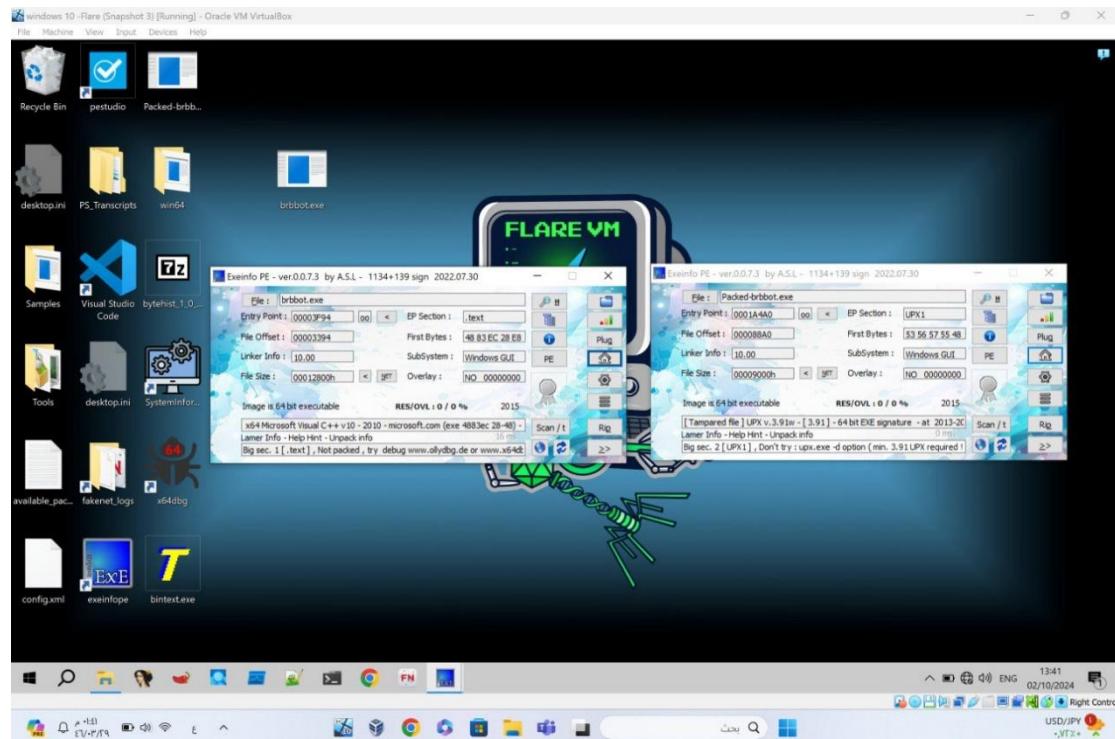


Reverse Engineering and Malware Analysis
Lab 5 – Unpacking Malware Using Debugger

Raghad lafe - 2111941

Task 1: Recognizing Packed Malware:

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



Two screenshots of pestudio 9.59 showing the analysis of brbbot.exe and packed-brbbot.exe.

