SHA256: 939c575e17fcf1afbe2889a4ddb44f095ff3a07cdf9f5dd3d5c7f49e93da68c0

Putting the exe on Detect-it-Easy shows that it is a 32bit file compiled in C/C++. Looking into the strings and entropy will also suggest that this file is packed or encrypted.

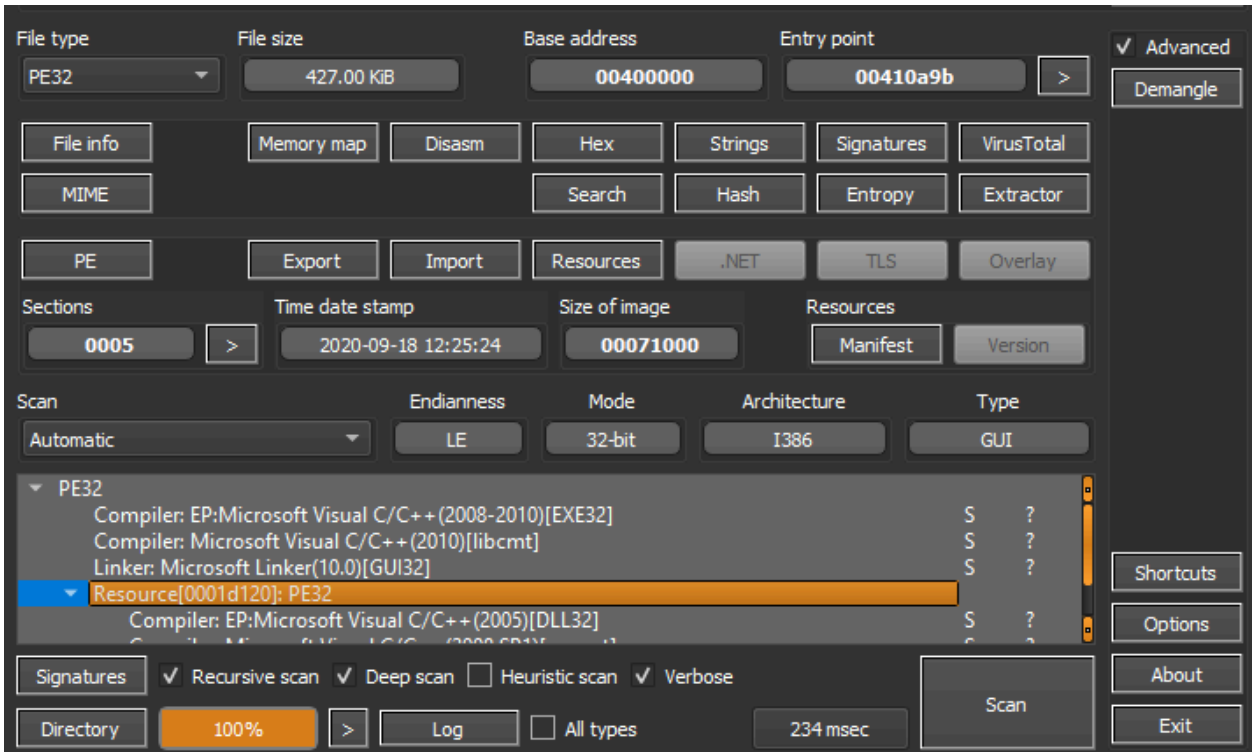
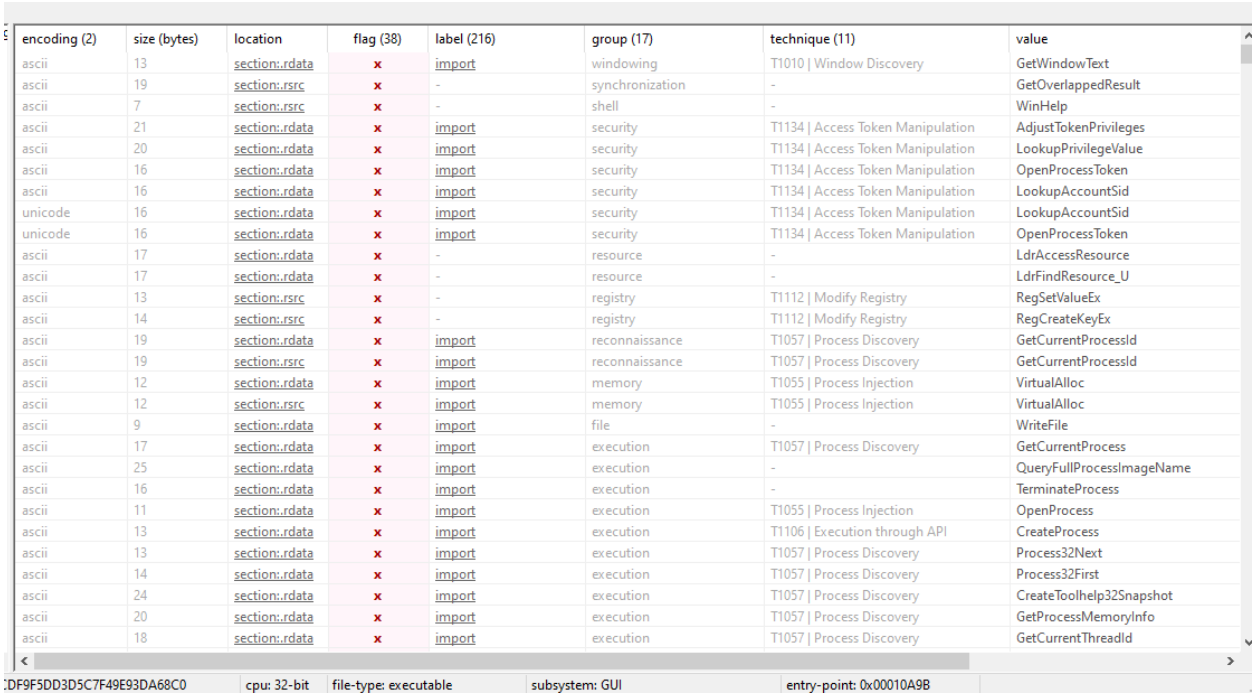


