 6E 5F 65 72	ce_allocation_er
00447DD0	72 6F 72 40 43 6F 6E 63 75 72 72 65 6E 63 79 40	ror@Concurrency@
00447DE0	40 00 00 00 E8 22 43 00 00 00 00 00 2E 3F 41 56	@...è"C......?AV
00447DF0	75 6E 73 75 70 70 6F 72 74 65 64 5F 6F 73 40 43	unsupported_os@C
00447E00	6F 6E 63 75 72 72 65 6E 63 79 40 40 00 00 00 00	oncurrency@@....
00447E10	E8 22 43 00 00 00 00 00 2E 3F 41 56 69 6E 76 61	è"C......?AVinva
00447E20	6C 69 64 5F 6F 70 65 72 61 74 69 6F 6E 40 43 6F	lid_operation@Co
00447E30	6E 63 75 72 72 65 6E 63 79 40 40 00 E8 22 43 00	ncurrency@@.è"C.
00447E40	00 00 00 00 2E 3F 41 56 52 65 73 6F 75 72 63 65	.....?AVResource
00447E50	4D 61 6E 61 67 65 72 40 64 65 74 61 69 6C 73 40	Manager@details@
00447E60	43 6F 6E 63 75 72 72 65 6E 63 79 40 40 00 00 00	Concurrency@@...
00447E70	E8 22 43 00 00 00 00 00 2E 3F 41 55 49 52 65 73	è"C......?AUIRes

Figure 8. Example of Strings on Allocated Memory

As the dumped file will not be interpretable by any of our tools since the MZ header is missing/corrupted it needs to be fixed by replacing the corrupted header on the dumped binary with a proper header from a file that we know is working. The size of the replaced header may either be larger or smaller which causes the other sections to misalign which will need to be fixed.

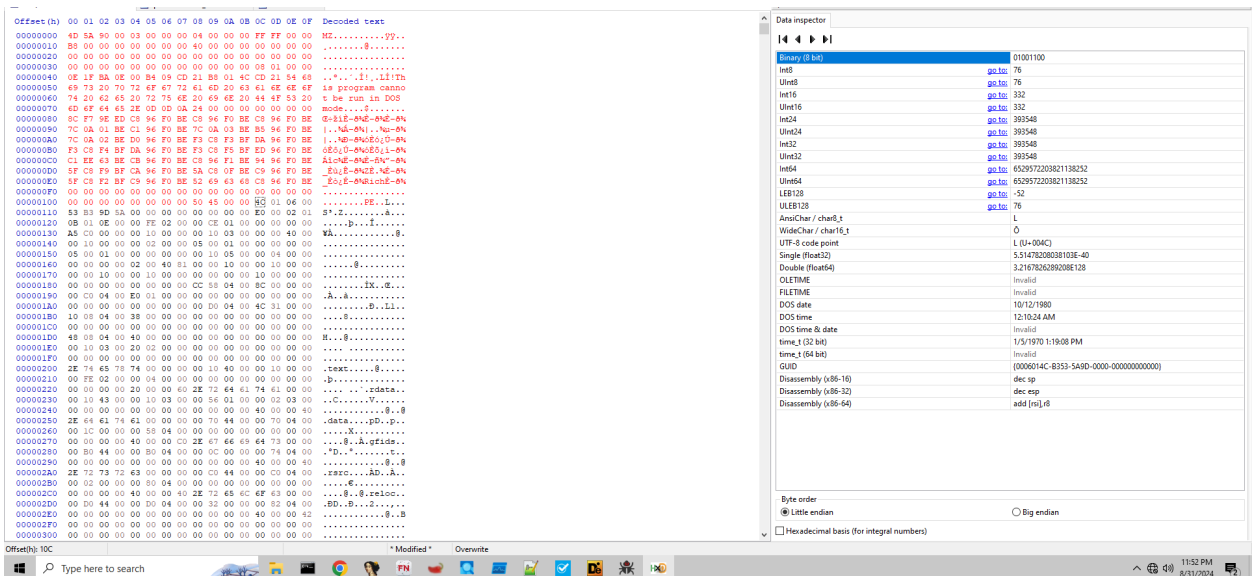


Figure 9. Replacing corrupted MZ Header.

Figure 10. Shows the section headers after opening the fixed dump on PEBear.

