

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.

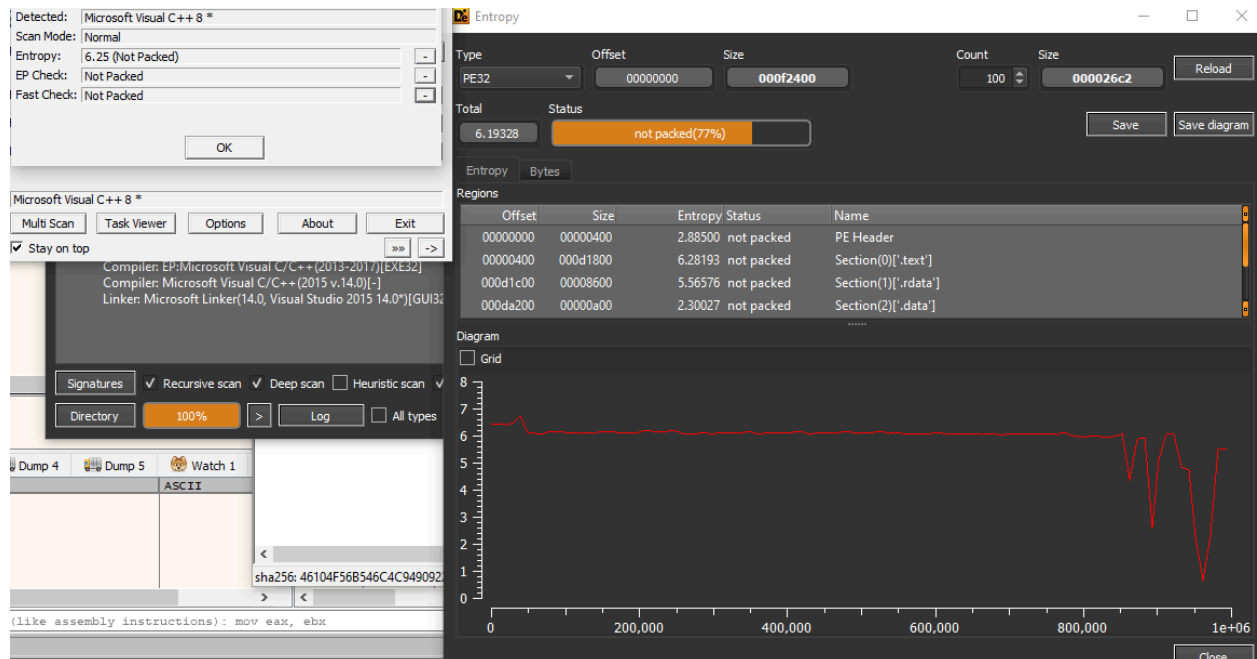


Figure 1. Checking if File is Packed

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.

indicators (mitre > technique)	footprint > sha256	5EA87D8F757100A0768BA1E597B0D9F2580B30CC23615B5F8C27458EB53D9727
footprints (count > 25)	location	.rsrc:0x000F0DB4
virustotal (status > offline)	file-type	executable
> dos-header (size > 64 bytes)	language	English-US
dos-stub (size > 200 bytes)	code-page	ANSI Latin 1
> rich-header (tooling > Visual Studio 2015)	ProductVersion	3.1.6.90
> file-header (executable > 32-bit)	FileVersion	3.1.6.90
> optional-header (subsystem > GUI)	LegalCopyright	Copyright (c) 2010
directories (count > 6)	LegalTrademarks	Vzjitornfmqdmpruj-ZPjhkFhzR 480997258921
> sections (files > 9)	CompanyName	Marathon Oil
libraries (KERNEL32.dll)	InternalName	Clock quite charles Shade promised game
imports (flag > 70)		
exports (n/a)		
thread-local-storage (n/a)		
.NET (n/a)		
resources (count > 27)		
strings (count > 15733)		
debug (stamp > Mar.2018)		
manifest (level > aslnvoker)		
version (LegalCopyright > Copyright (c) 2010)		
certificate (n/a)		
overlay (n/a)		

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.