Figure 1. Detect it Easy

The next step I performed is to open the file in PE-Studio and look at the details of the files there. The image below shows the flagged strings, such as, VirtualAlloc and a bunch of others that are commonly found on malware.

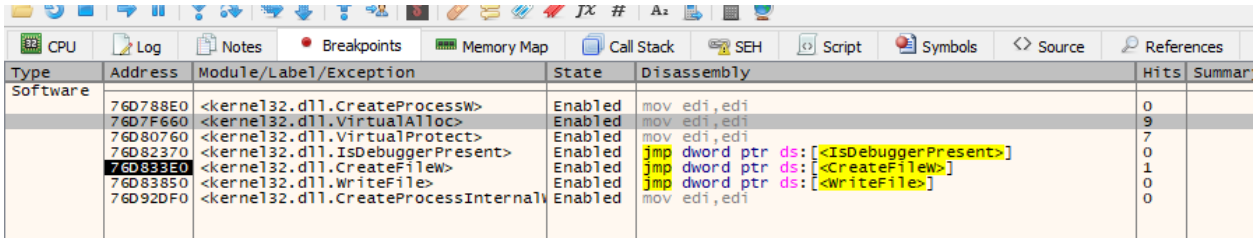


encoding (2)	size (bytes)	location	flag (38)	label (216)	group (17)	technique (11)	value
ascii	13	section:.rdata	x	import	windowing	T1010 | Window Discovery	GetWindowText
ascii	19	section:.rsrc	x	-	synchronization	-	GetOverlappedResult
ascii	7	section:.rsrc	x	-	shell	-	WinHelp
ascii	21	section:.rdata	x	import	security	T1134 | Access Token Manipulation	AdjustTokenPrivileges
ascii	20	section:.rdata	x	import	security	T1134 | Access Token Manipulation	LookupPrivilegeValue
ascii	16	section:.rdata	x	import	security	T1134 | Access Token Manipulation	OpenProcessToken
ascii	16	section:.rdata	x	import	security	T1134 | Access Token Manipulation	LookupAccountSid
unicode	16	section:.rdata	x	import	security	T1134 | Access Token Manipulation	LookupAccountSid
unicode	16	section:.rdata	x	import	security	T1134 | Access Token Manipulation	OpenProcessToken
ascii	17	section:.rdata	x	-	resource	-	LdrAccessResource
ascii	17	section:.rdata	x	-	resource	-	LdrFindResource_U
ascii	13	section:.rsrc	x	-	registry	T1112 | Modify Registry	RegSetValueEx
ascii	14	section:.rsrc	x	-	registry	T1112 | Modify Registry	RegCreateKeyEx
ascii	19	section:.rdata	x	import	reconnaissance	T1057 | Process Discovery	GetCurrentProcessId
ascii	19	section:.rsrc	x	import	reconnaissance	T1057 | Process Discovery	GetCurrentProcessId
ascii	12	section:.rdata	x	import	memory	T1055 | Process Injection	VirtualAlloc
ascii	12	section:.rsrc	x	import	memory	T1055 | Process Injection	VirtualAlloc
ascii	9	section:.rdata	x	import	file	-	WriteFile
ascii	17	section:.rdata	x	import	execution	T1057 | Process Discovery	GetCurrentProcess
ascii	25	section:.rdata	x	import	execution	-	QueryFullProcessImageName
ascii	16	section:.rdata	x	import	execution	-	TerminateProcess
ascii	11	section:.rdata	x	import	execution	T1055 | Process Injection	OpenProcess
ascii	13	section:.rdata	x	import	execution	T1106 | Execution through API	CreateProcess
ascii	13	section:.rdata	x	import	execution	T1057 | Process Discovery	Process32Next
ascii	14	section:.rdata	x	import	execution	T1057 | Process Discovery	Process32First
ascii	24	section:.rdata	x	import	execution	T1057 | Process Discovery	CreateToolhelp32Snapshot
ascii	20	section:.rdata	x	import	execution	T1057 | Process Discovery	GetProcessMemoryInfo
ascii	18	section:.rdata	x	import	execution	T1057 | Process Discovery	GetCurrentThreadId

:DF9F5DD3D5C7F49E93DA68C0 | cpu: 32-bit | file-type: executable | subsystem: GUI | entry-point: 0x00010A9B

Figure 2. Flagged Strings

I set the usual breakpoints on x64dbg such as VirtualAlloc/Protect along with CreateFileW and WriteFile. I also later on set more breakpoints based on the behavior of the file such as DeleteFile and TerminateProcess.



CPU | Log | Notes | Breakpoints | Memory Map | Call Stack | SEH | Script | Symbols | Source | References

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Summar
Software						
	76D788E0	<kernel32.dll.CreateProcessW>	Enabled	mov edi,edi	0	
	76D7F660	<kernel32.dll.VirtualAlloc>	Enabled	mov edi,edi	9	
	76D80760	<kernel32.dll.VirtualProtect>	Enabled	mov edi,edi	7	
	76D82370	<kernel32.dll.IsDebuggerPresent>	Enabled	jmp dword ptr ds:[<IsDebuggerPresent>]	0	
	76D833E0	<kernel32.dll.CreateFileW>	Enabled	jmp dword ptr ds:[<CreateFileW>]	1	
	76D83850	<kernel32.dll.WriteFile>	Enabled	jmp dword ptr ds:[<WriteFile>]	0	
	76D92DF0	<kernel32.dll.CreateProcessInternalW	Enabled	mov edi,edi	0	

Figure 3. Breakpoints

After hitting the following breakpoints and following through the VirtualAlloc calls until VirtualProtect attempts to change the memory protection of the allocated memory and until the MZ header is found.