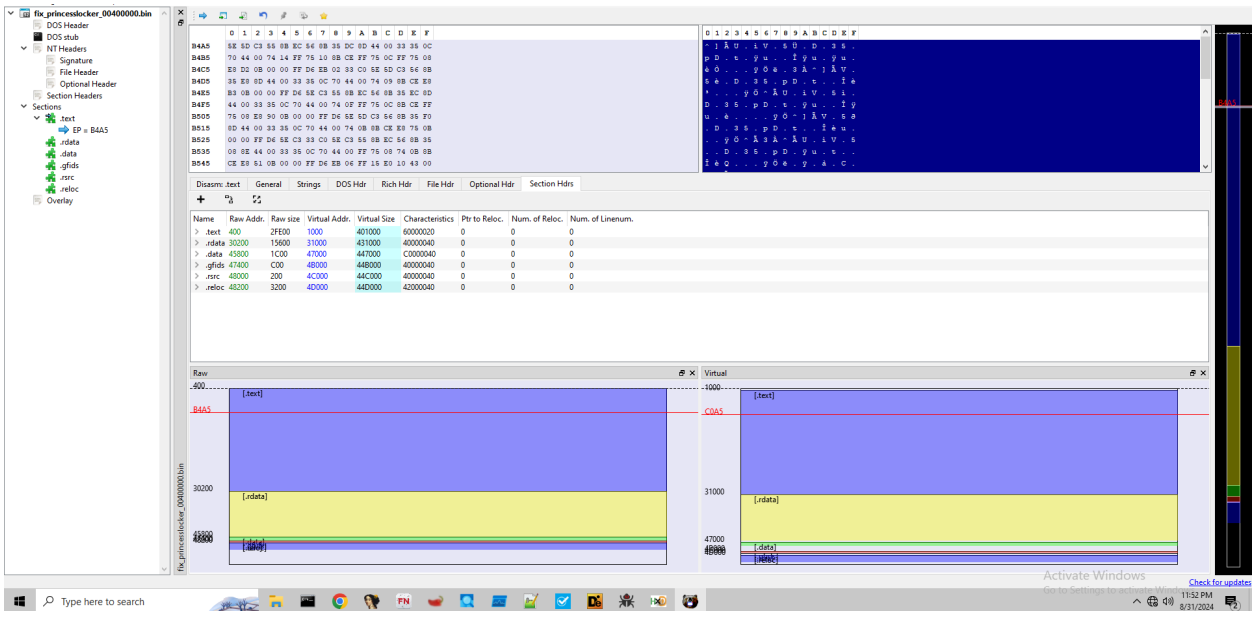


Figure 10. Section Headers on PE-Bear

We then change the raw address to match the virtual address as we dumped a mapped file and the raw address and virtual address should be the same. The raw size is then calculated by subtracting

the raw addresses.