File Settings about

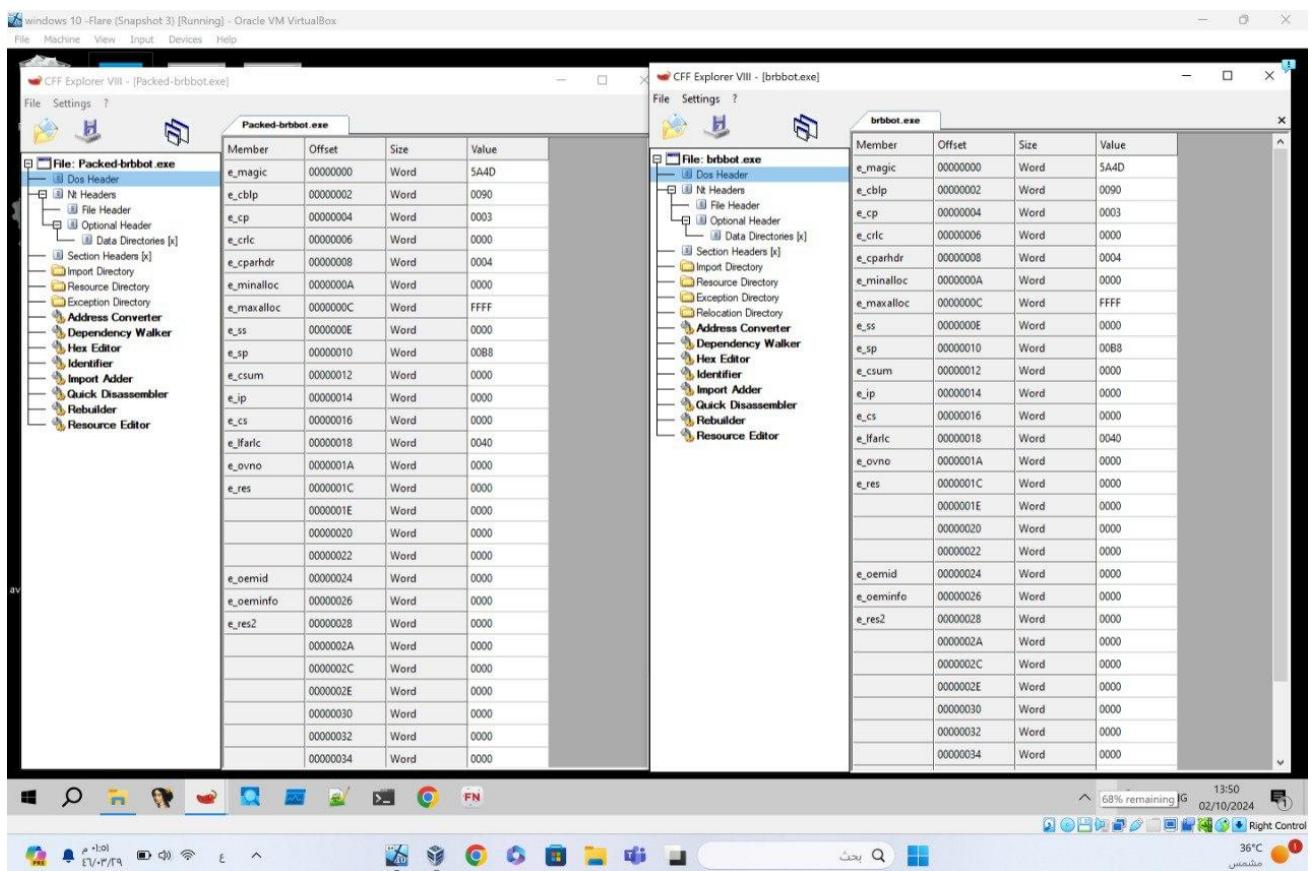
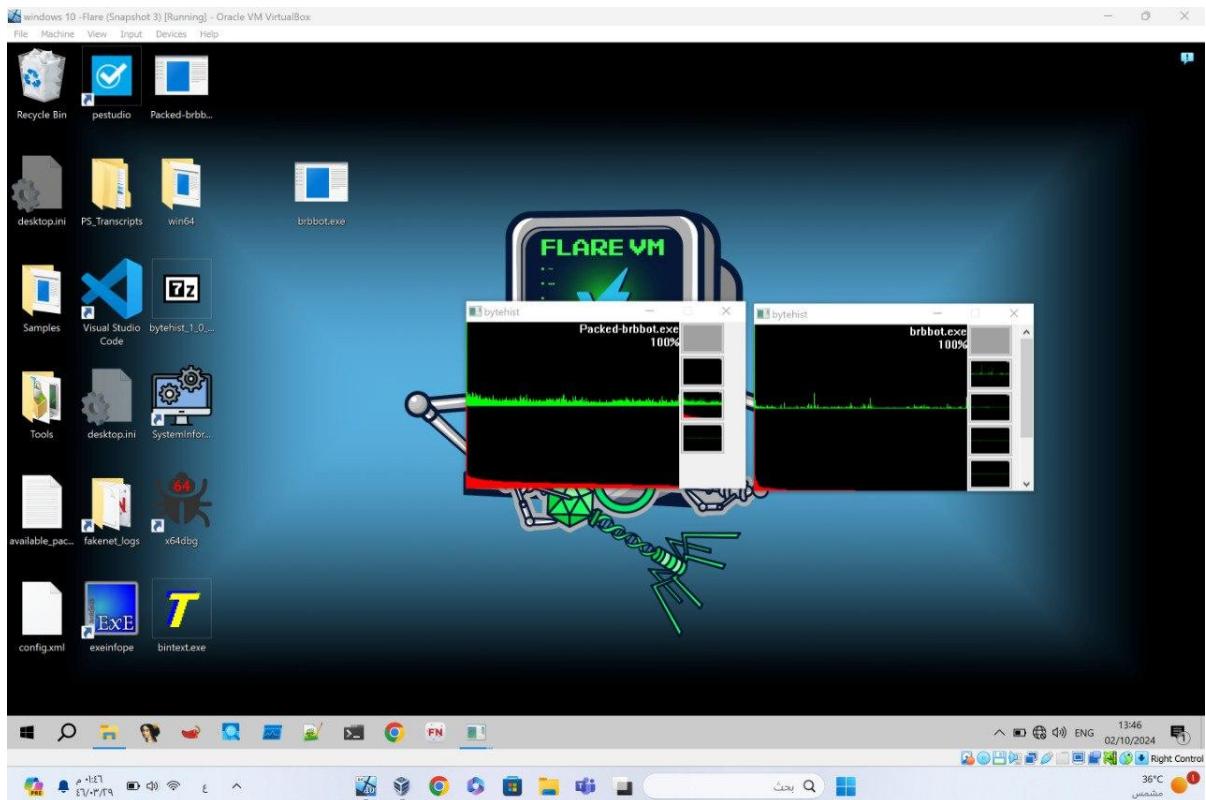
c:\users\ra\Desktop\brbbot.exe

property	value
file > sha256	F47060D0F7DE5EE651878EB18DD2D4
file > first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF
file > first-bytes-text	M Z
size	75776 bytes
entropy	5.918
file > type	executable
CPU	64-bit
subsystem	GUI
version	n/a
description	n/a
entry-point > first-bytes-hex	48 83 EC 28 E8 F7 49 00 00 48 83 C4 28
entry-point > location	0x0003F94 (section[.text])
signature tooling	Visual Studio 2010
stamps	Wed Feb 25 06:12:18 2015 (UTC)
compiler-stamp	n/a
debug-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a
names	
file	c:\users\ra\Desktop\brbbot.exe
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET module	n/a
certificate > program-name	n/a