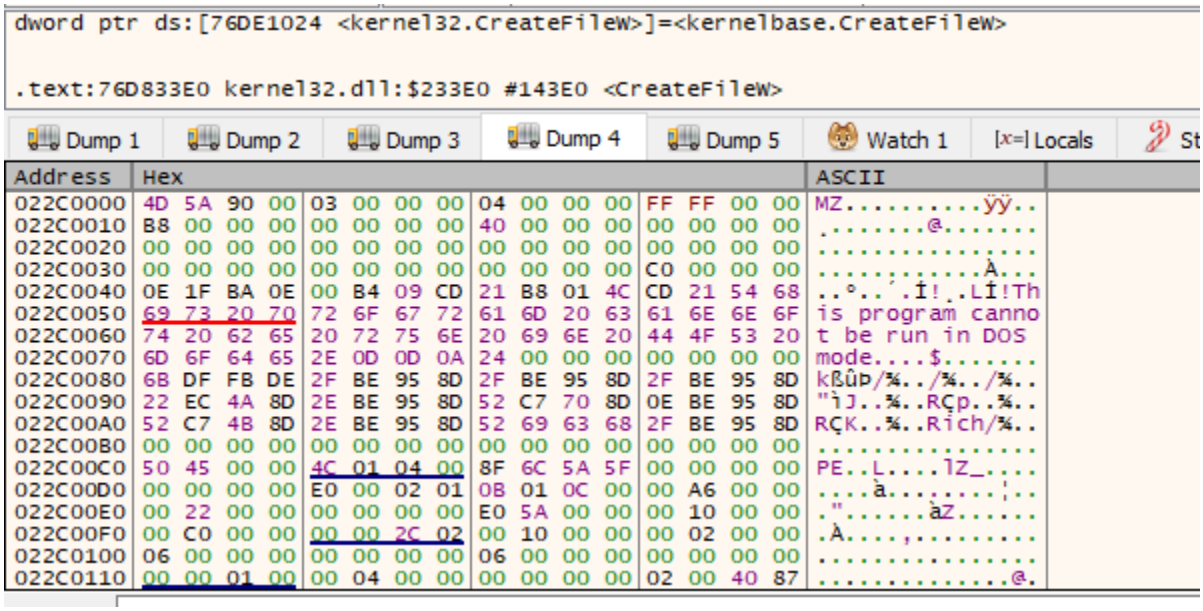


Figure 4. MZ Header on Virtual Alloc

The binary is then dumped and is unmapped and repaired with PE-Bear and will be used for analysis later on.