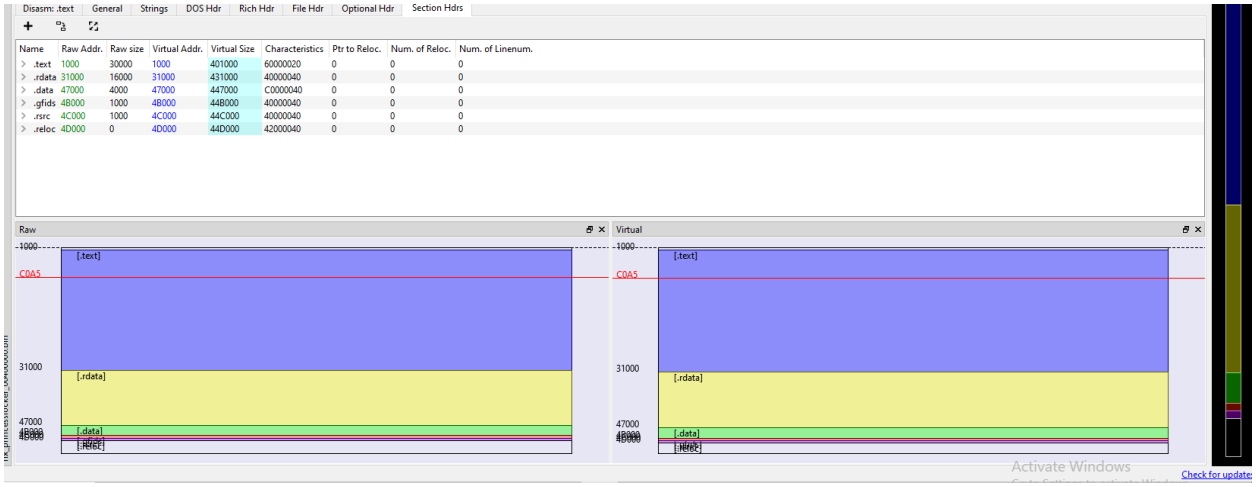


Figure 11. Fixed Section Headers

After unmapping the file, it is then saved and opened up again on a Hex editor to edit and align the sections on what was set on the PE-Bear. To do this the data needs to be adjusted to the address of 00001000h. This is done by padding null bytes. Each horizontal row is 16 bytes and therefore we need to add 33 bytes.

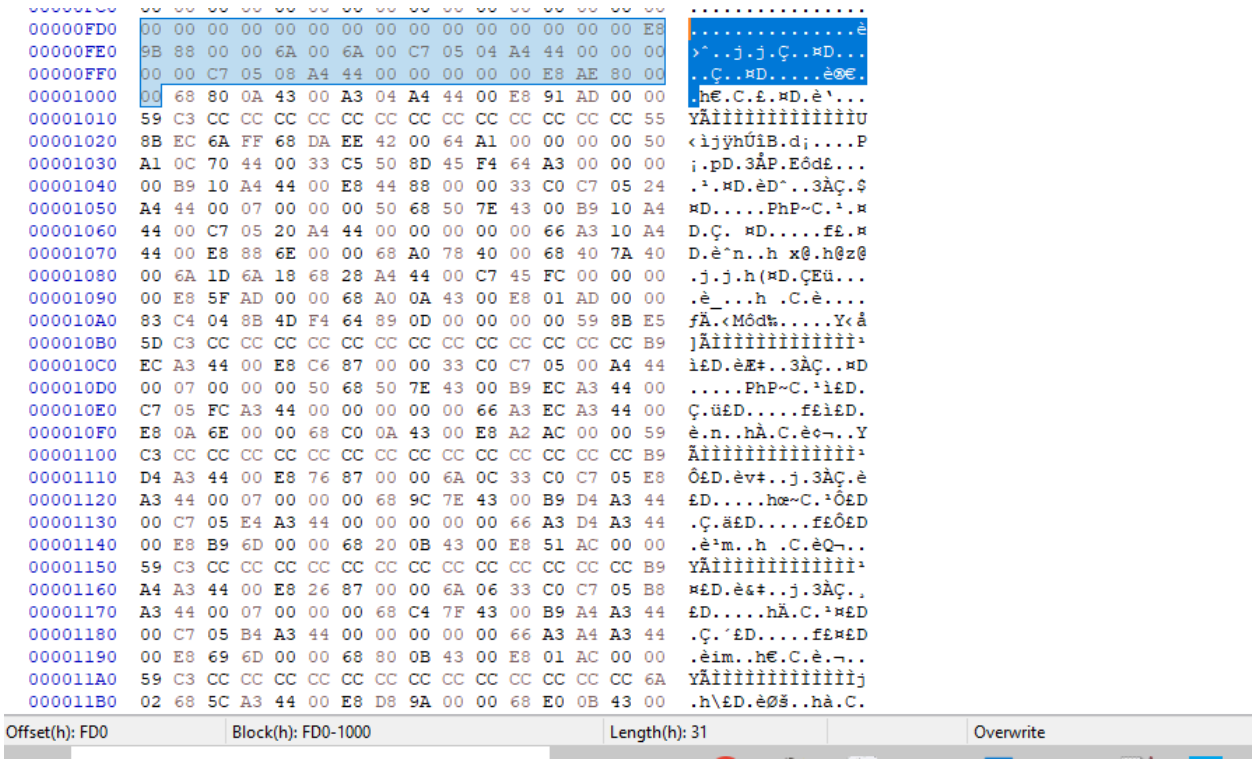


Figure 12. Realigning Section Headers

We can now open the unpacked file on IDA or Ghidra and compare the imports and exports of the unpacked payload compared with the old one. Figure 13 shows the packed one and Figure 14 shows the unpacked one.

