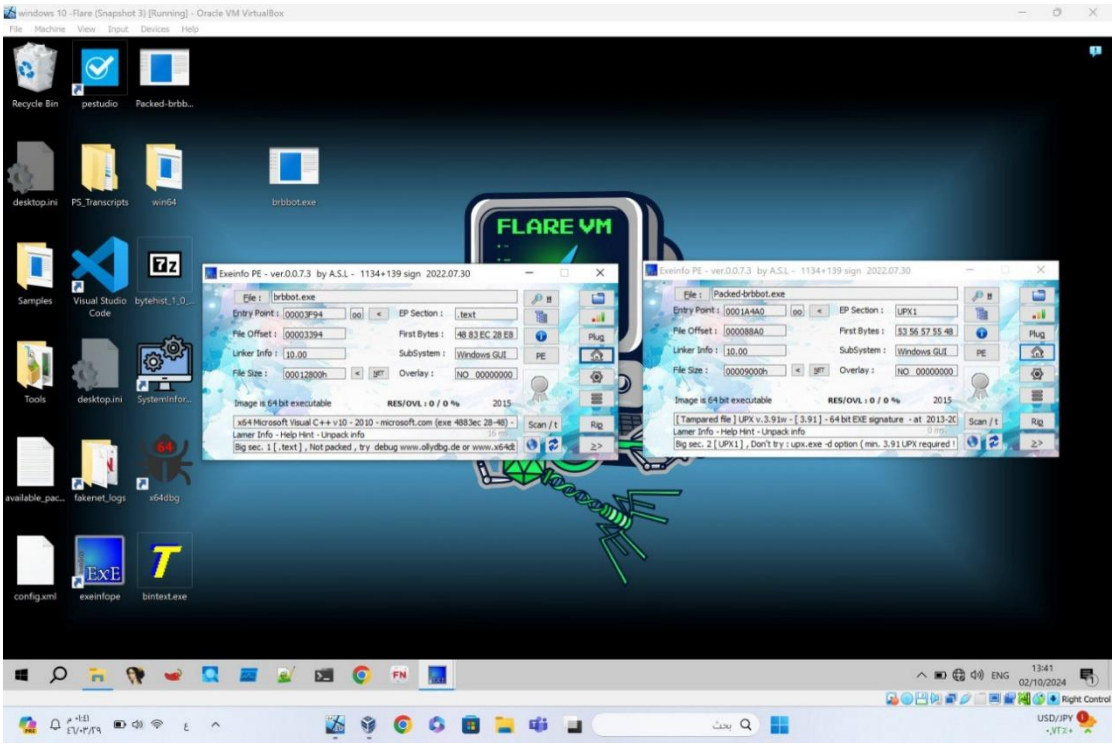
Reverse Engineering and Malware Analysis

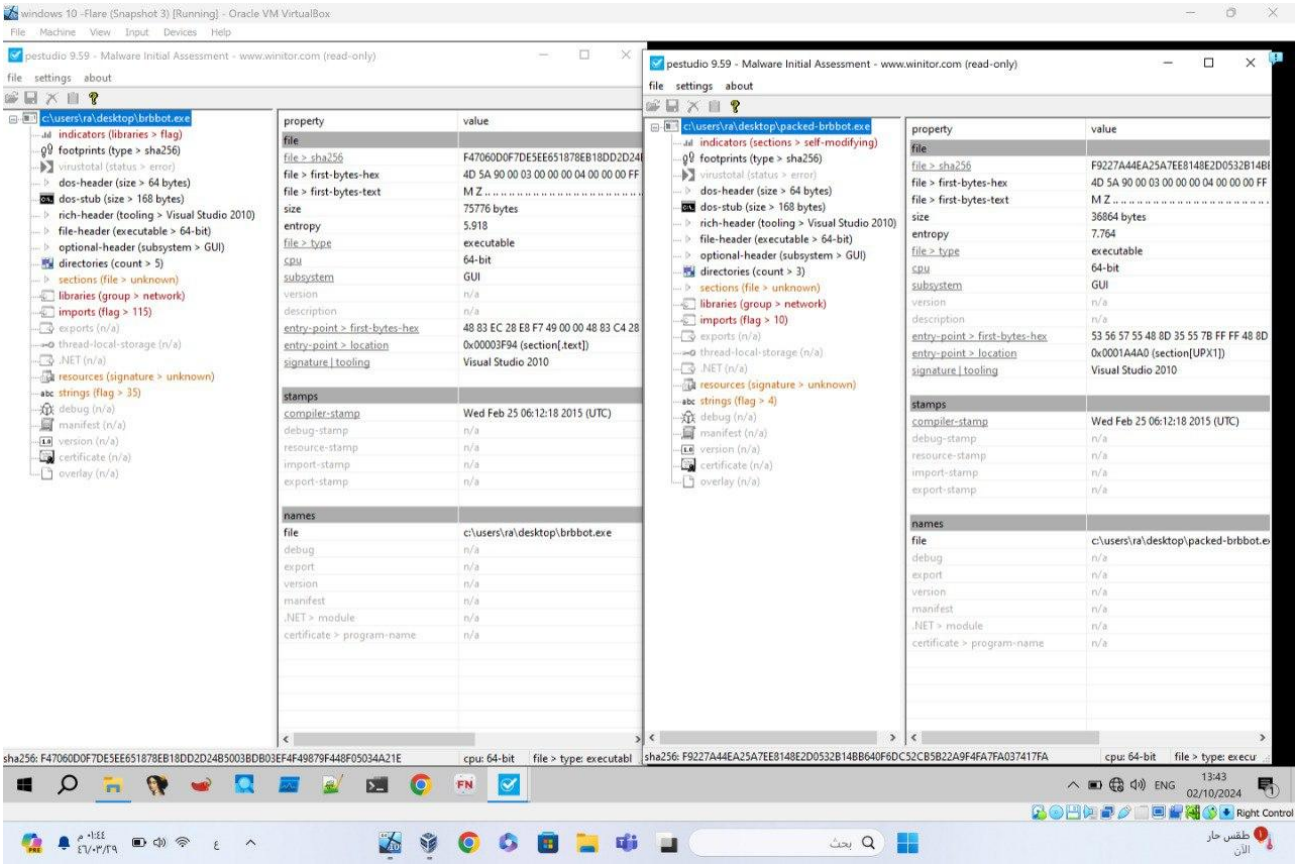
Lab 5 – Unpacking Malware Using Debugger

Raghad lafe - 2111941

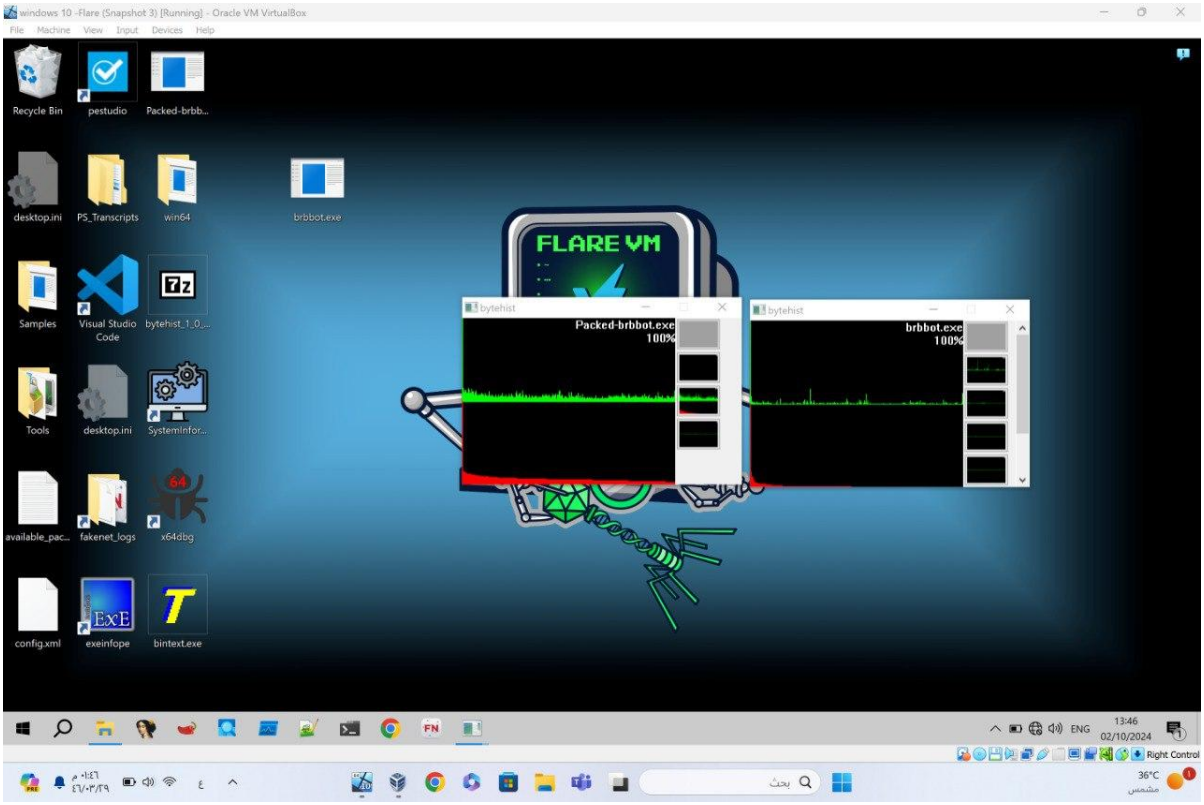
Task 1: Recognizing Packed Malware:

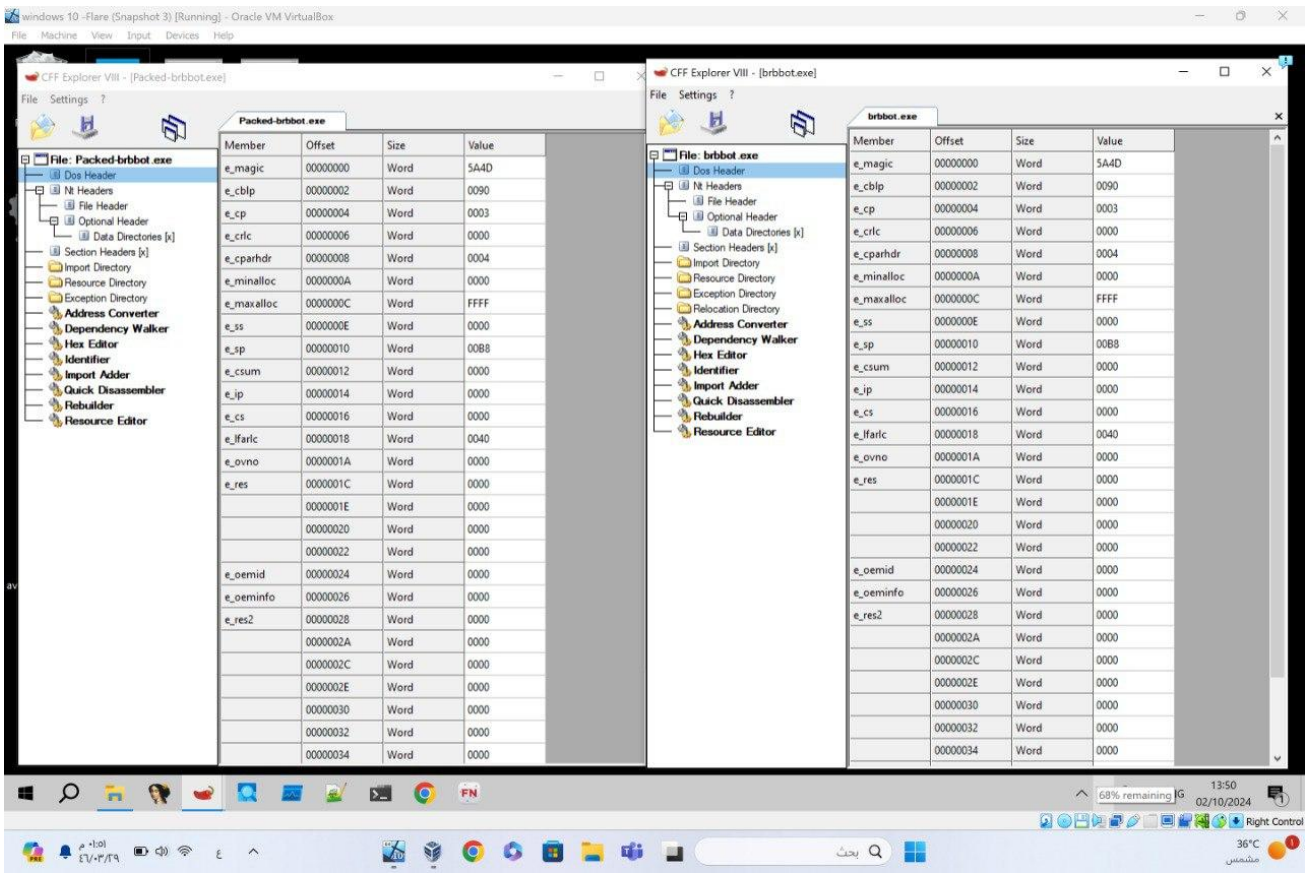
1- A screenshot of both samples (brbbot.exe and packed_brbbot.exe)



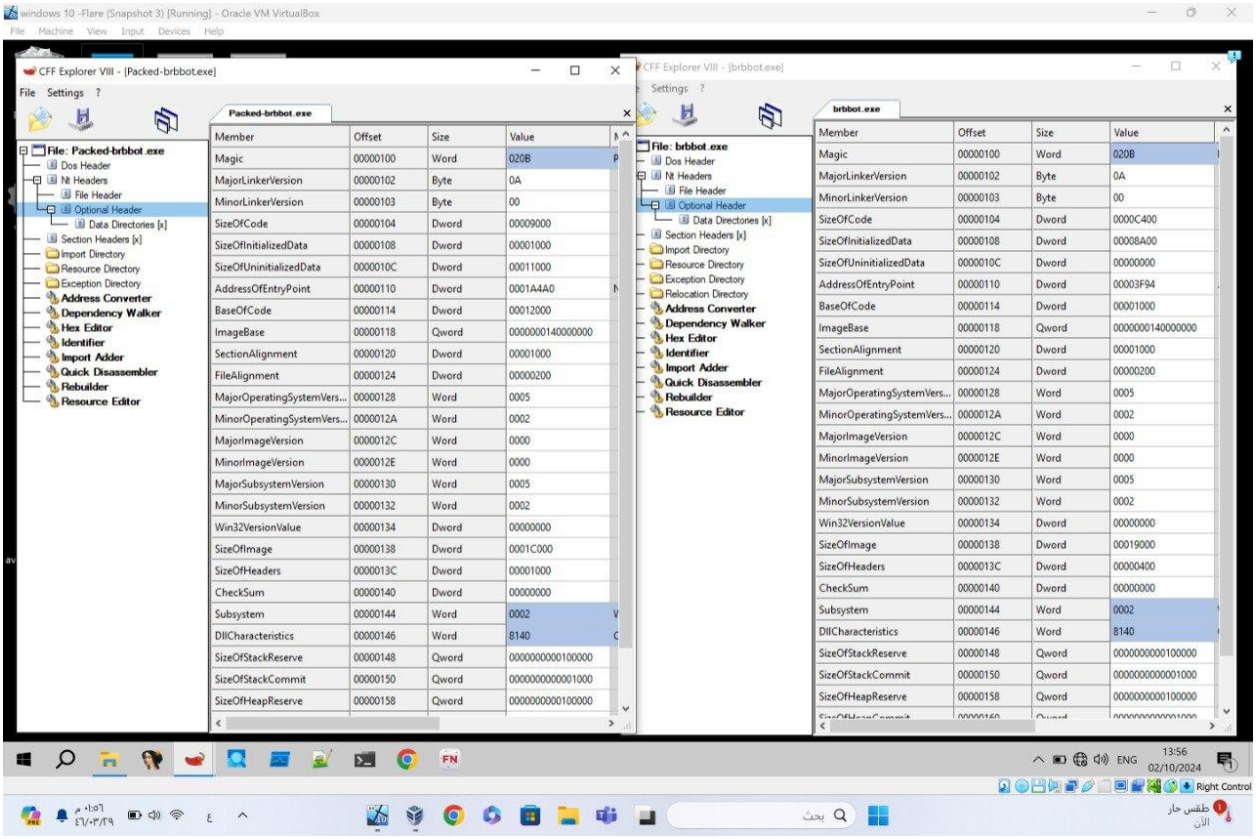


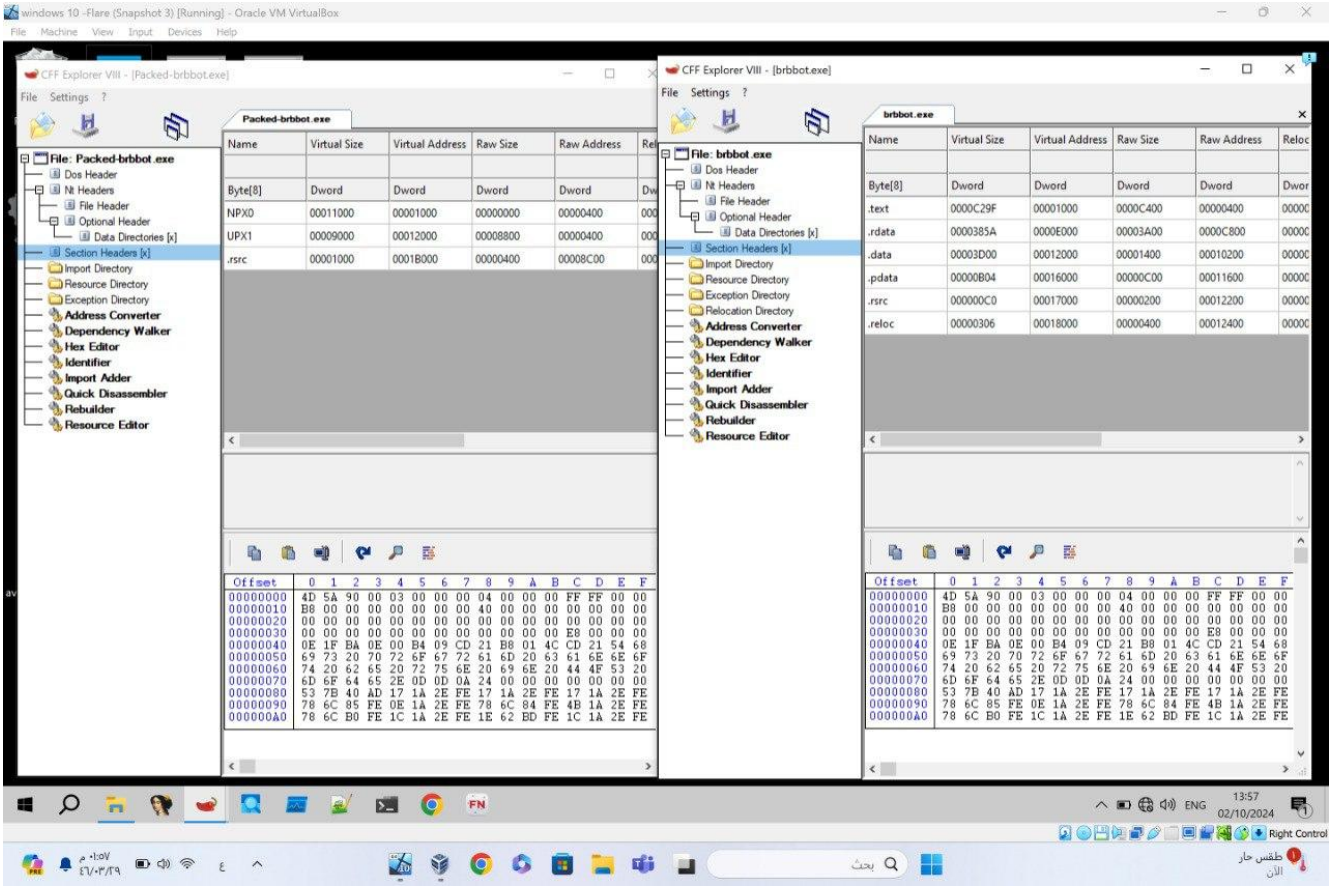
2- A screenshot of the bytehist tool and we could say that the packed file diagram has higher noise and is not stable compared to unpacked one.