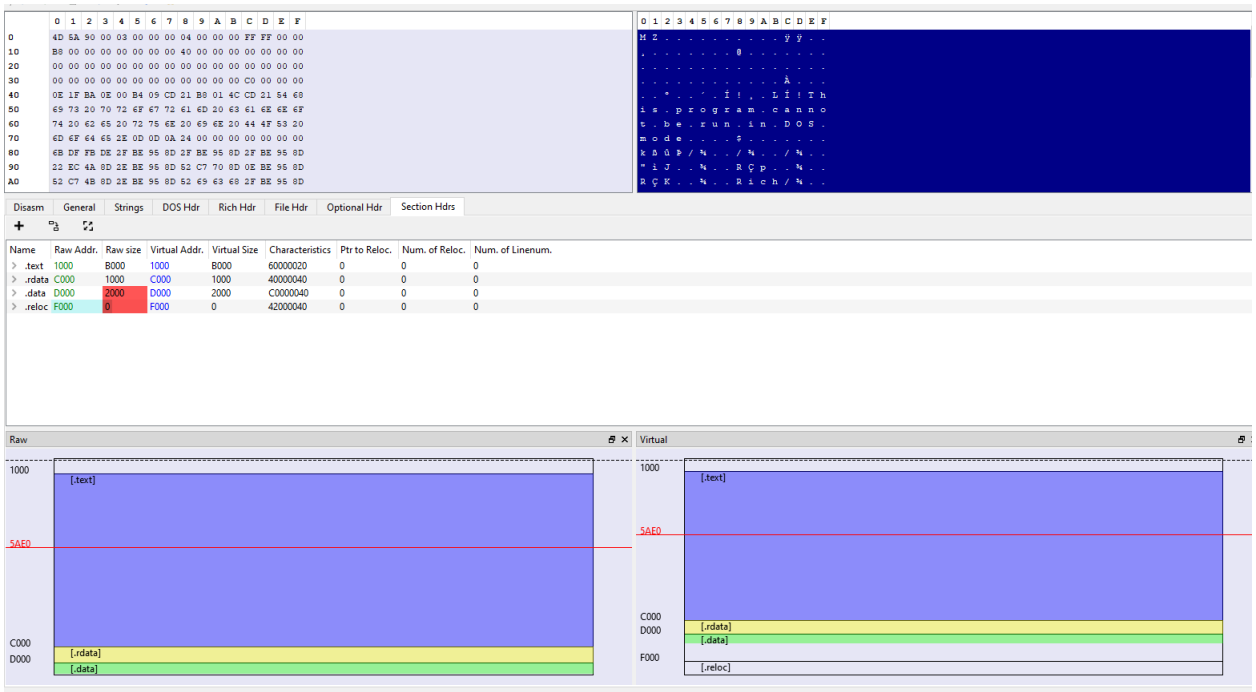


Figure 5. Unmapping and Repairing IAT

After dumping the previous binary, I then continue running the executable and see that it is going to create a file in the SysWow64 folder. These file names are random but it is always going to be on the SysWow folder then on a random generated subfolder but it is going to be duplicate of itself after deleting the original running process.

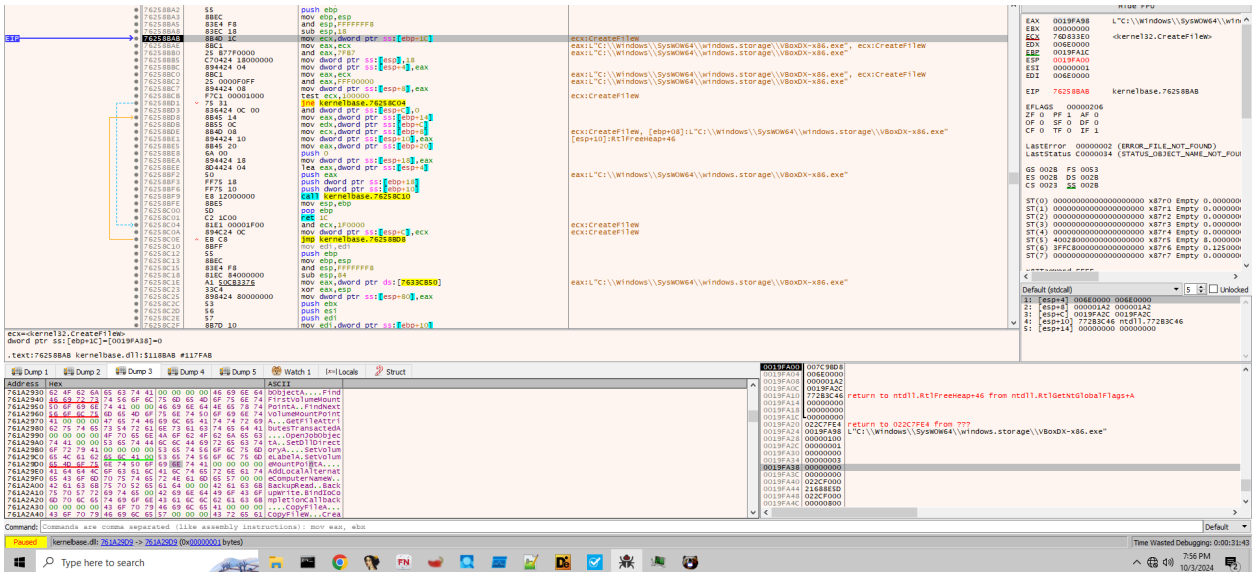


Figure 6. File Creation

Hash of created file matches the original file used for analysis.