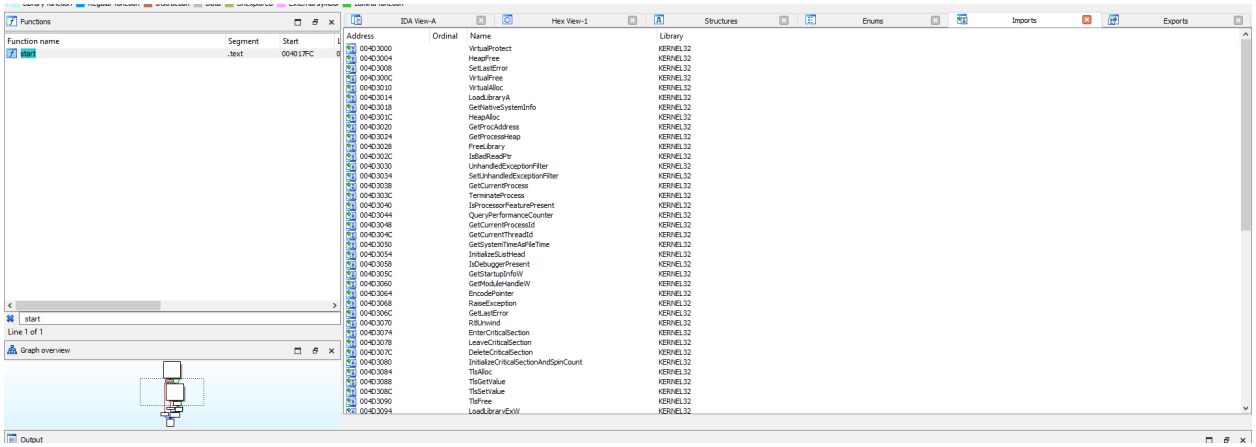


Figure 13. Packed Exports

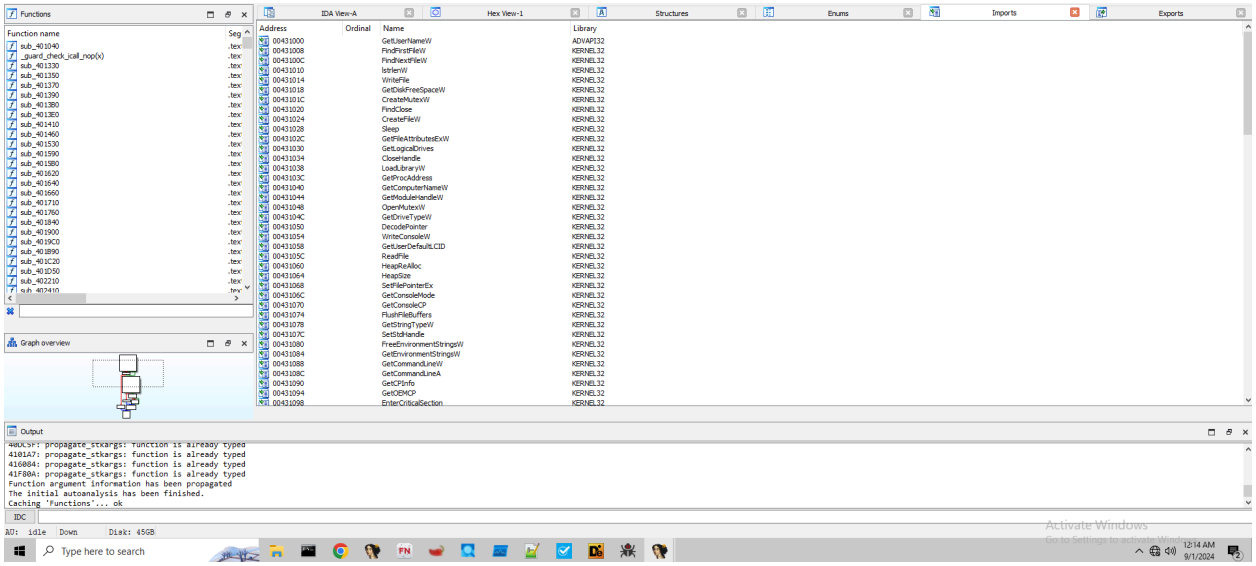


Figure 14. Unpacked Exports

After unpacking this the next steps I plan to do is to examine the payload on either IDA or Ghidra or run it in a debugger to trace its functionality.