unicode	38	section:rsrc	-	-	-	-	eJobbernoleRegularJobbernoleJobbern
unicode	12	section:rsrc	-	-	-	-	MS Shell Dlg
unicode	12	section:rsrc	-	-	-	-	MS Shell Dlg
unicode	3	section:rsrc	-	-	-	-	\[s
unicode	12	section:rsrc	-	-	-	-	MS Shell Dlg
unicode	18	section:rsrc	-	-	-	-	Daniel would enjoy
unicode	12	section:rsrc	-	-	-	-	MS Shell Dlg
unicode	12	section:rsrc	-	-	-	-	MS Shell Dlg
unicode	15	version	-	-	-	-	VS_VERSION_INFO
unicode	14	version	-	-	-	-	StringFileInfo
unicode	8	version	-	-	-	-	040904E4
unicode	14	version	-	-	-	-	ProductVersion
unicode	11	version	-	-	-	-	FileVersion
unicode	14	version	-	-	-	-	LegalCopyright
unicode	18	version	-	-	-	-	Copyright (c) 2010
unicode	15	version	-	-	-	-	LegalTrademarks
unicode	41	version	-	-	-	-	Vzjitornfmqdmpruj-ZPjhkFhzR 480997
unicode	11	version	-	-	-	-	CompanyName
unicode	12	version	-	-	-	-	Marathon Oil
unicode	12	version	-	-	-	-	InternalName
unicode	39	version	-	-	-	-	Clock quite charles Shade promised ga
unicode	11	version	-	-	-	-	VarFileInfo
unicode	11	version	-	-	-	-	Translation

Figure 3. Example of Included Strings.

00	footprints (count > 25)	WriteFile	x	0x000B3FA	0x000B3FA	1317 (0x0525)	file	-	implicit
01	virustotal (status > offline)	VirtualProtect	x	0x000B008	0x000B008	1263 (0x04EF)	memory	T1055 Process Injection	implicit
02	dos-header (size > 64 bytes)	VirtualAlloc	x	0x000B114	0x000B114	1257 (0x04E9)	memory	T1055 Process Injection	implicit
03	dos-stub (size > 200 bytes)	TerminateProcess	x	0x000B1F4	0x000B1F4	1216 (0x04C0)	execution	-	implicit
04	rich-header (tooling > Visual Studio 2015)	BaseException	x	0x000B2E4	0x000B2E4	945 (0x03B1)	exception	-	implicit
05	file-header (executable > 32-bit)	GetNativeSystemInfo	x	0x000B134	0x000B134	549 (0x0225)	reconnaissance	-	implicit
06	optional-header (subsystem > GUI)	GetModuleHandleExW	x	0x000B3D4	0x000B3D4	535 (0x0217)	dynamic-library	-	implicit
07	directories (count > 6)	GetCurrentThread	x	0x000B50E	0x000B50E	474 (0x01DA)	execution	-	implicit
08	sections (files > 9)	GetCurrentProcess	x	0x000B254	0x000B254	453 (0x01C5)	execution	T1057 Process Discovery	implicit
09	libraries (KERNEL32.dll)	GetCurrentProcessId	x	0x000B23E	0x000B23E	449 (0x01C1)	reconnaissance	T1057 Process Discovery	implicit
10	imports (flag > 70)	FindNextFileW	x	0x000B1E0	0x000B1E0	448 (0x01C0)	execution	T1057 Process Discovery	implicit
11	exports (n/a)	FindFirstFileW	x	0x000B480	0x000B480	325 (0x0145)	file	T1083 File and Directory Discovery	implicit
12	thread-local-storage (n/a)	WriteConsoleW	-	0x000B49C	0x000B49C	308 (0x0134)	file	T1083 File and Directory Discovery	implicit
13	.NET (n/a)	WideCharToMultiByte	-	0x000B586	0x000B586	1316 (0x0524)	console	-	implicit
14	resources (count > 27)	VirtualFree	-	0x000B432	0x000B432	1297 (0x0511)	-	-	implicit
15	strings (count > 15733)	UnhandledExceptionFilter	-	0x000B106	0x000B106	1260 (0x04EC)	memory	T1055 Process Injection	implicit
16	debug (stamp > Mar.2018)	TlsSetValue	-	0x000B1A6	0x000B1A6	1235 (0x04D3)	exception	-	implicit
17	manifest (level > asnInvoker)	TlsGetValue	-	0x000B39C	0x000B39C	1224 (0x04C8)	execution	-	implicit
18	version (LegalCopyright > Copyright (c) 2010)	TlsFree	-	0x000B38E	0x000B38E	1223 (0x04C7)	execution	-	implicit
19	certificate (n/a)	TlsAlloc	-	0x000B3AA	0x000B3AA	1222 (0x04C6)	execution	-	implicit
20	overlay (n/a)	SetUnhandledExceptionFilter	-	0x000B382	0x000B382	1221 (0x04C5)	execution	-	implicit
21			-	0x000B1C2	0x000B1C2	1189 (0x04A5)	exception	-	implicit