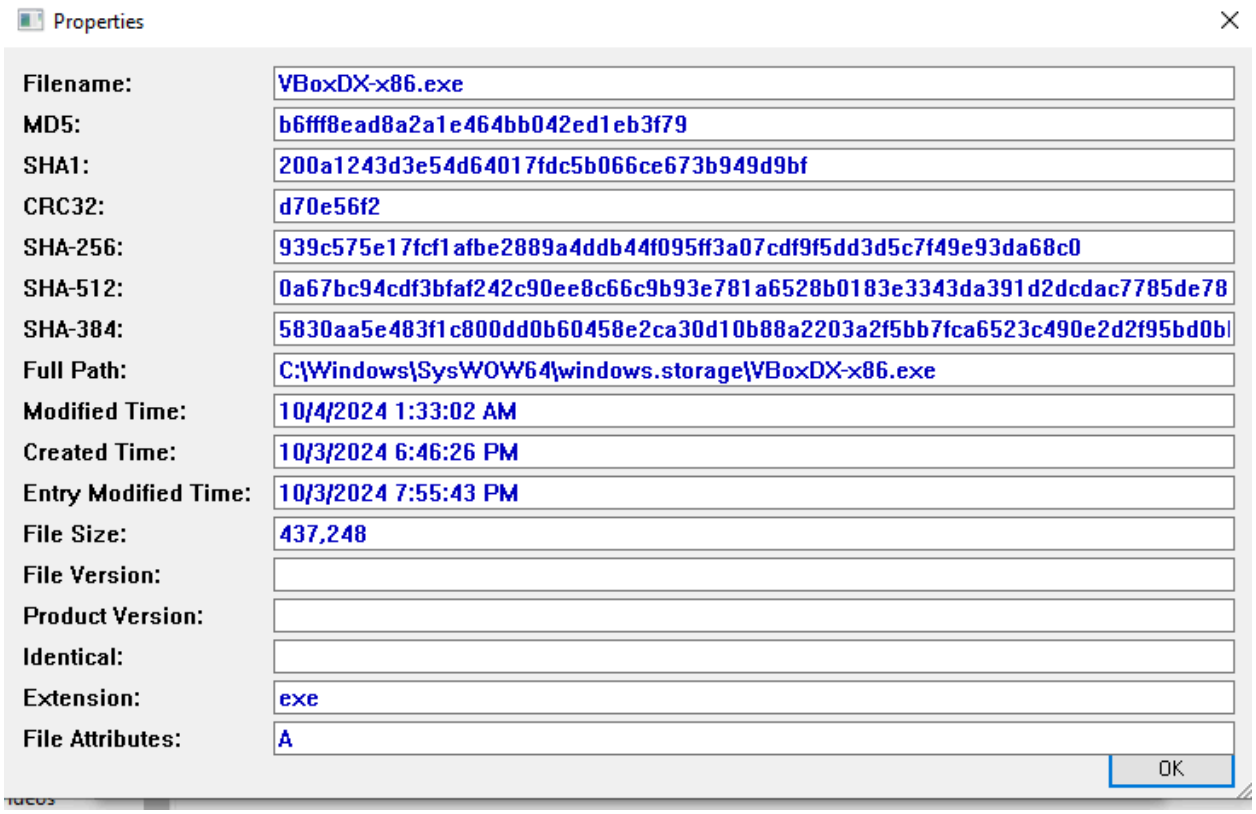


Figure 7. Hash

The image below shows a different subfolder and different filename created with the same hash after re-running the executable showing that the naming of these files and folders are randomized.