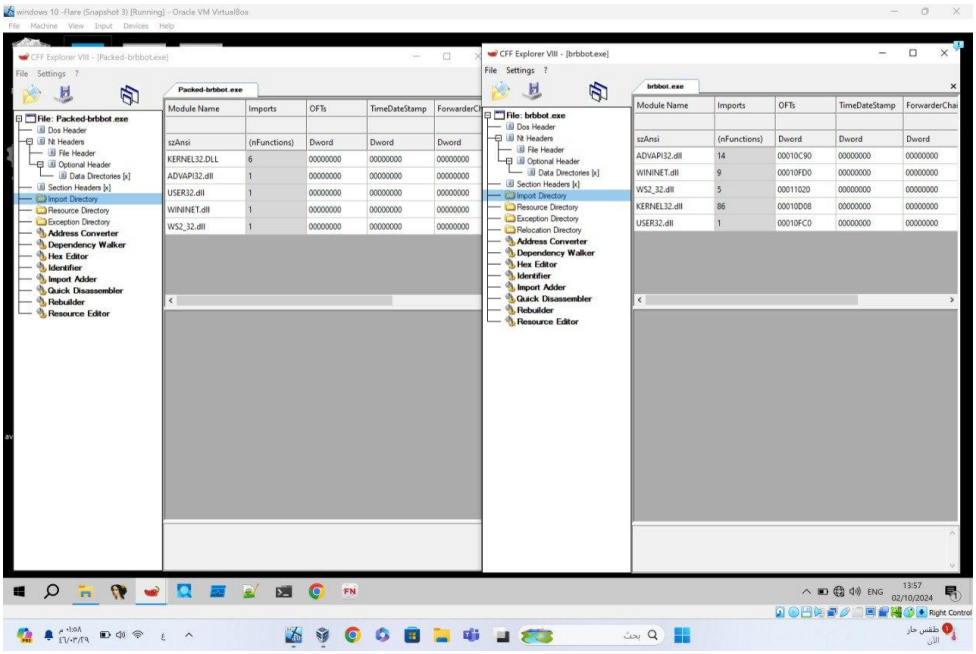




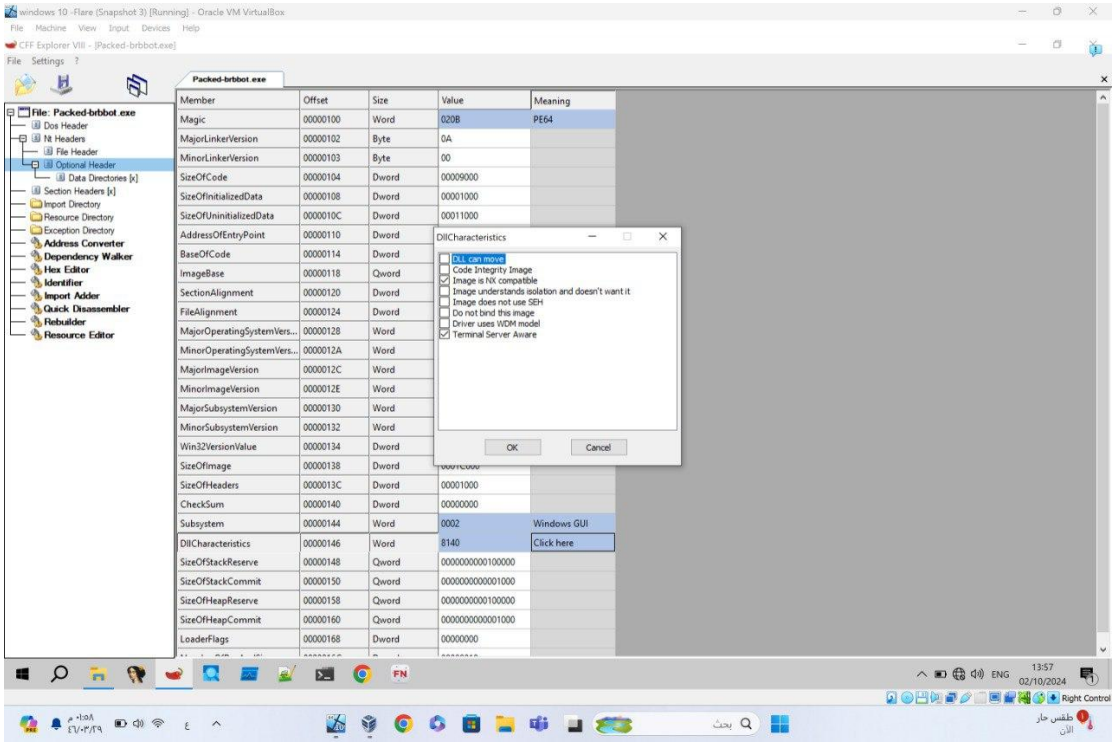
3- From CFF explorer we could see the image Base, sections, import directory.

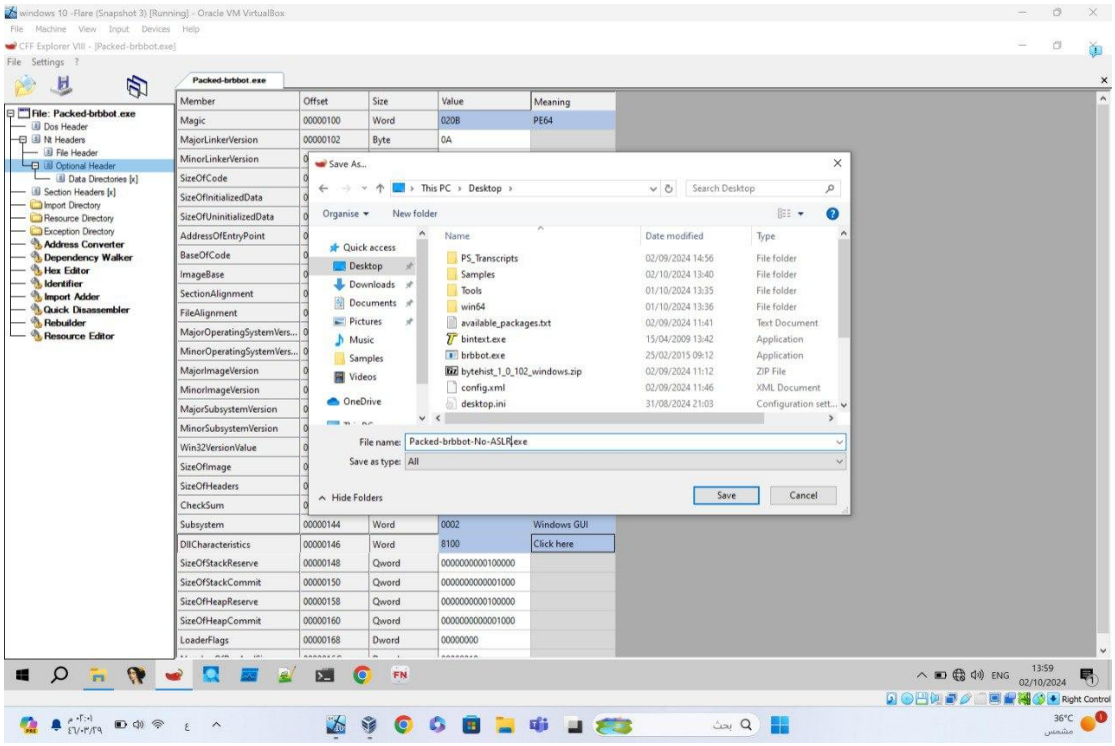






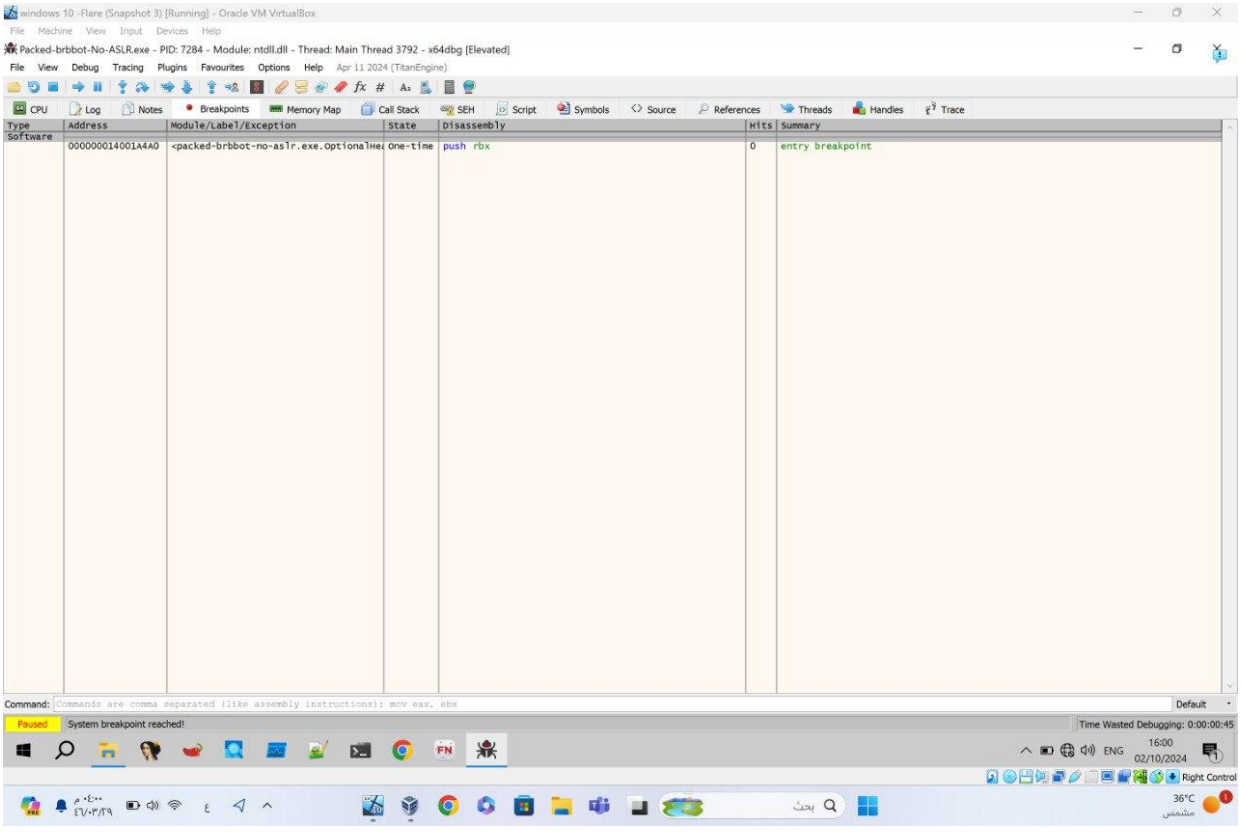
4- Now from CFF going to Disable the ASLR and save a version that has no ASLR so could debug it and unpack it.