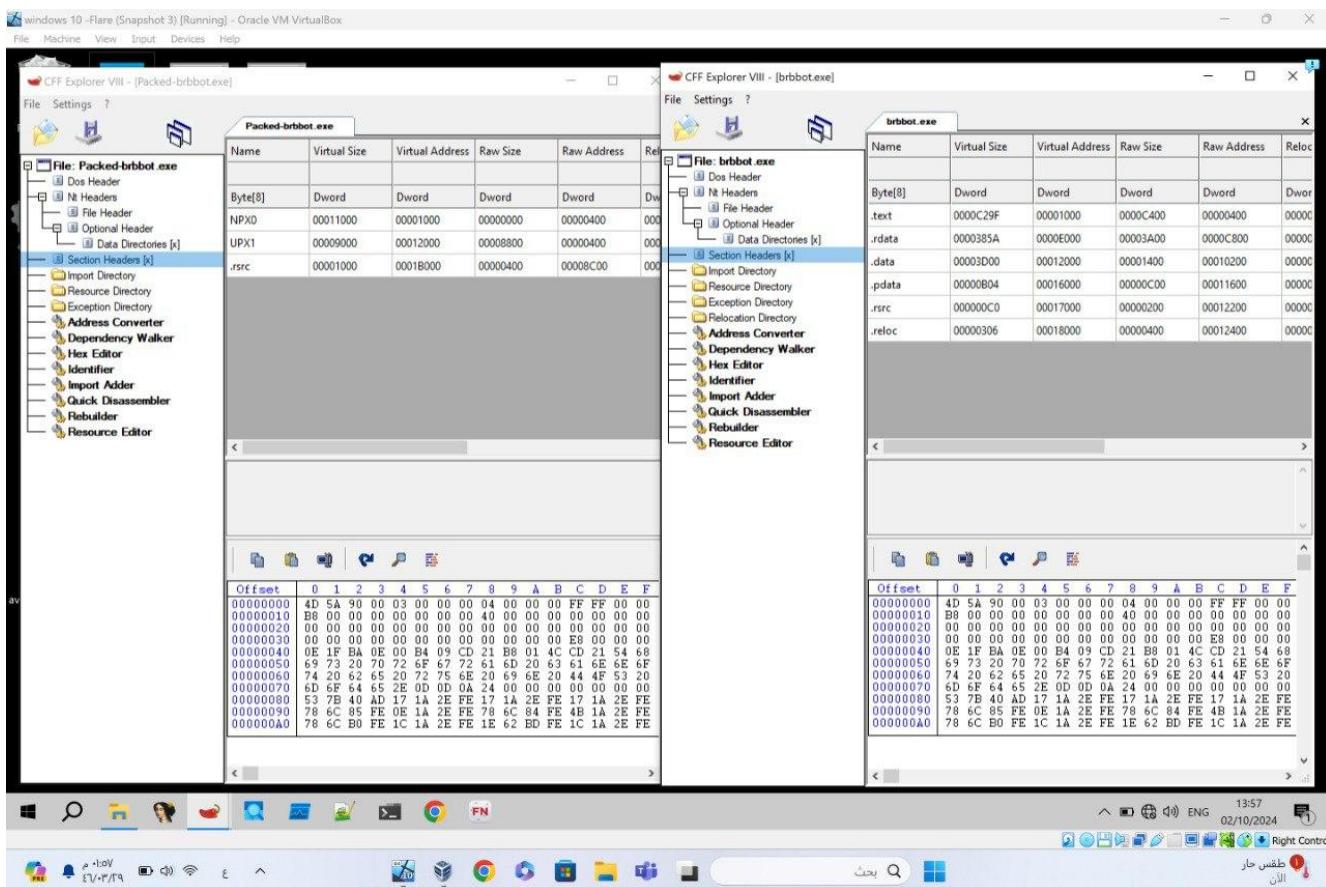
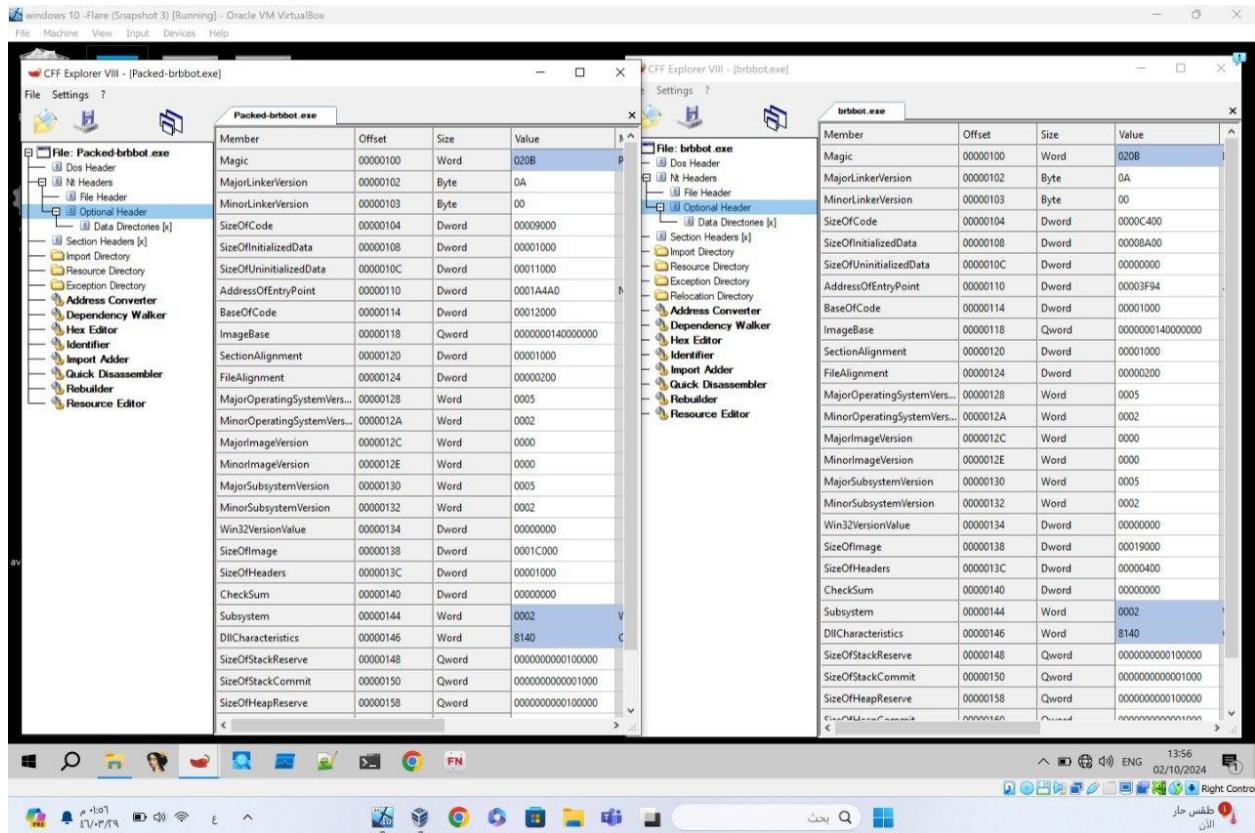
c:\users\ra\Desktop\packed-brbbot.exe

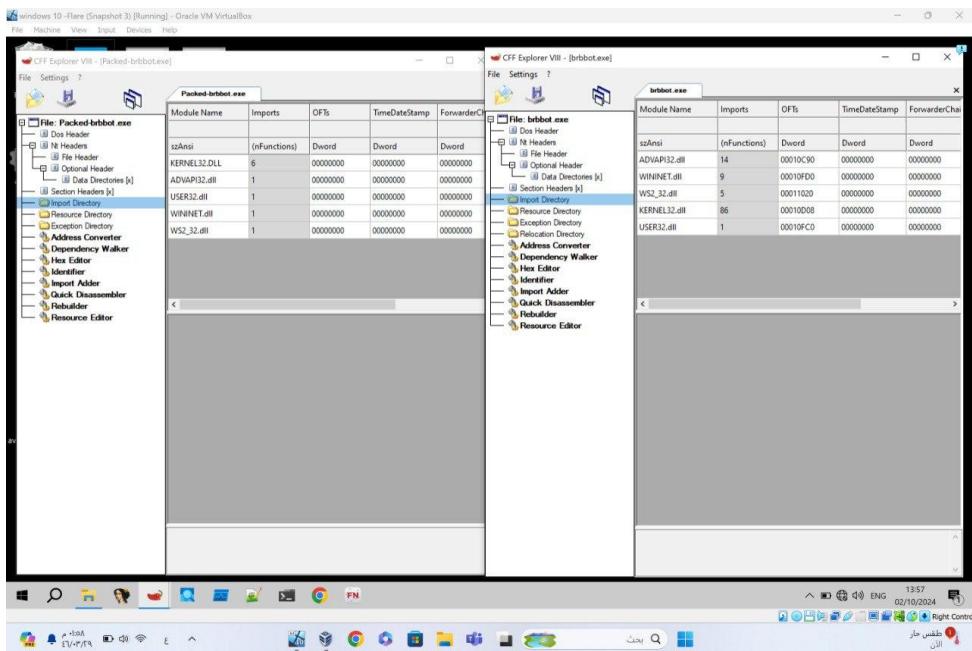
property	value
file > sha256	F9227A44E25A7EE8148E2D0532B148B640F6DC52C85B22A9F4FA7FA037417FA
file > first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF
file > first-bytes-text	M Z
size	36864 bytes
entropy	7.764
file > type	executable
CPU	64-bit
subsystem	GUI
version	n/a
description	n/a
entry-point > first-bytes-hex	53 36 37 35 48 8D 35 55 7F FF 48 8D
entry-point > location	0x0001AA0 (section[UPX1])
signature tooling	Visual Studio 2010
stamps	Wed Feb 25 06:12:18 2015 (UTC)
compiler-stamp	n/a
debug-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a
names	
file	c:\users\ra\Desktop\packed-brbbot.e
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET module	n/a
certificate > program-name	n/a