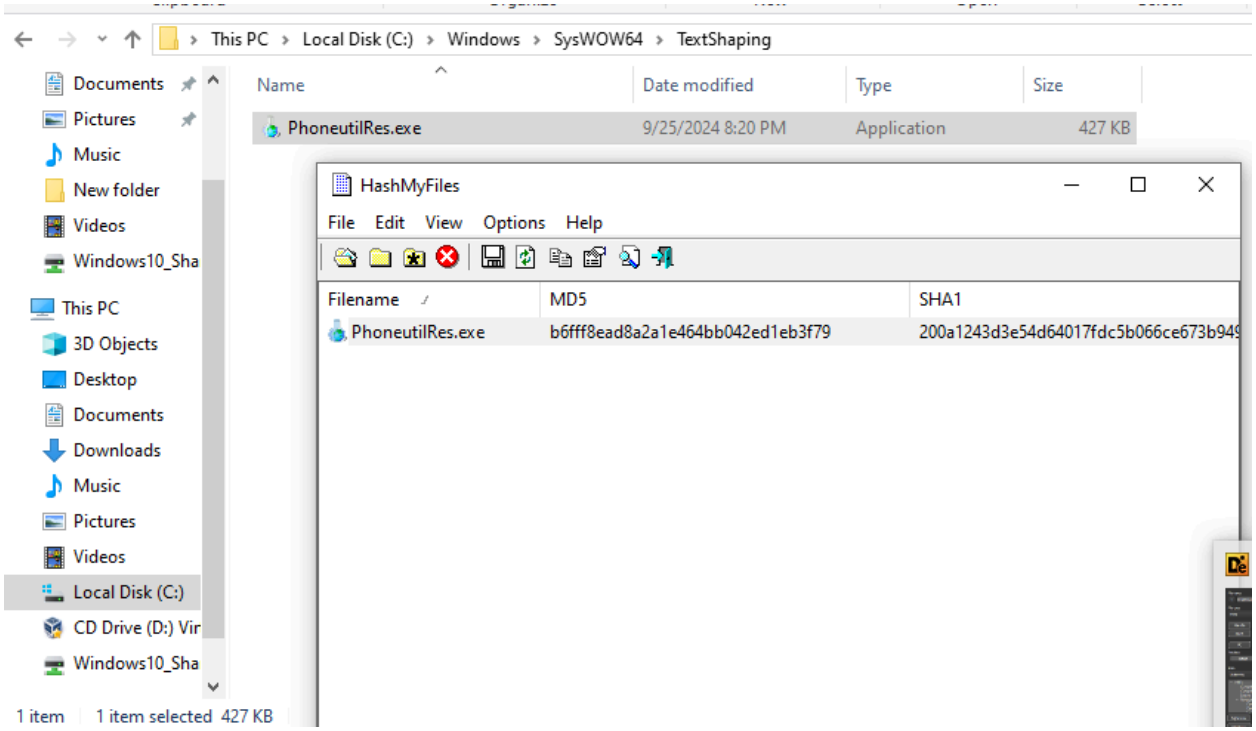


Figure 8. Different Subfolder and File Name

The image below shows that the executable has created a new process that has autostart enabled for persistence.