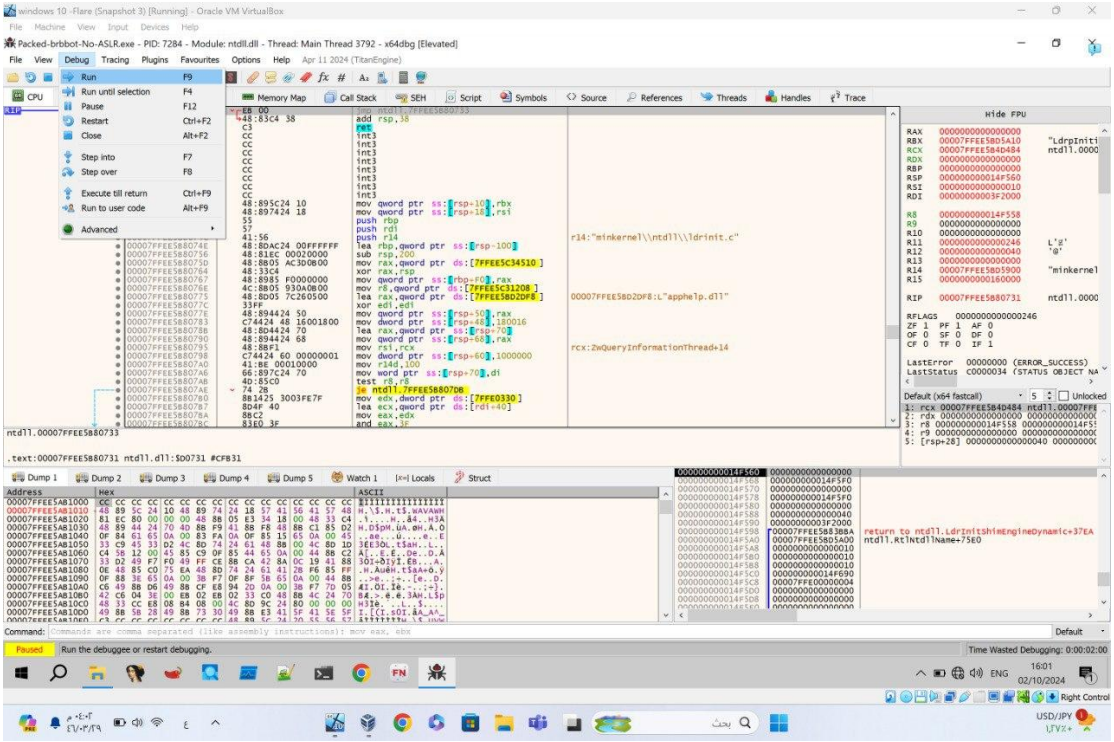




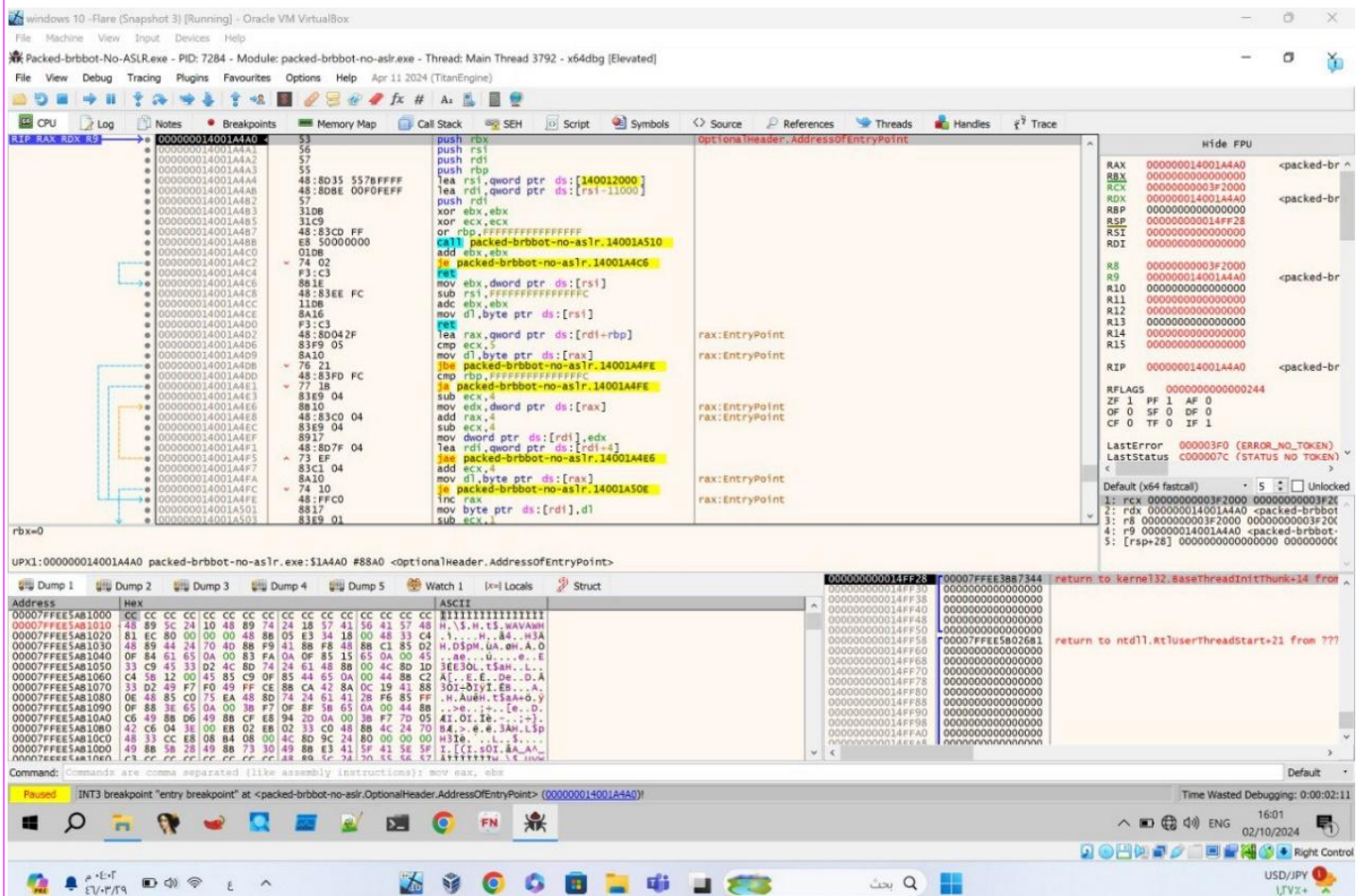
Task 2: Recognizing the Code Patterns of the Packed Malware and Utilize Debugging to Dump the Unpacked Version of the Malware:

1- I am going to use x64 debugger because of the file architecture.
2- And run the file to the entry point.