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.

76D7F660	<kernel32.dll.VirtualAlloc>	Enabled	mov edi,edi	0
76D80760	<kernel32.dll.VirtualProtect>	Enabled	mov edi,edi	0
76D82370	<kernel32.dll.IsDebuggerPresent>	Enabled	jmp dword ptr ds:[<IsDebuggerPresent>]	0
76D833E0	<kernel32.dll.CreateFileW>	Enabled	jmp dword ptr ds:[<CreateFileW>]	0
76D83850	<kernel32.dll.WriteFile>	Enabled	jmp dword ptr ds:[<WriteFile>]	0
76D92DF0	<kernel32.dll.CreateProcessInternalW>	Enabled	mov edi,edi	0

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.

The screenshot shows the x32dbg interface. The CPU registers window on the right displays EAX = 00A80000. The dump window on the bottom left shows the memory dump starting at 00A80000, which contains shellcode and section headers. The command window on the bottom right shows the instruction 'return to princesslocker.007C3F8 from 777'.

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.

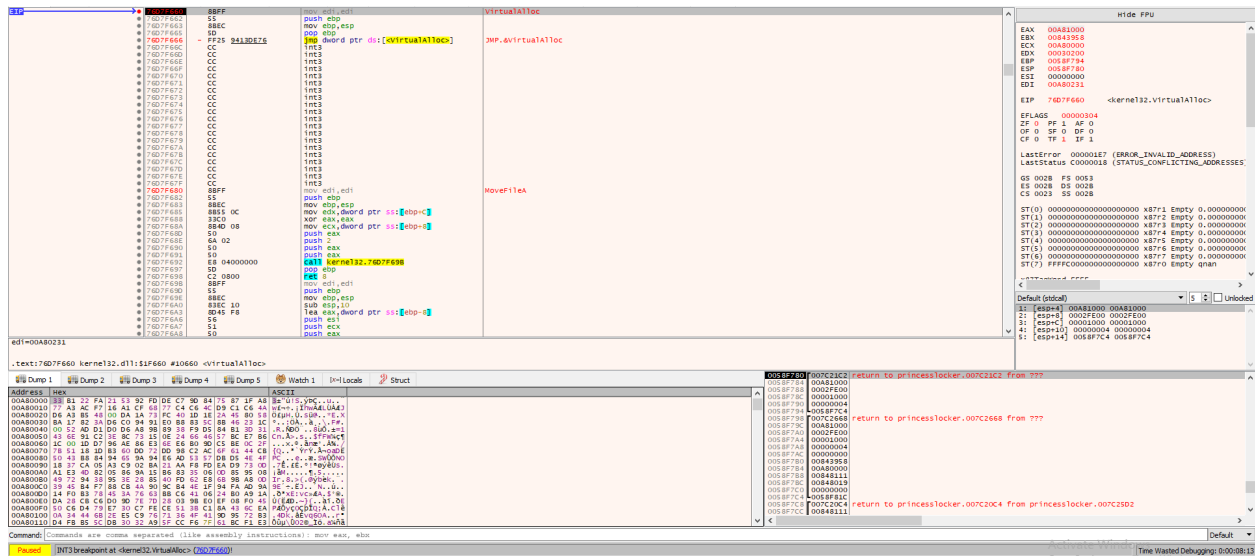


Figure 6. Data Stored in Memory

Address	Hex	ASCII
00A801B0	00 00 C0 04	.A.a.....
00A801C0	00 00 00 00D.L1
00A801D0	00 10 08 048.....
00A801E0	00 00 00 00
00A801F0	00 48 08 04	.H...@.....
00A80200	00 00 10 03