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.

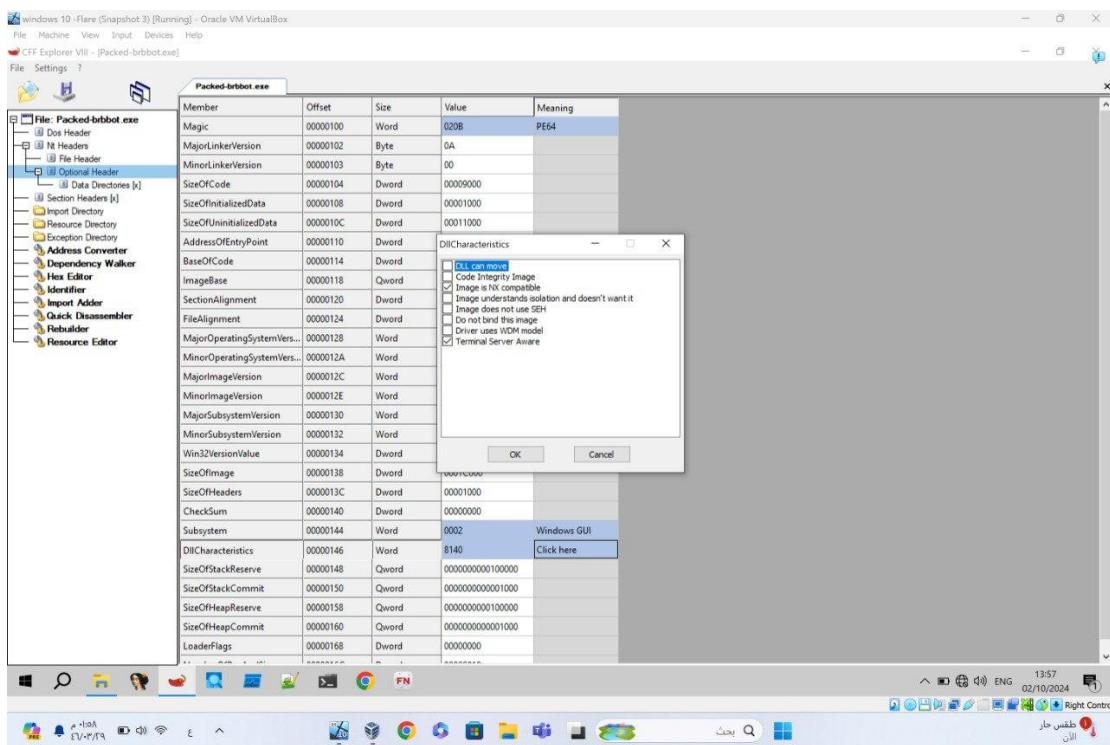


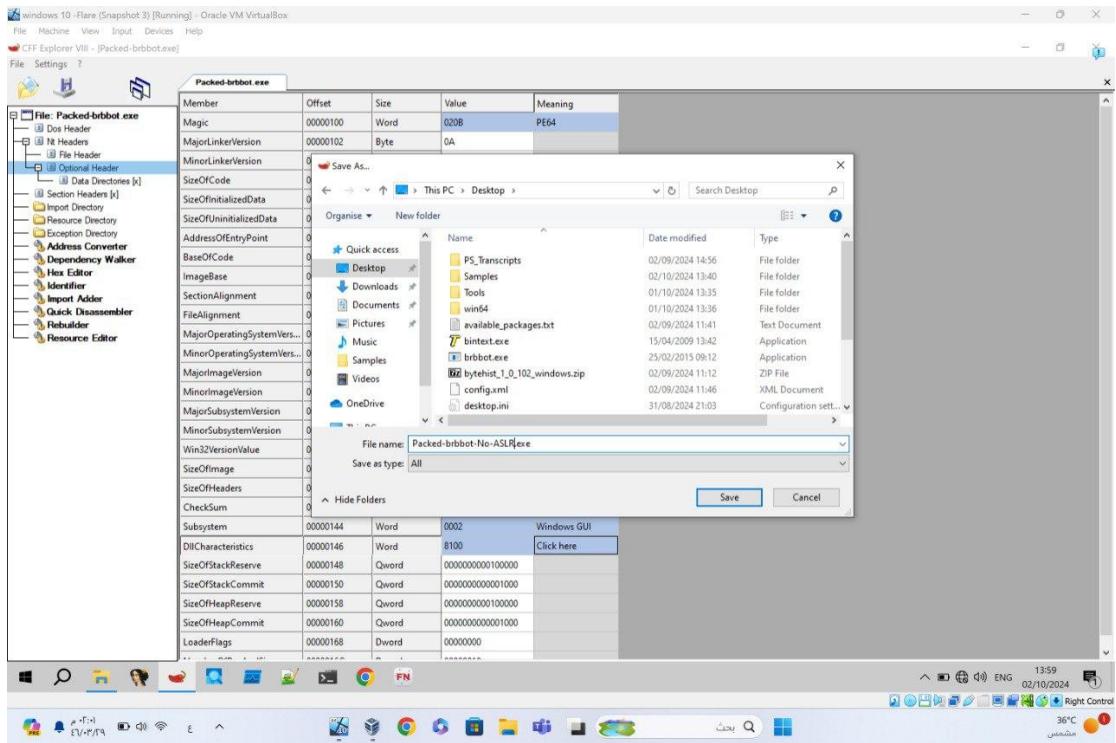
3- From CFF explorer we could see the image Base, sections, import directory.