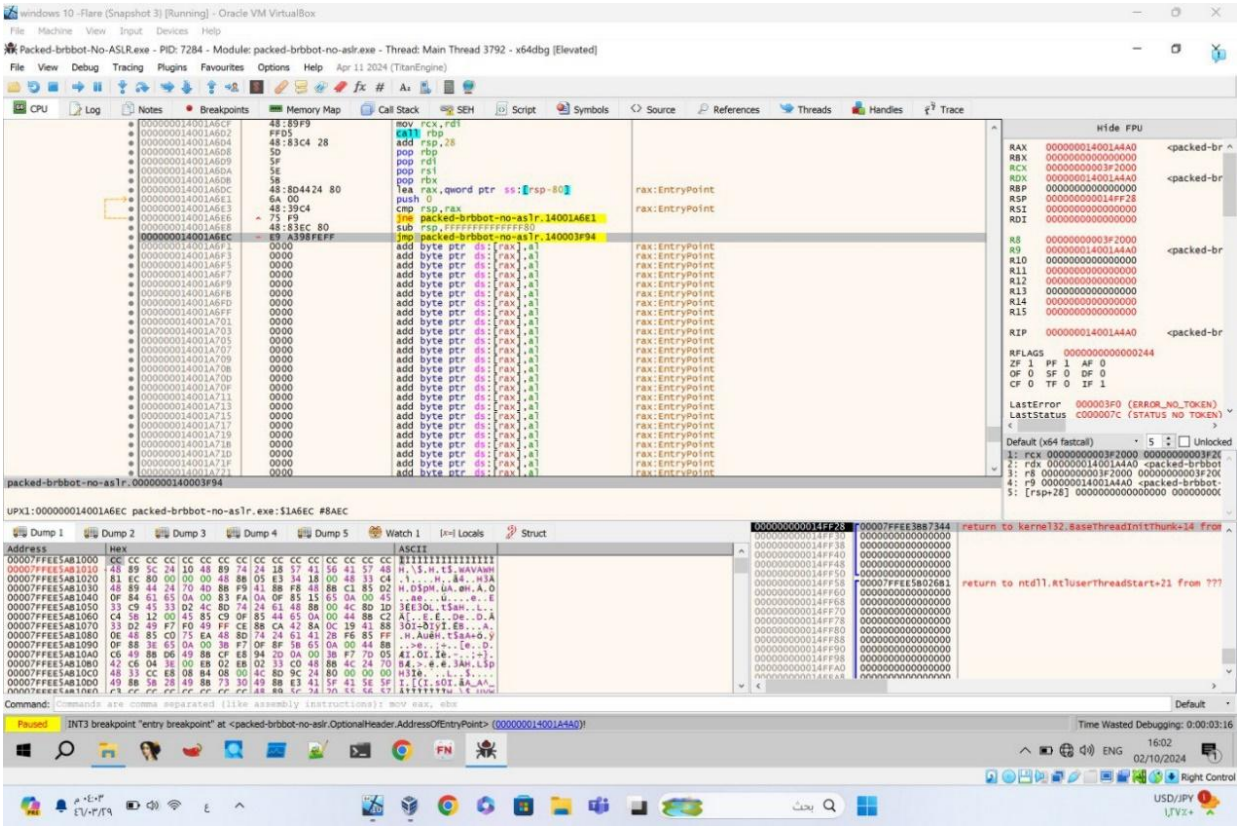


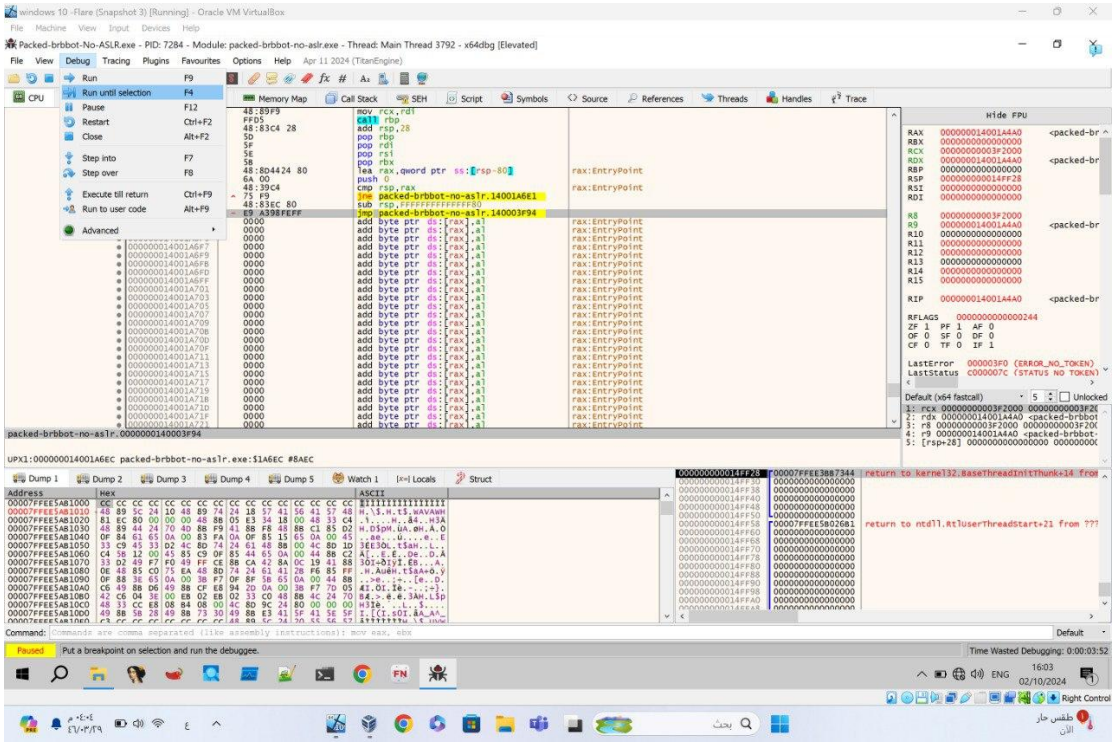


3- Here I could notice the first thing after running the malware is push registers which indicate that the malware is packed.

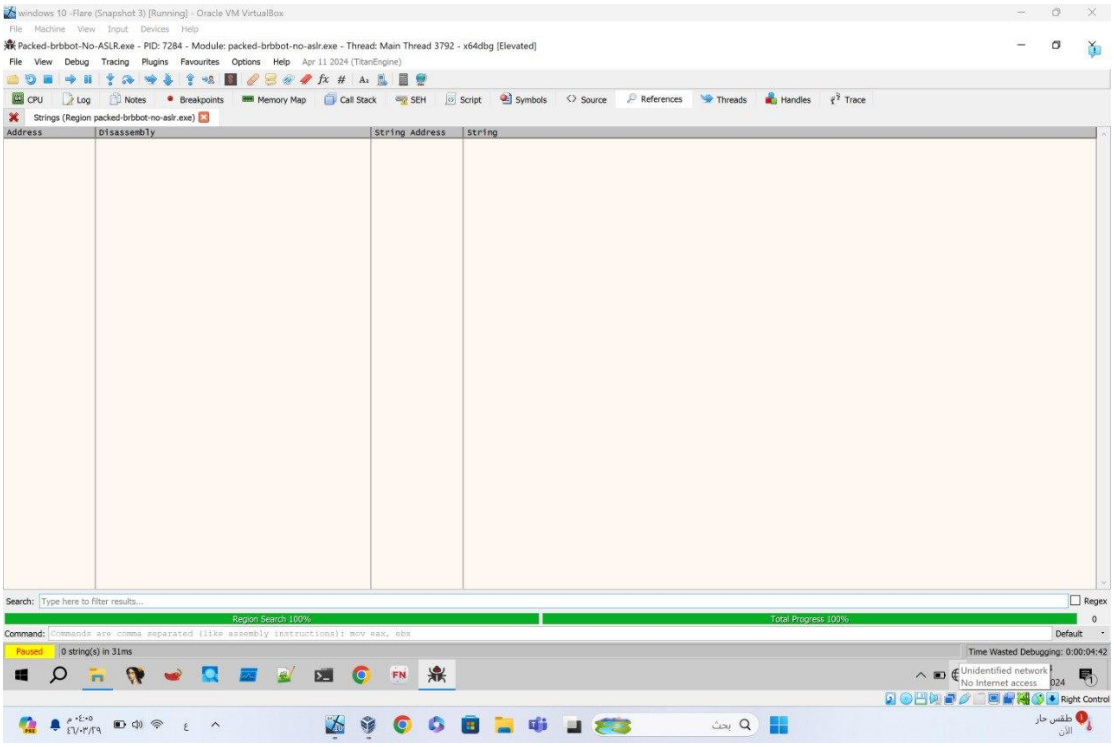


4- I scroll down till I saw zeros address that indicates to me the end of unpacking code of the malware.
5- So, I could conclude that the jmp instruction would take me to the unpacked part of the malware.

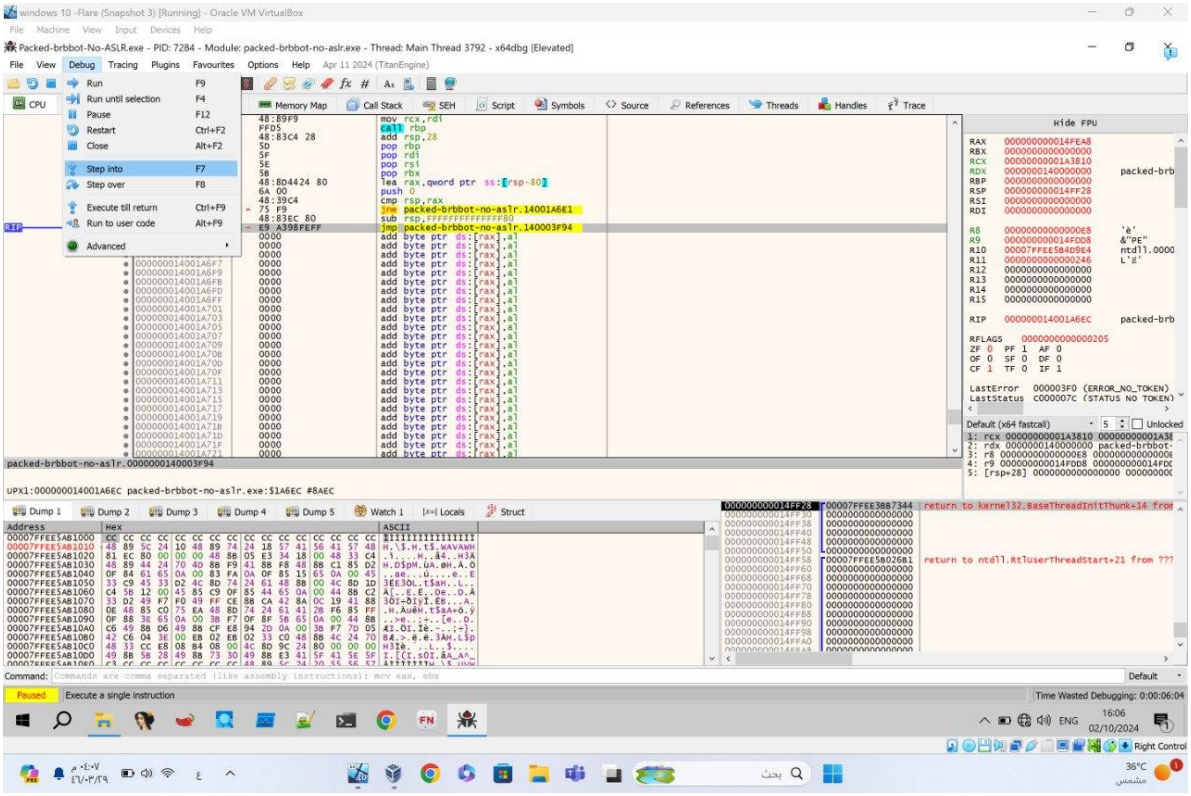


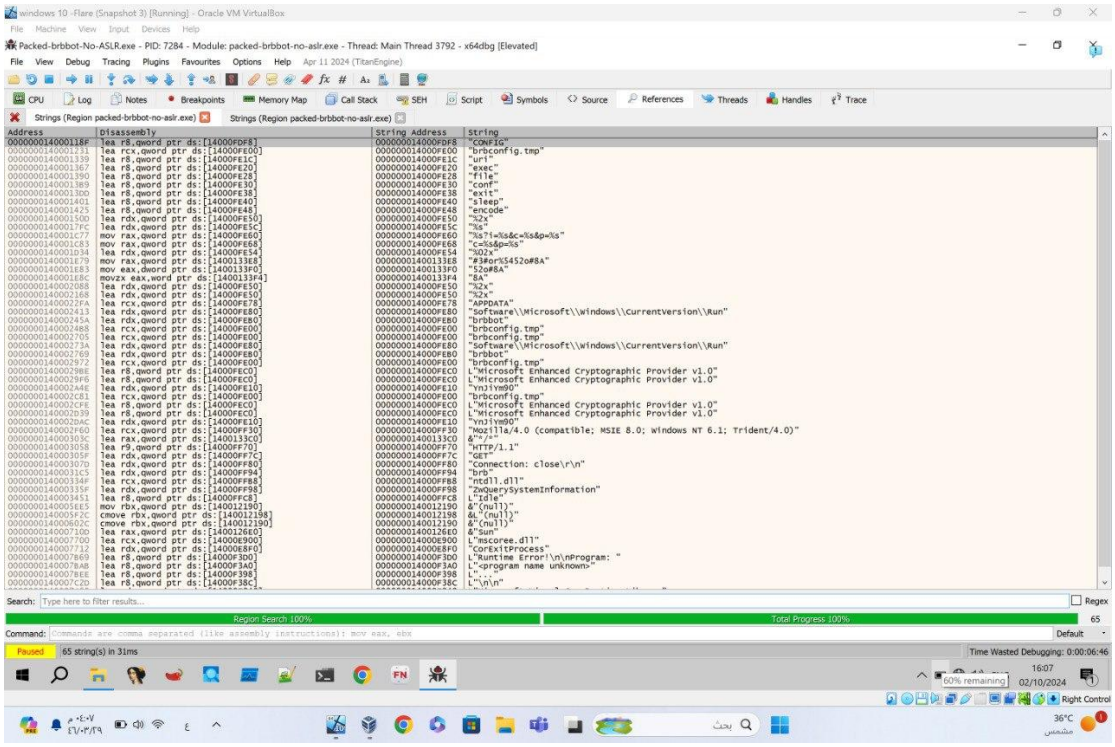


6- I ran to the jmp instruction and searched if there were strings from the unpacked part, but it did not show anything which indicated that I still did not reach to the unpacked part of the malware.