00A80210	00 00 00 00
00A80220	00 2E 74 65	.text...>y...
00A80230	00 00 FE 02p.....
00A80240	00 00 00 00rdata...
00A80250	00 EC 54 01	.it.....V....
00A80260	00 00 00 00@.....
00A80270	40 2E 64 61	@.data...06..p.
00A80280	00 00 1C 00x.....
00A80290	00 00 00 00@.A.gfids.
00A802A0	00 70 0A 00	.p....°.....t.
00A802B0	00 00 00 00@.....
00A802C0	40 2E 72 73	@.rsrc...a...A.

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	@@@details@Concu
00447D70	72 72 65 6E	rency@...e"C.
00447D80	00 00 00 2E?AVimproper
00447D90	5F 6C 6F 63	_lock@Concurrenc
00447DA0	79 40 40 00	y@e"C.....?AV
00447DB0	73 63 68 65	scheduler_resour
00447DC0	63 65 5F 61	ce_allocation_er
00447DD0	72 6F 72 40	ror@Concurrenc
00447DE0	40 00 00 00	@...e"C.....?AV
00447DF0	75 64 5F 63	unsupporte_os@c
00447E00	6F 6E 63 75	ncurrency@e"C.
00447E10	E8 22 43 00	e"C.....?AVinva
00447E20	6C 69 64 5F	lid_operation@Co
00447E30	6E 63 75 72	ncurrency@e"C.
00447E40	00 00 00 2E?AVresource
00447E50	40 61 6E 61	Manager@details@
00447E60	43 6F 6E 63	Concurrenc
00447E70	E8 22 43 00	e"C.....?AVires

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.

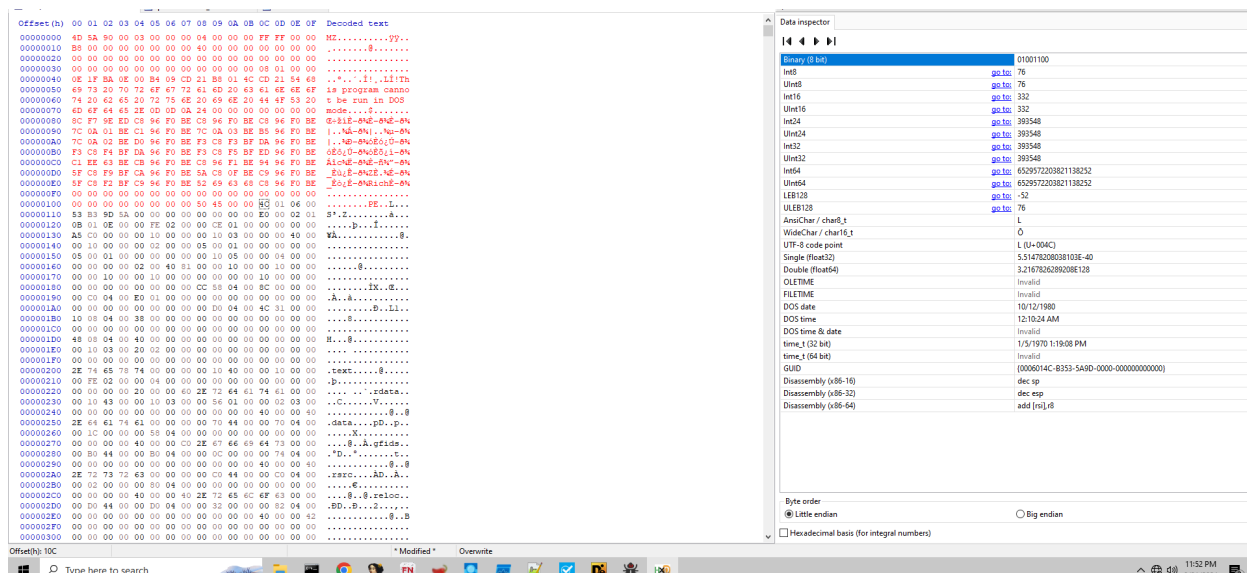


Figure 9. Replacing corrupted MZ Header.