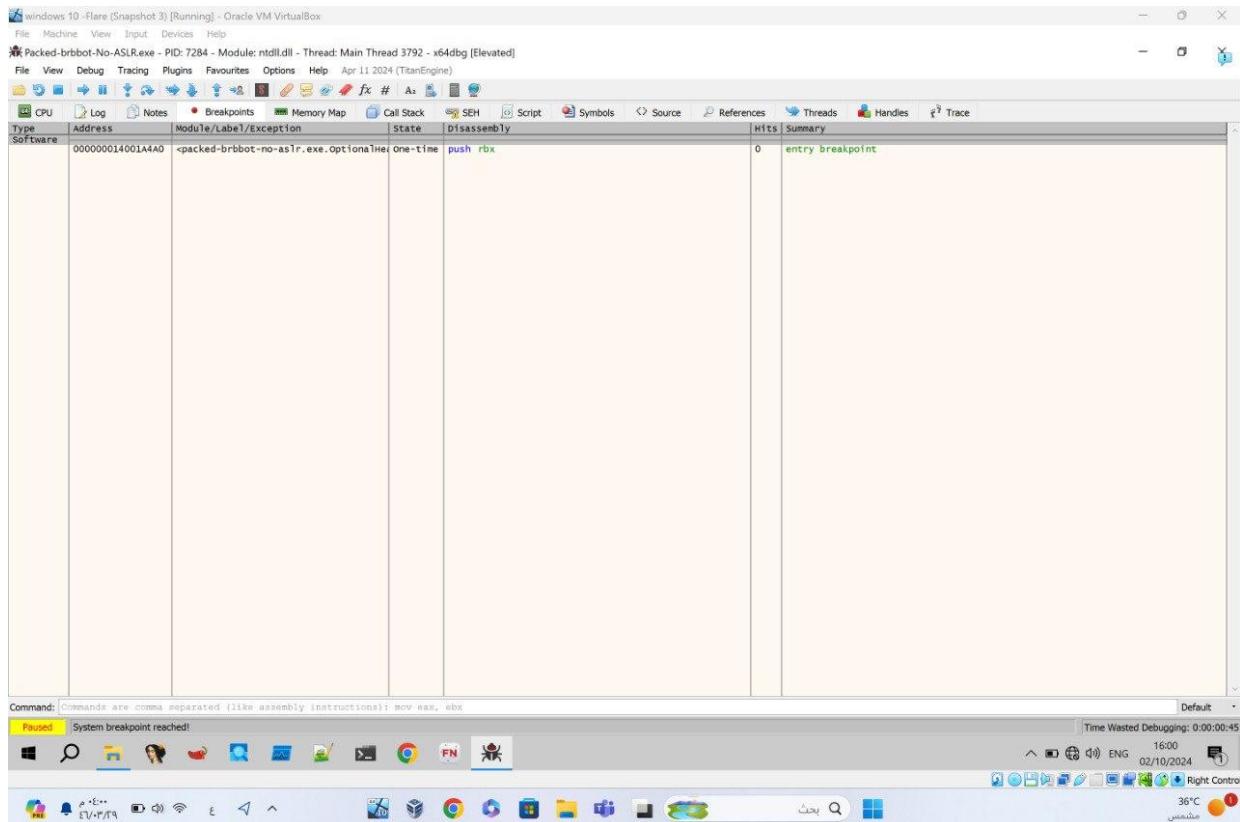
4- Now from CFF going to Disable the ASLR and save a version that has no ASLR so could debug it and unpack it.





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.