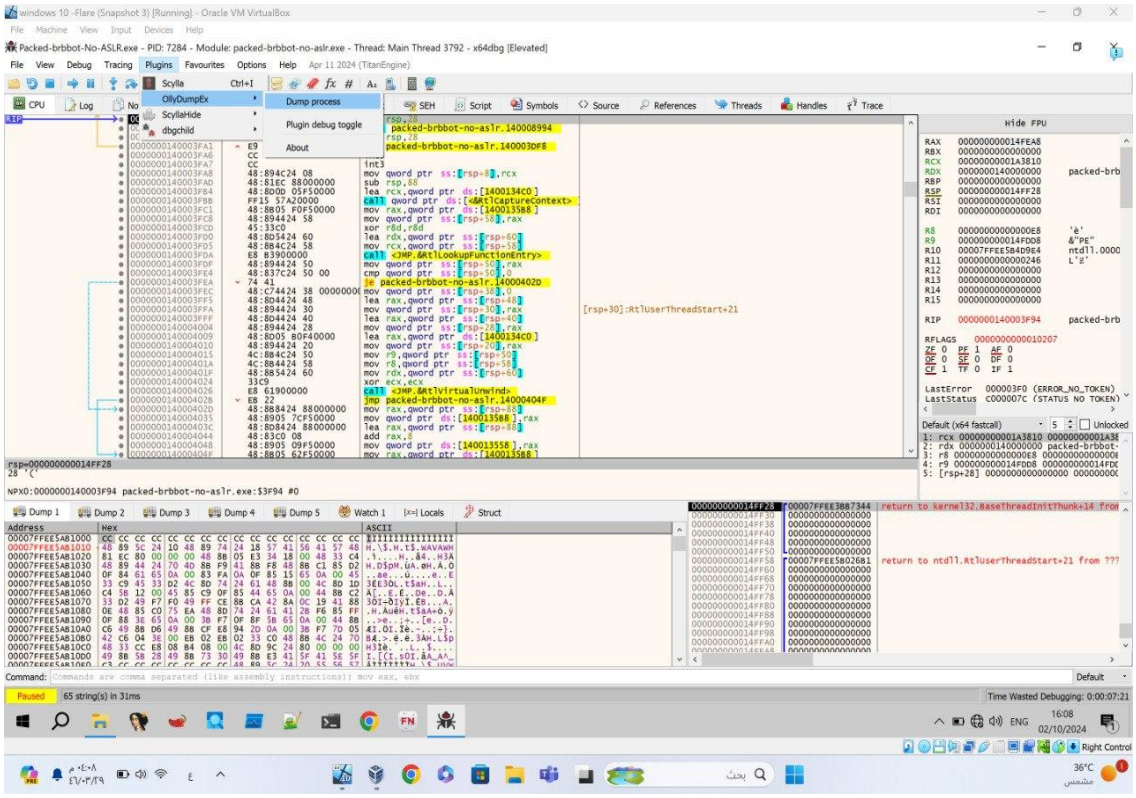


7- So, I will go to the next instruction, jmp instruction go to and search again to see if there are strings shown the indicates me that I reach the unpacked part.

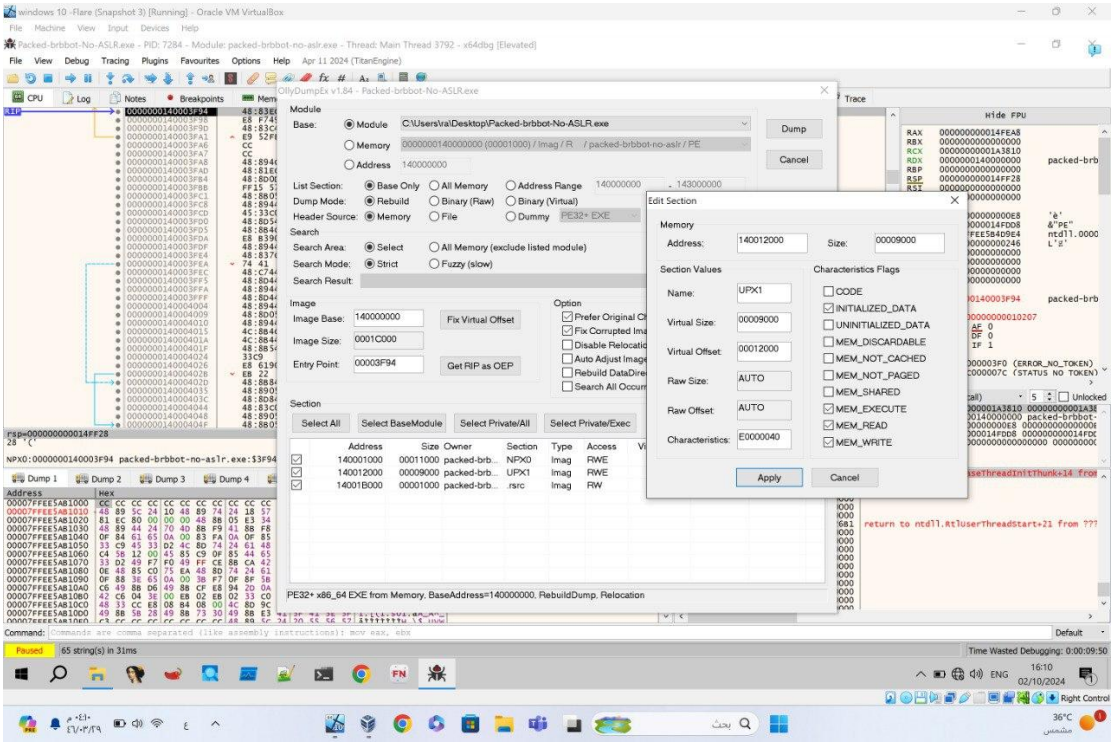


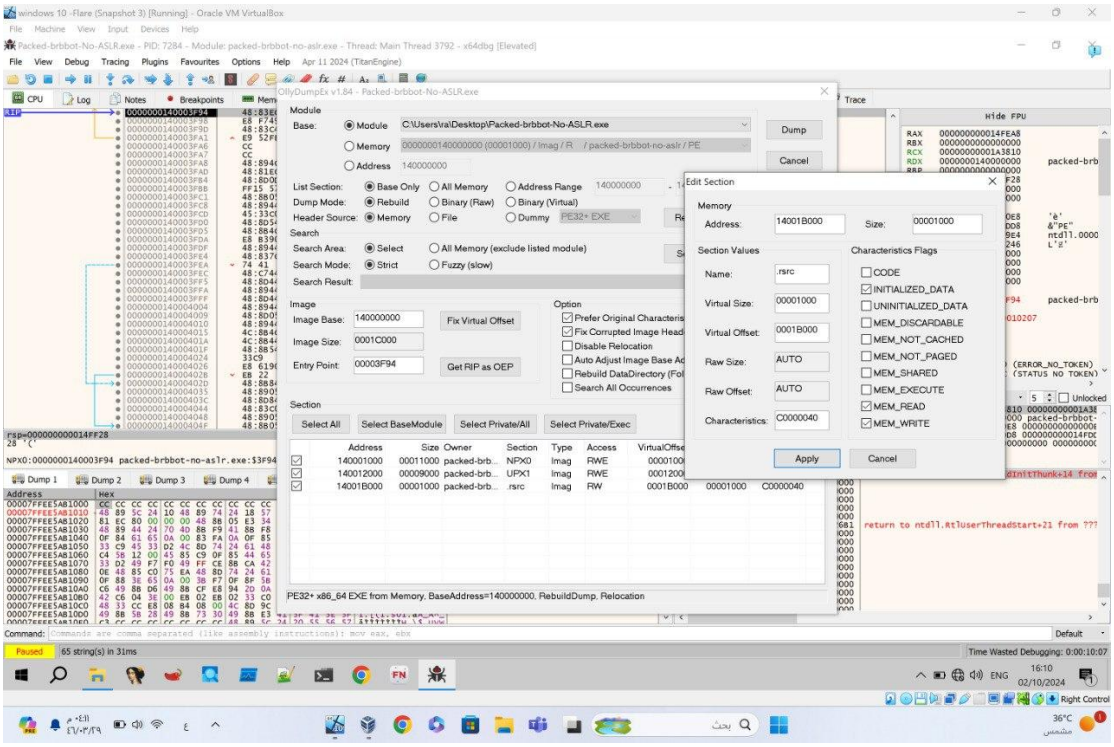


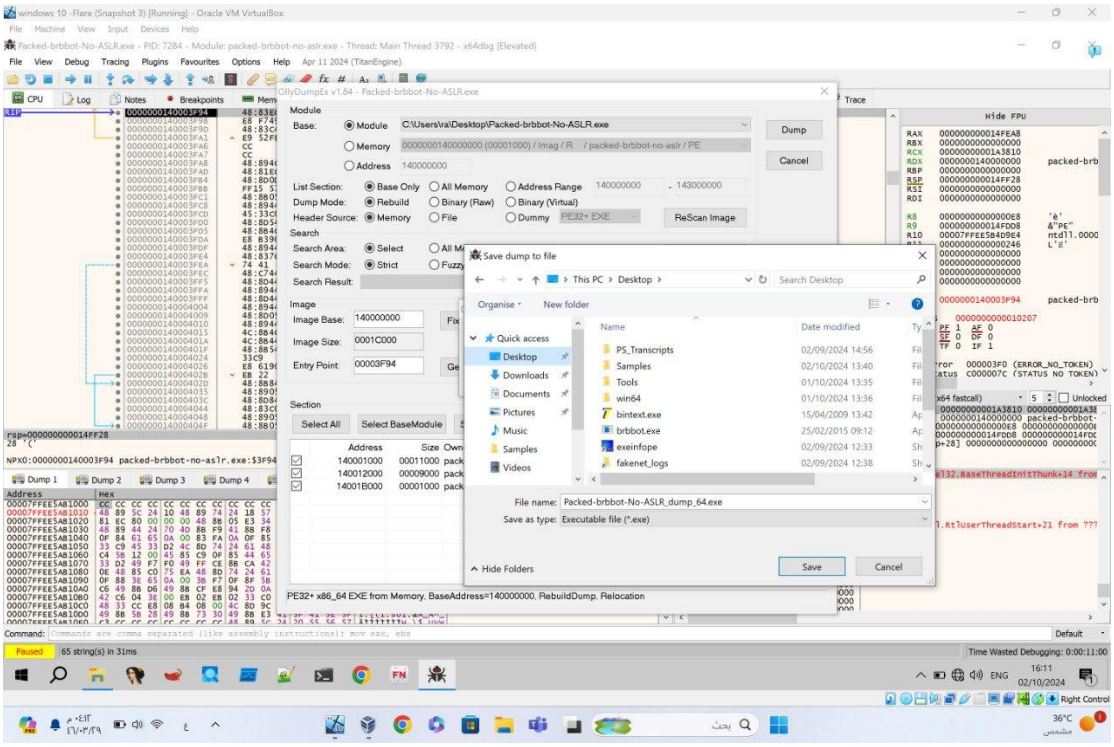
8- Now from the first instruction on the unpacked part I will create a dump.