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

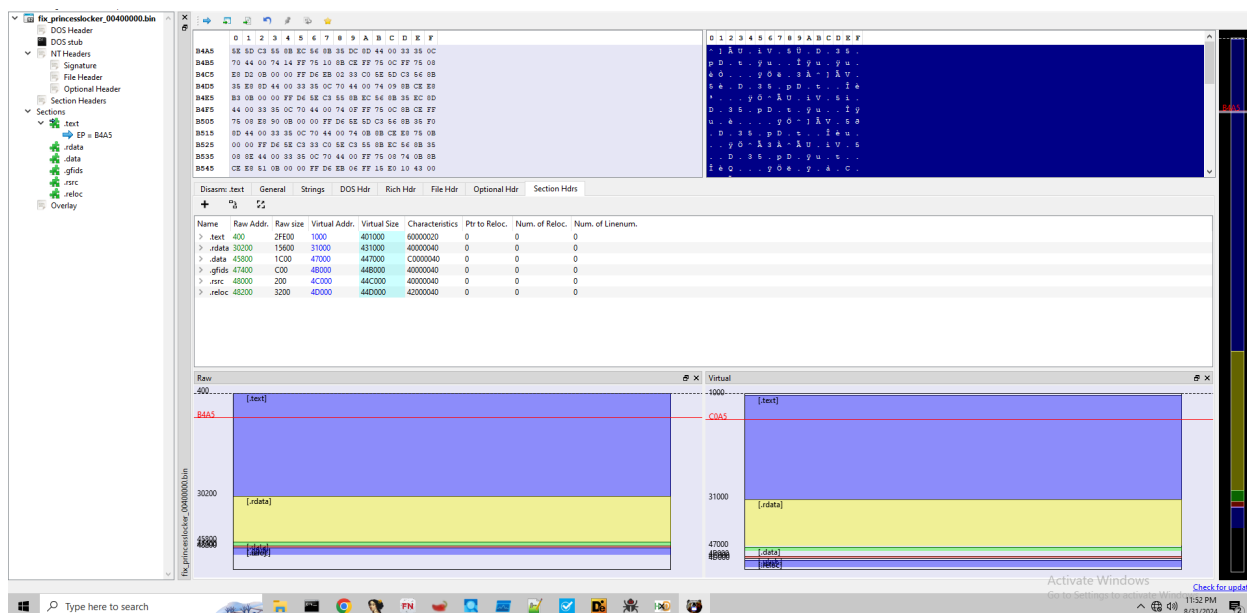


Figure 10. Section Headers on PE-Bear

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.