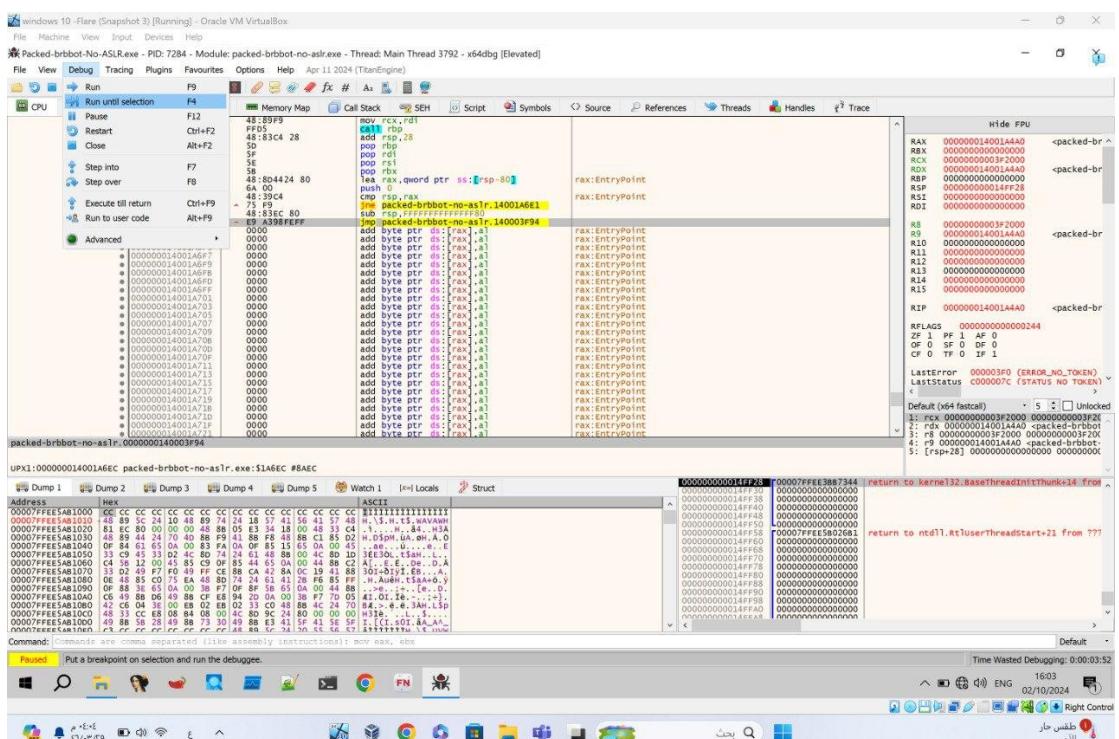
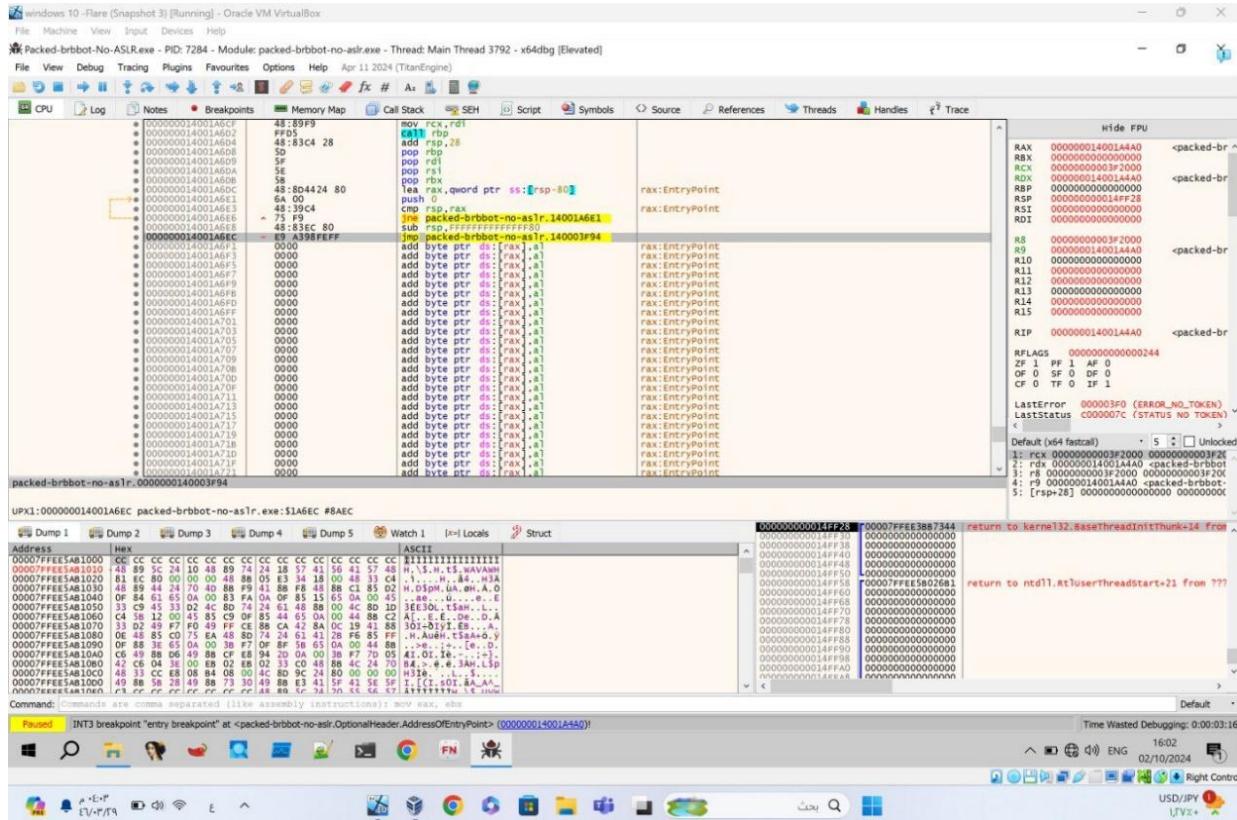


This screenshot shows a debugger session within Oracle VM VirtualBox. The assembly code is displayed in the main pane, and the CPU dump is visible in the bottom pane. The status bar shows the command 'Command: [instructions] mov eax, ebx' and the time 'Time Wasted Debugging: 0:00:02:08'.

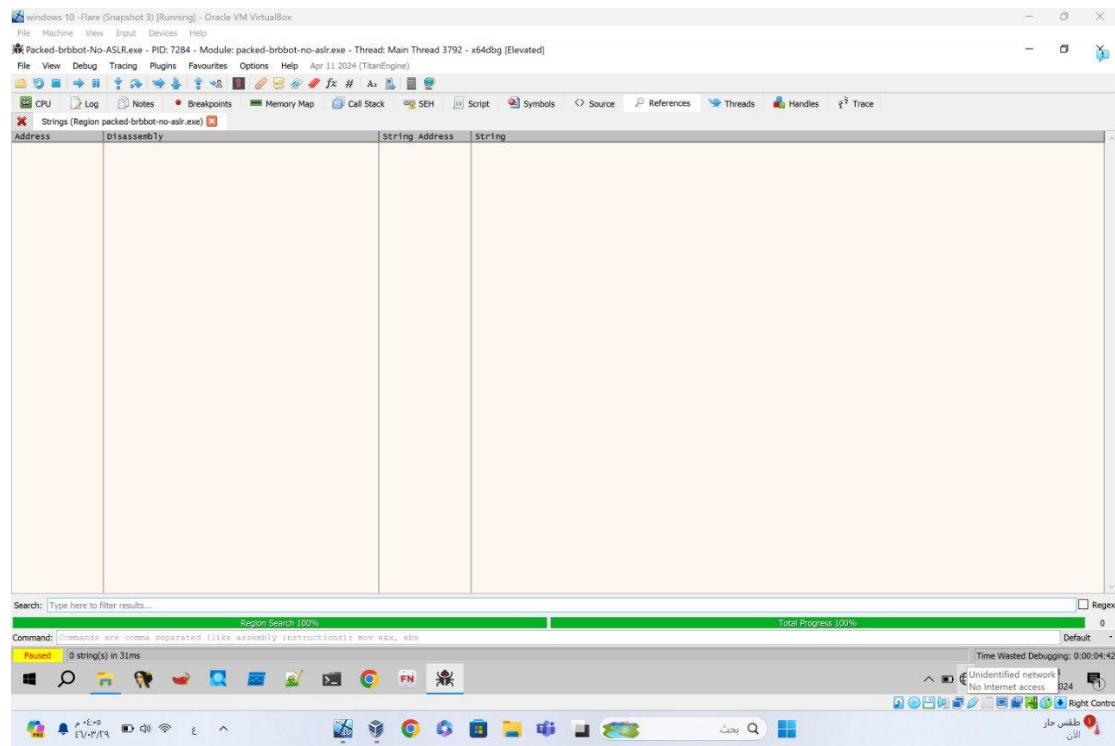
3- Here I could notice the first thing after running the malware is push registers which indicate that the malware is packed.