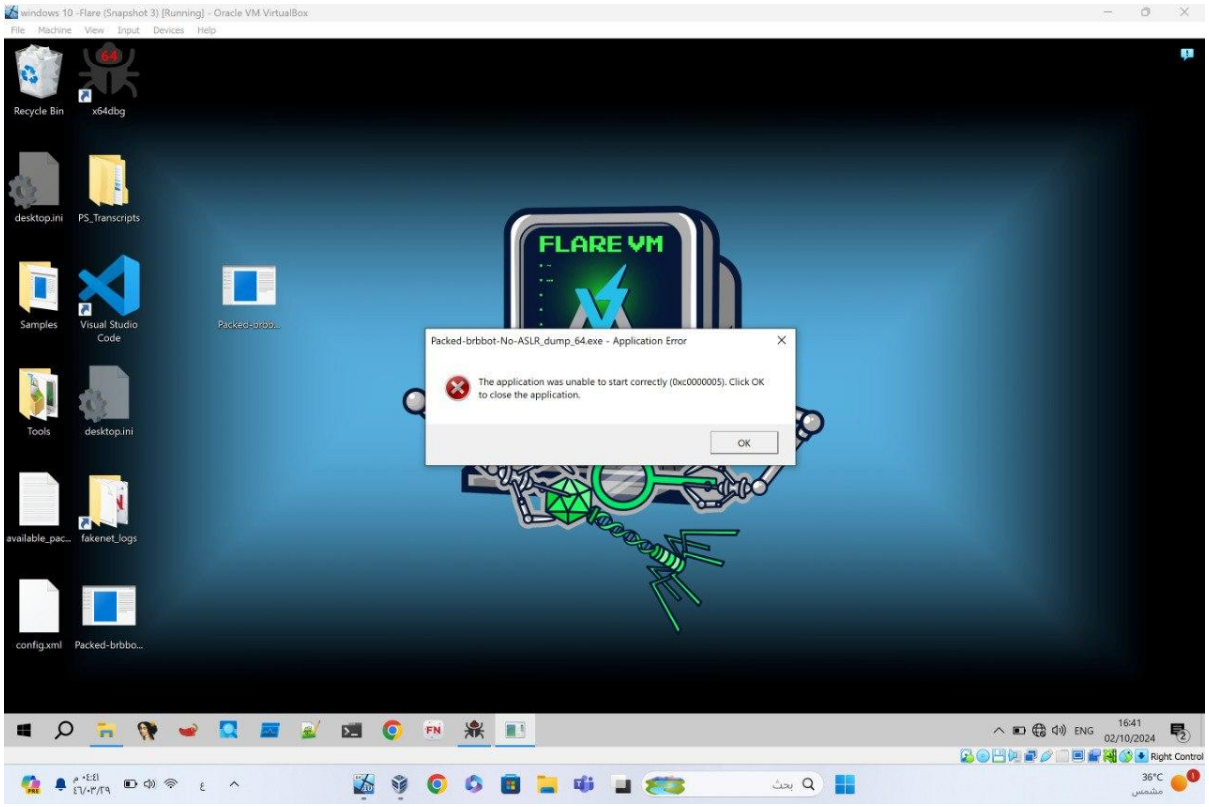
9- Press correcting entry point to be the start of the first instruction from unpacked part rather than original entry point of the file.
10- Then give every section memory write privilege.

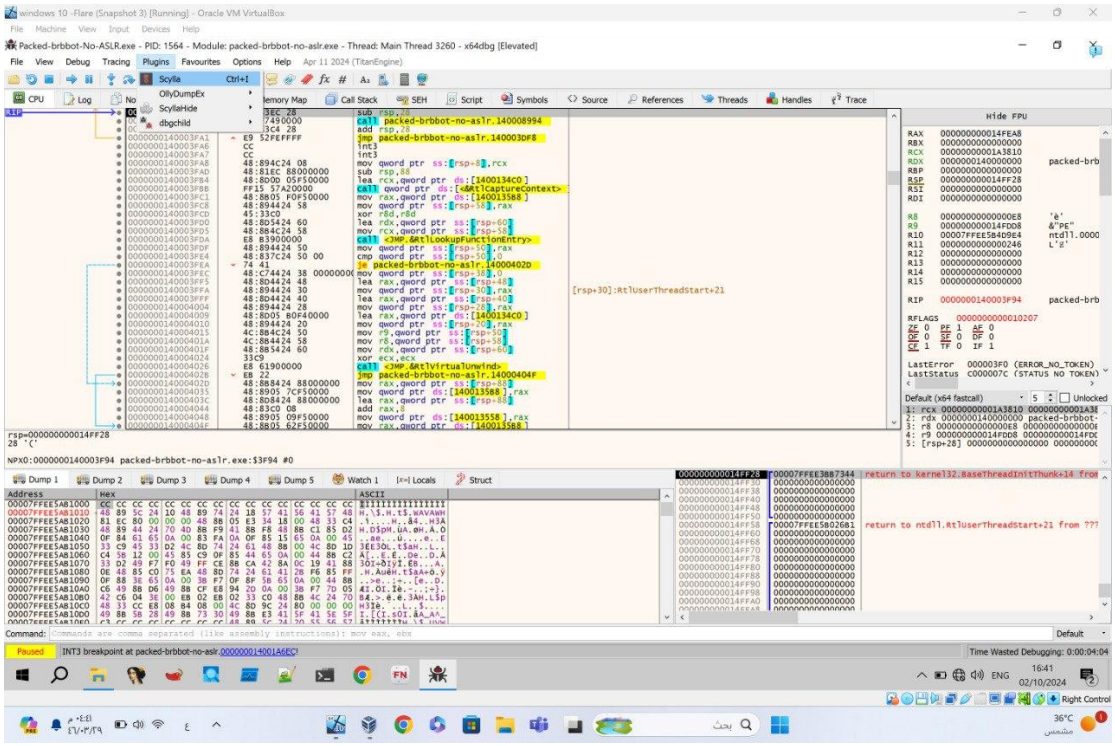


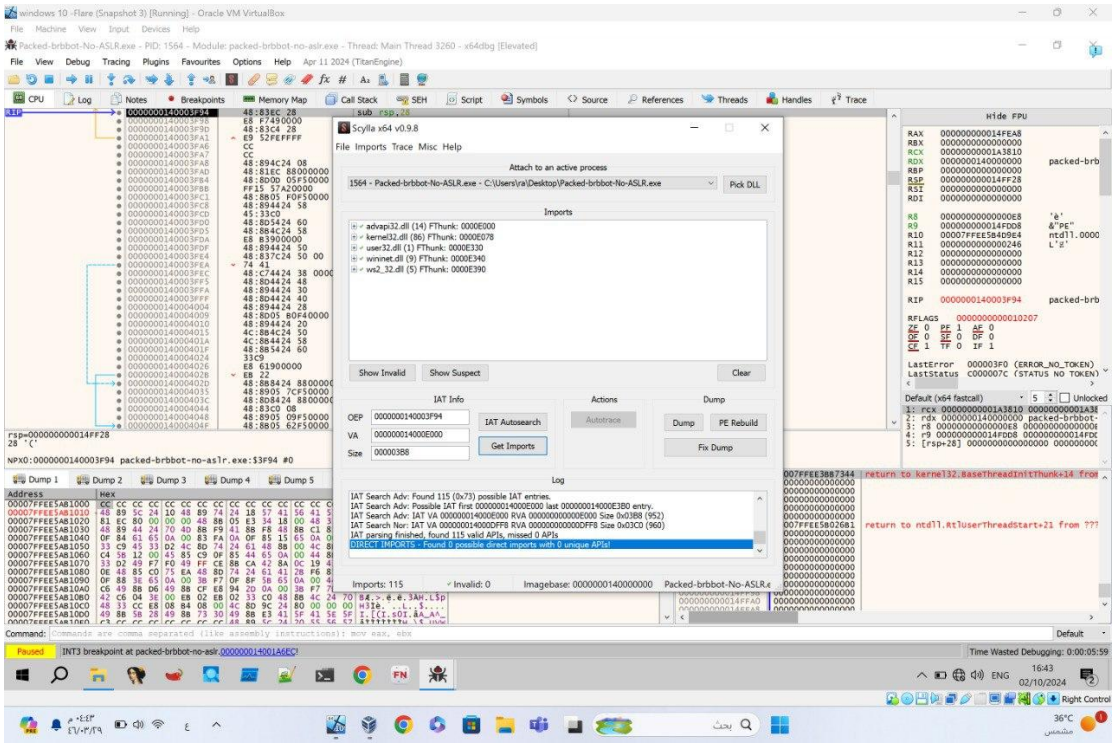


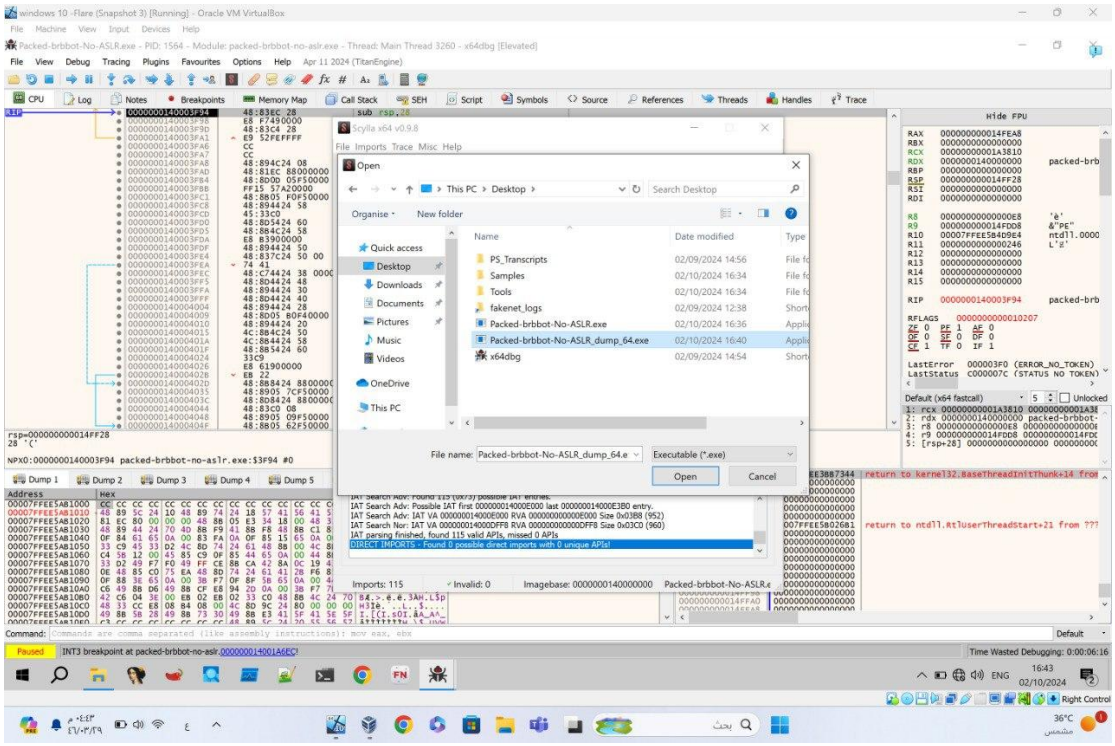


11- I could see it is not working so I will fix it by using Scylla.