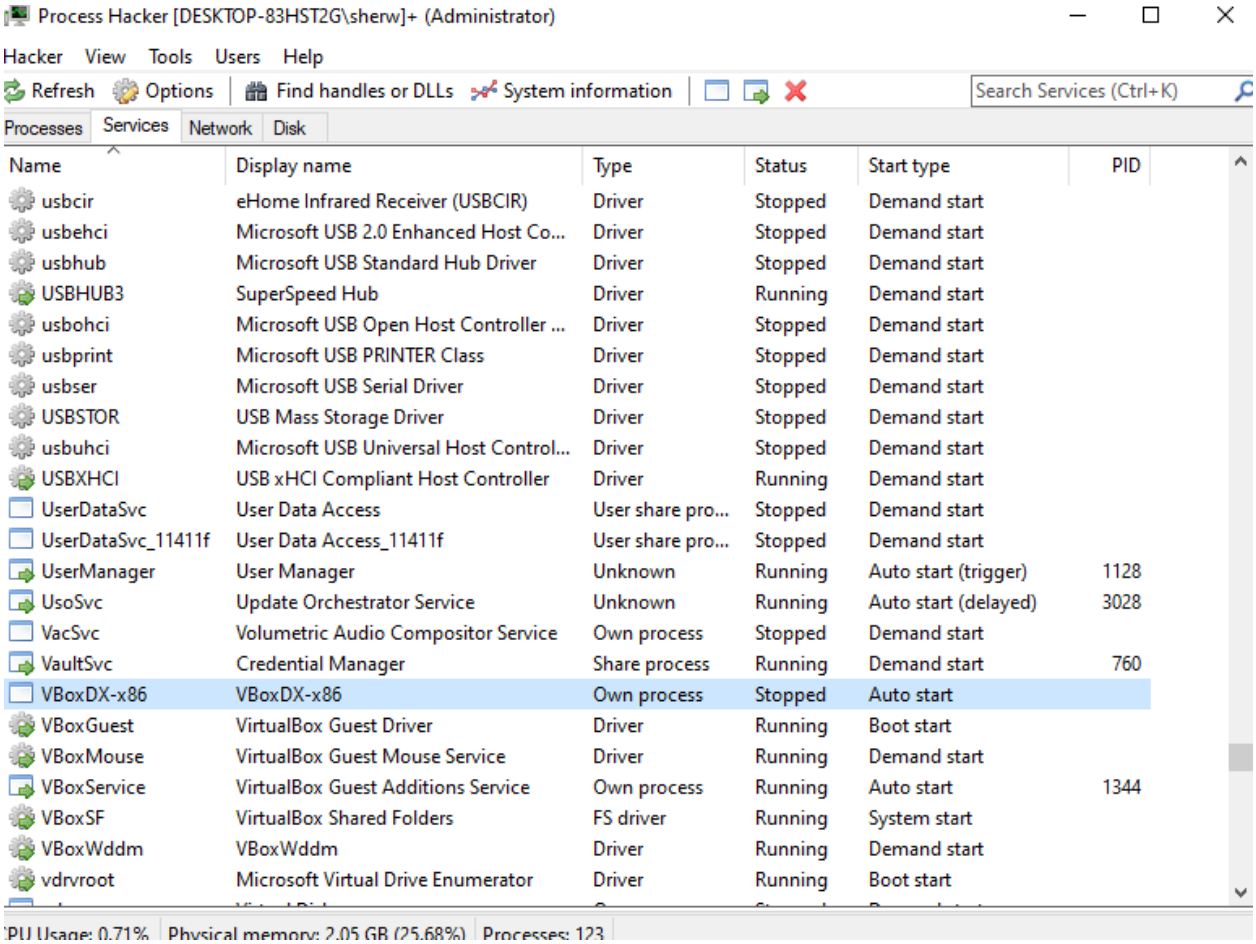


Figure 9. New Service Created

Continuing the execution of the program shows that CreateProcessW and CreateProcessInternalW are being called. This launches the newly created file stealthily as there was no input ever that these