The screenshot shows the x64dbg debugger interface. The assembly pane displays the packed-brbbot-no-ASLR executable's code, highlighting various assembly instructions and memory addresses. The memory dump pane below shows the memory dump starting at address 00007FFEE5A81000, with columns for Address, Hex, Dump 2, Dump 3, Dump 4, Dump 5, Watch 1, Locals, and Struct. A command line at the bottom allows for assembly-level commands like mov eax, ebx. The status bar at the bottom right indicates the time as 00:02:11 and the date as 02/10/2024.

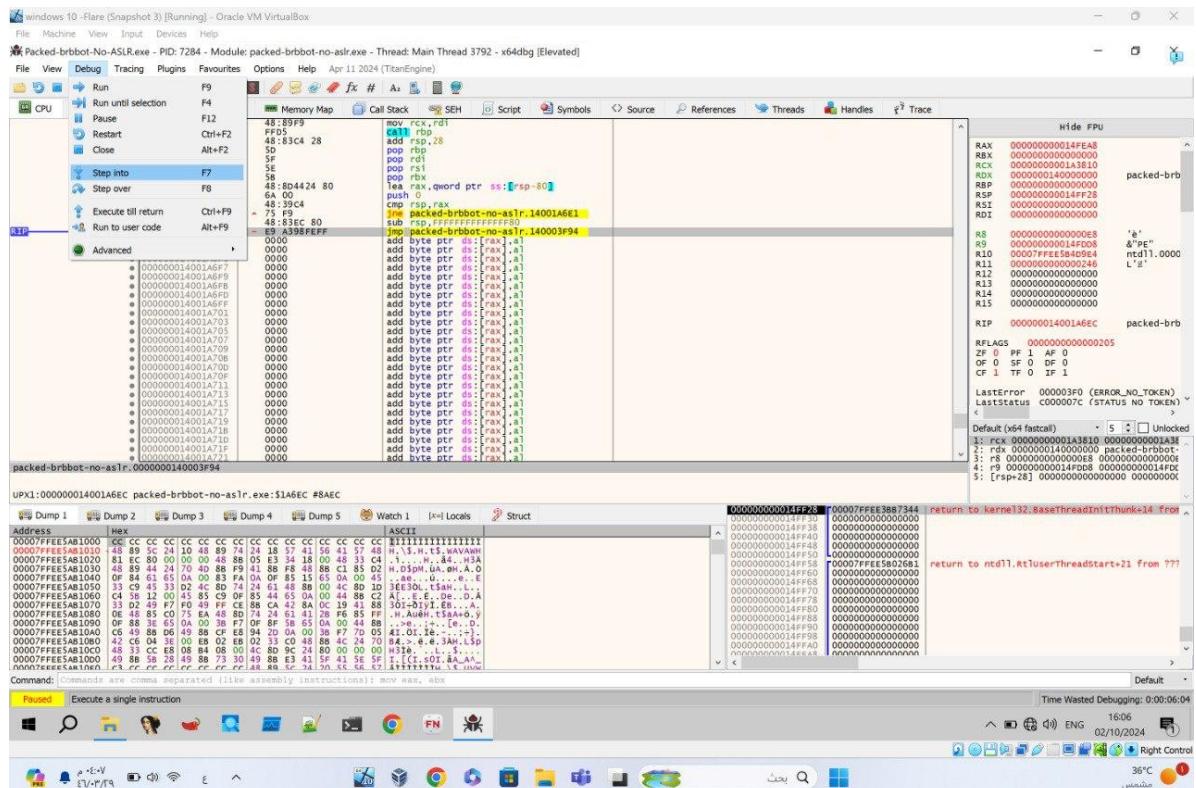
- 4- I scroll down till I saw zeros address that indicates to me the end of unpacking code of the malware.
 - 5- So, I could conclude that the jmp instruction would take me to the unpacked part of the malware.



- 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.