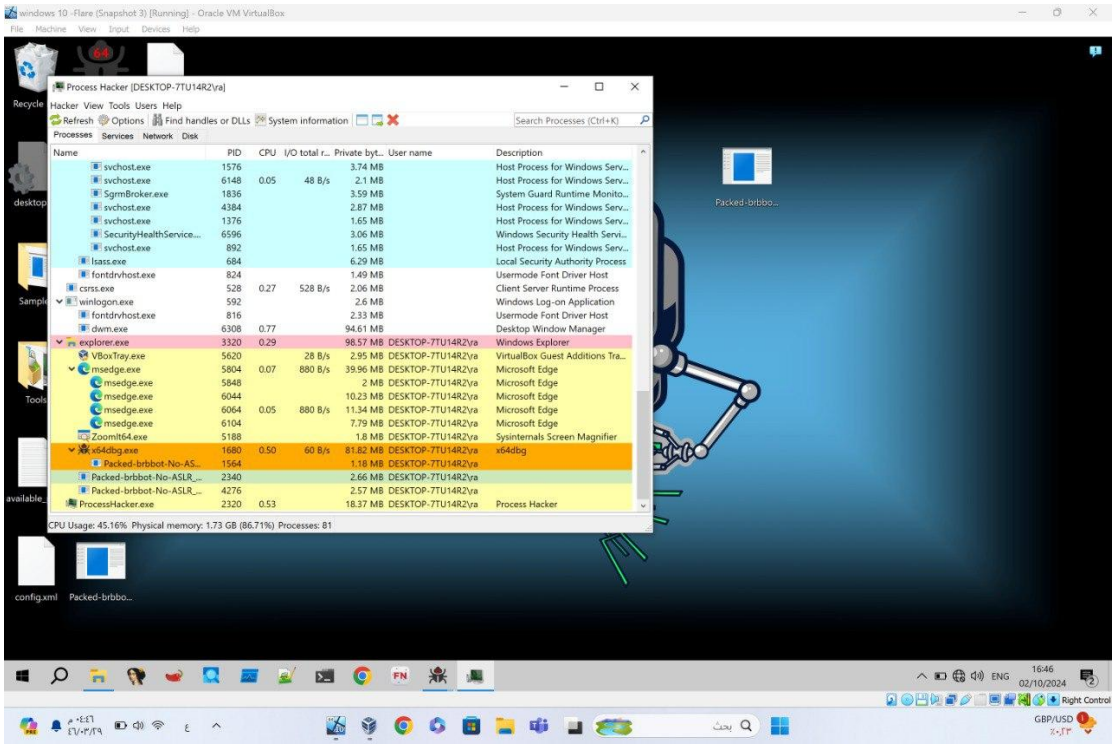


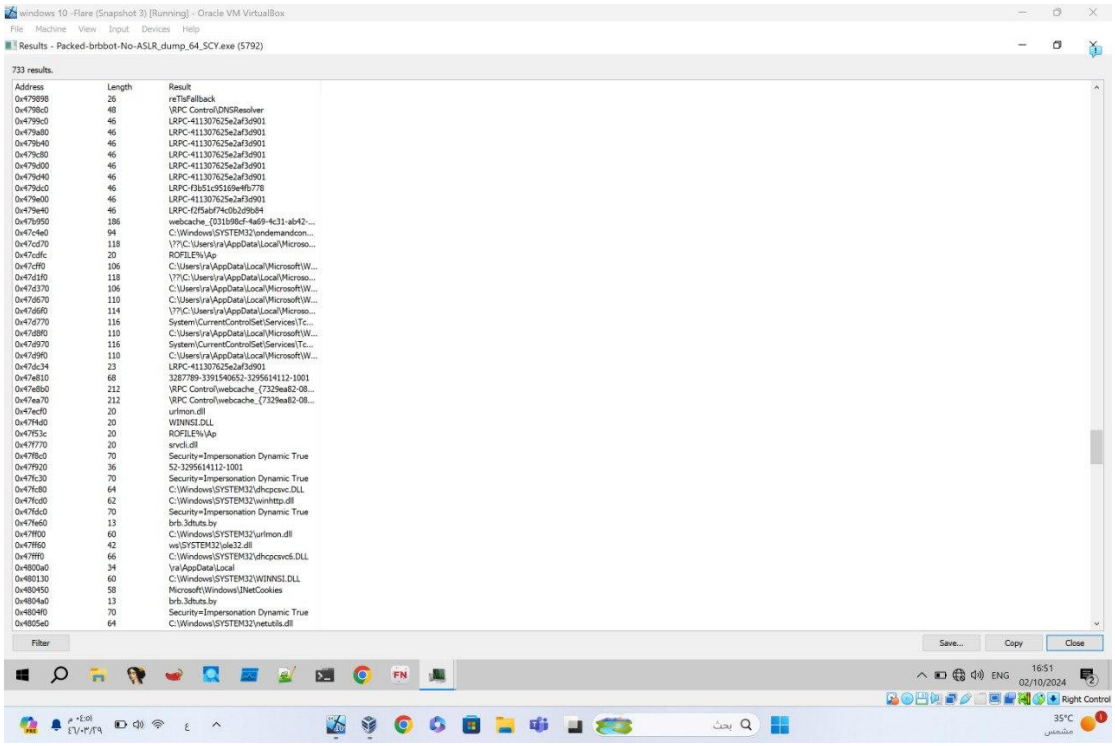




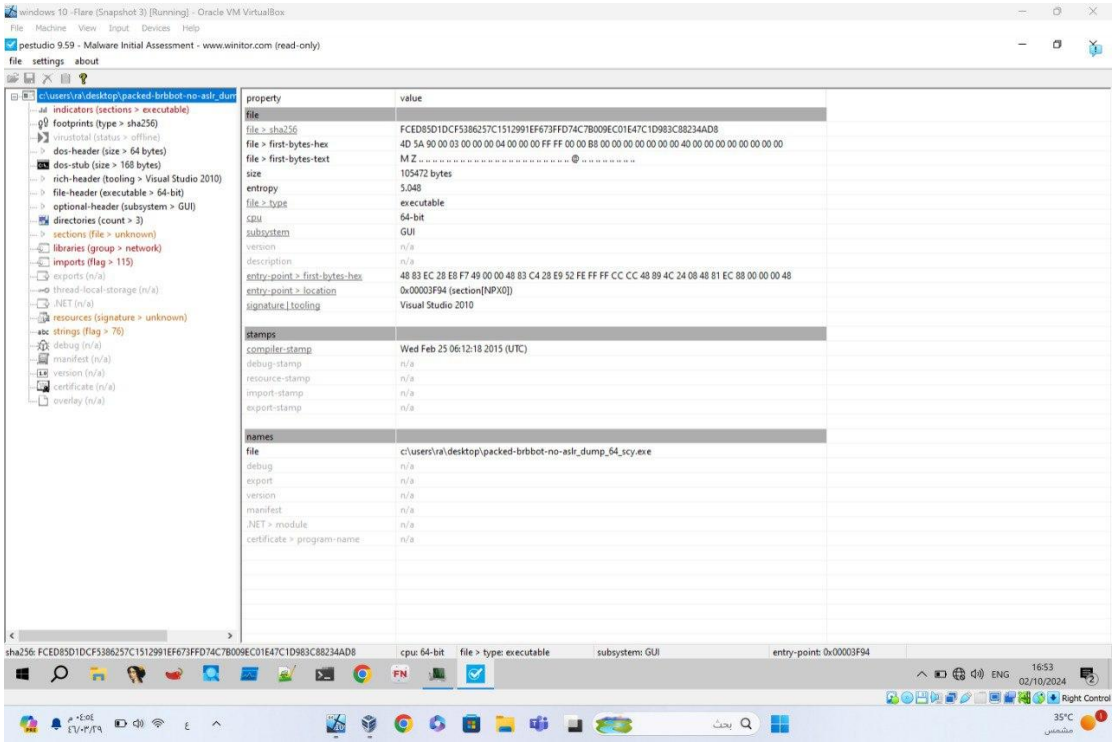


12- Now I could run the malware and analyze it in process hacker.





13- Also, after analyzing it in PE I could see that it has flag number as same as the unpacked malware.



How do unpackers use the VirtualAlloc API?

VirtualAlloc API used by packers to allocate memory in the process's address space to unpack the code. In packed malware, this API is used to allocate space in memory for unpacking the encrypted or compressed sections of the malware. After the malware is unpacked, the malicious code is written to this allocated memory and then executed.

Why do we need to disable the DLL can move option?

To ensure that the DLL stays in a fixed memory location. In debugging packed malware, it's important to know the exact memory locations where the unpacked code resides. If the DLL is allowed to move, then it will be complicated debugging because the addresses of the functions and variables change every time we run the code.

List All the patterns you have recognized in the code that confirms that this is a packed malware.

- Less **number of flags**: Packed malware often strips away unnecessary imports and flags, resulting in a much smaller import table.
- Less **number of strings**: Malware that is packed typically hides most of its strings, making it harder to analyze statically. Only a minimal number of strings will be visible before the malware is unpacked.

Compare the import table in this step with the import table of the packed_brbbot.exe

The packed version of brbbot.exe on the right side has a much smaller import table (only 10 imports), compared to the unpacked version on the left side (115 imports). This difference is a key indicator that the malware was packed, as the original import table is significantly reduced to conceal its true behavior. Once the malware is unpacked, the full list of APIs it relies on becomes visible, revealing its actual capabilities.