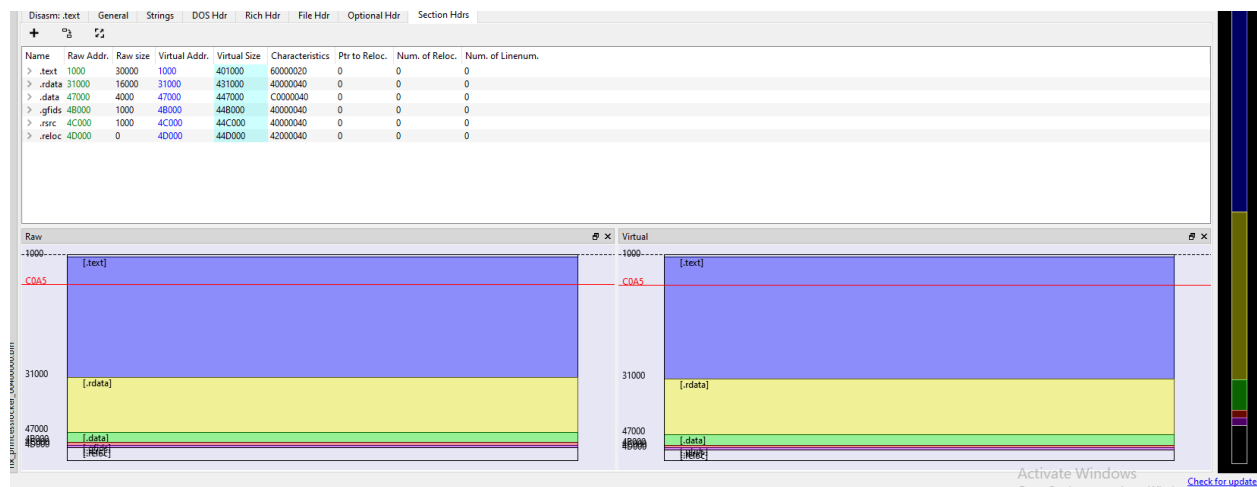


Figure 11. Fixed Section Headers

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.

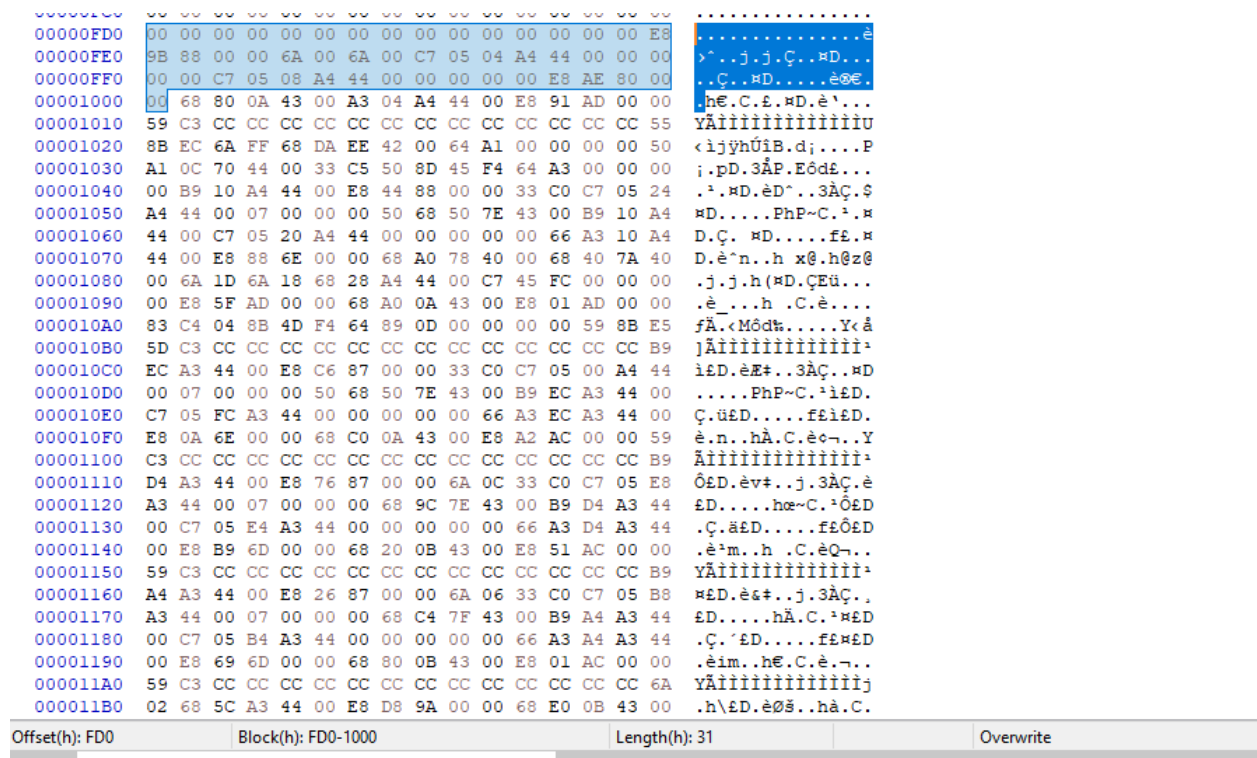


Figure 12. Realigning Section Headers

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.