files were created if not seen through an analysis machine.

Figure 10. Launching Newly Made Files

The Image below shows the value of the registry key set by the executable along with its description.

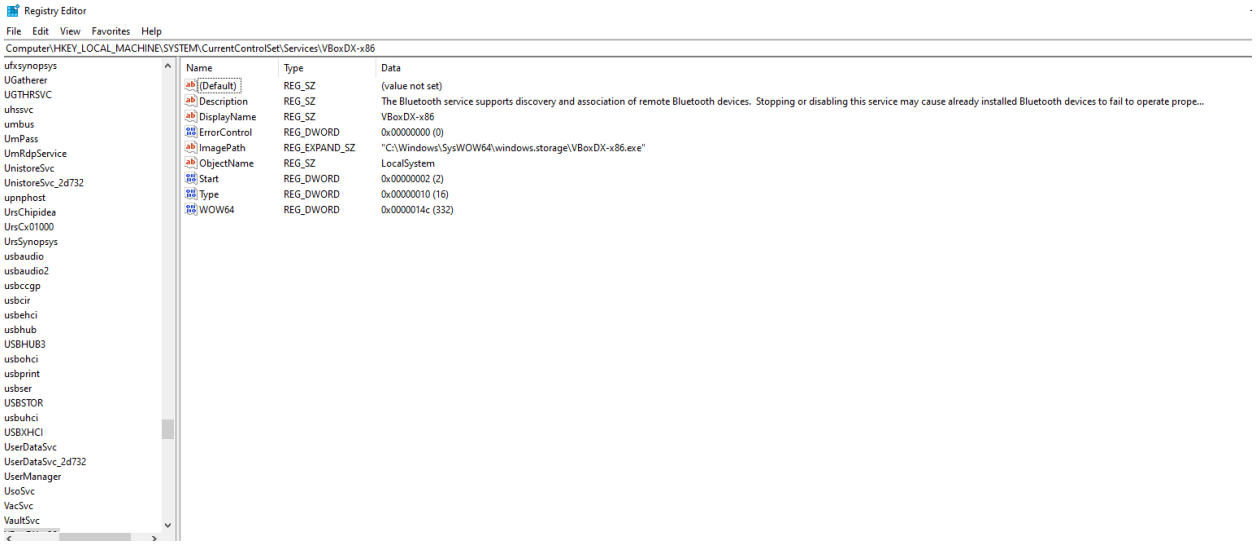


Figure 11. Registry Key