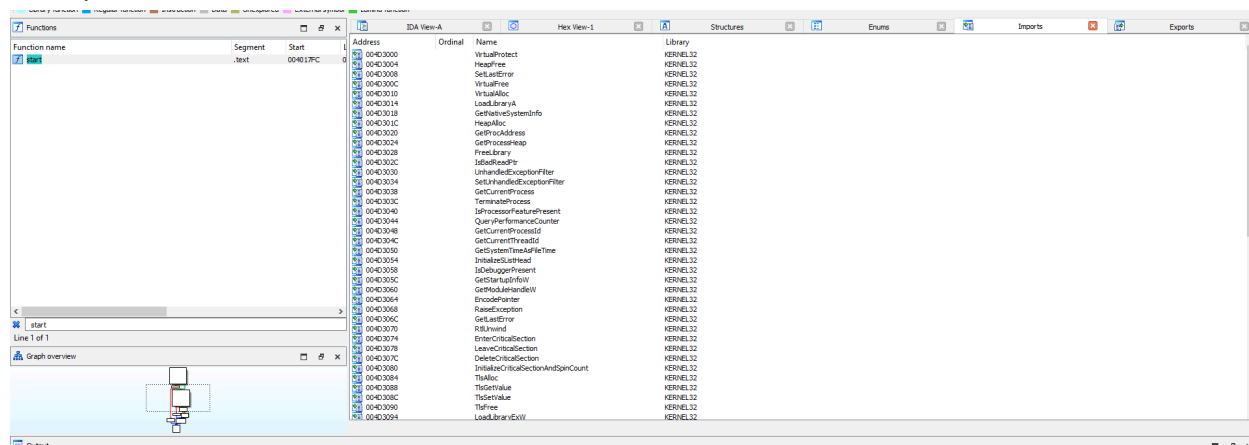


Figure 13. Packed Exports

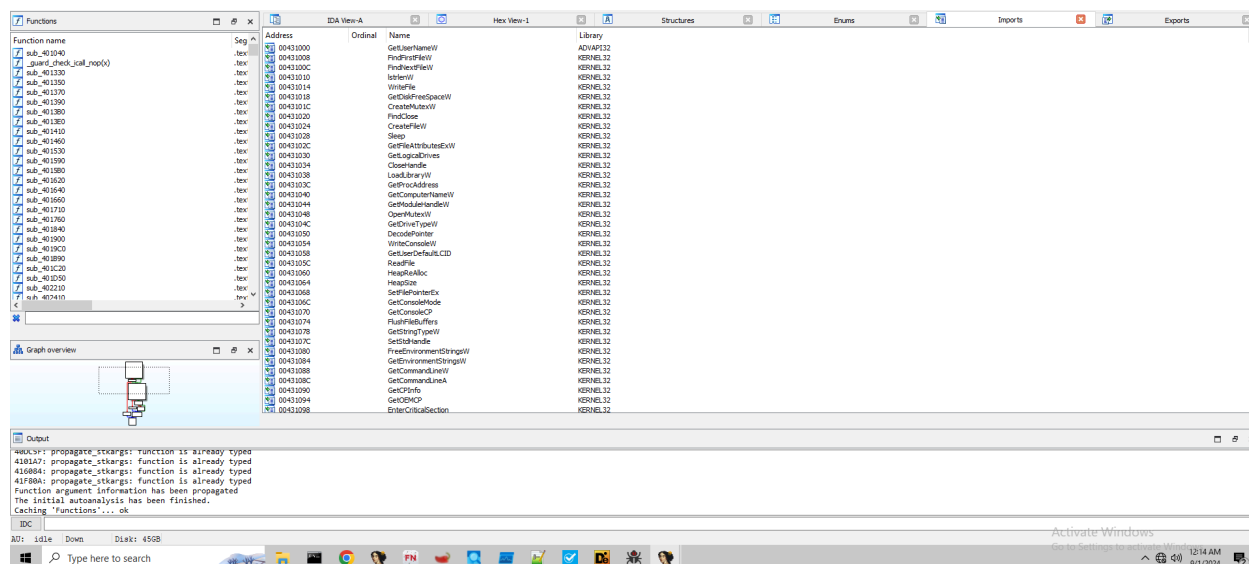


Figure 14. Unpacked Exports

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.