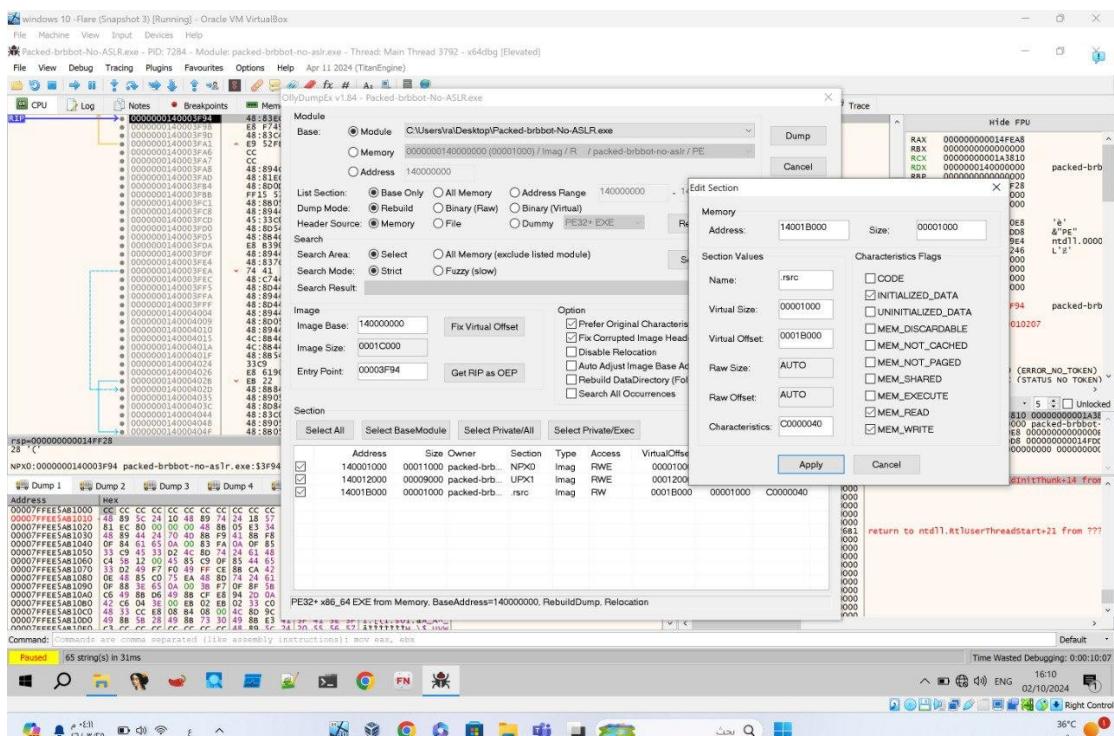
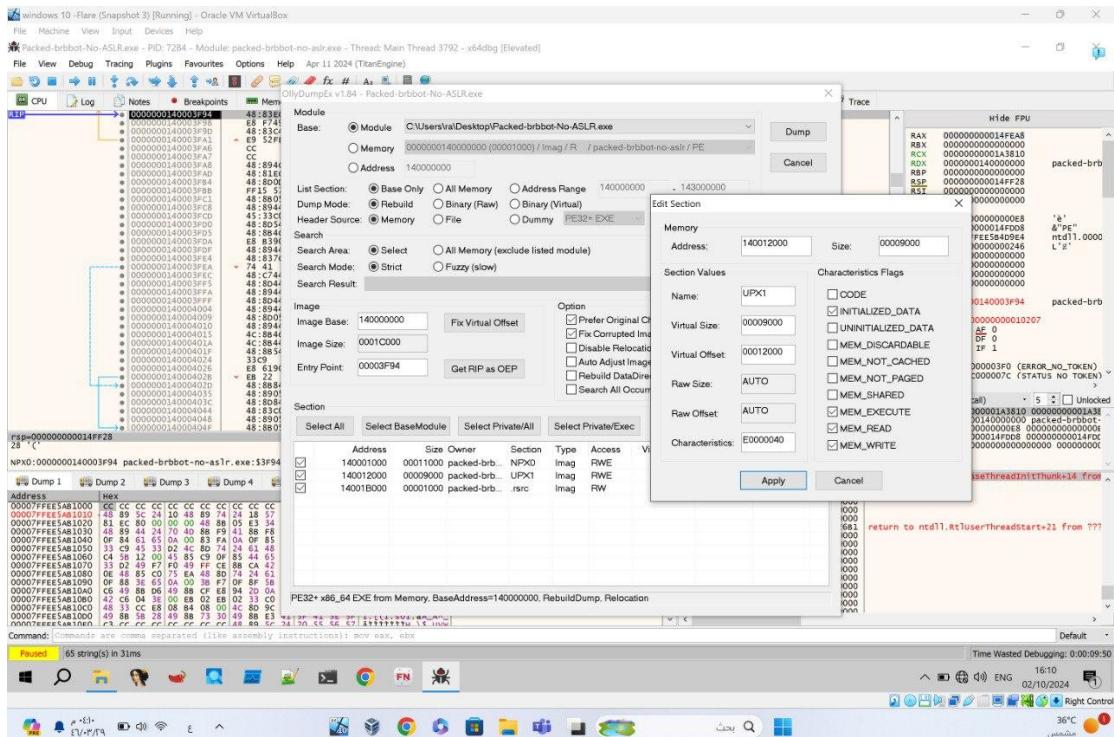


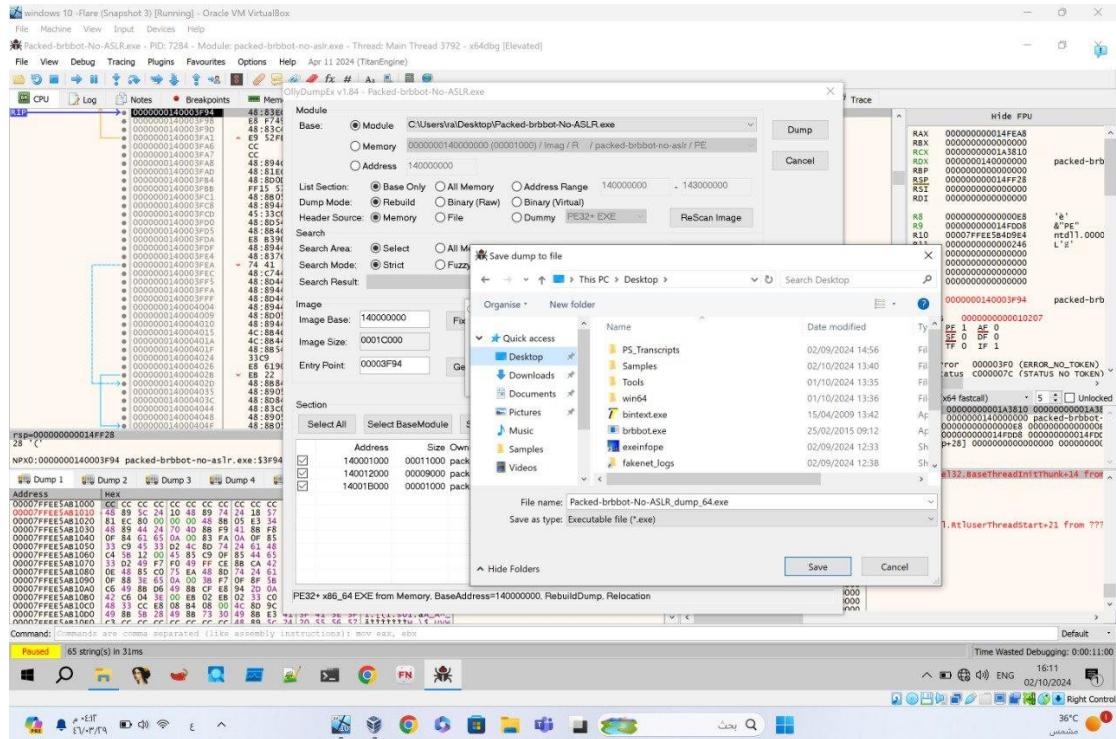
The screenshot shows the x64dbg debugger interface with the following details:

- Title Bar:** Windows 10 - Rflare (Snapshot 3) [Running] - Oracle VM VirtualBox
- Menu Bar:** File, Machine, View, Input, Devices, Help
- Toolbar:** Packed-brbbot-No-ASLR.exe - PID: 728 - Module: packed-brbbot-no-aslr.exe - Thread: Main Thread 3792 - x64dbg [Elevated]
- Breakpoints:** 65
- Registers:** CPU, Log, Notes, Breakpoints, Memory Map, Call Stack, SEH, Script, Symbols, Source, References, Threads, Handles, Trace
- Address:** 0x0000014000128F
- String:** Region packed-brbbot-no-aslr.exe
- String Address:** 0x0000014000128F
- String:** "String" "CONFIG" "url" "brbconfig.tmp" "rurl" "ext" "cor" "sleep" "%2a" "%31-%d&%s&%s" "%c%d&%s" "#%04\$%42\$%08A" "BA" "%2a" "AppB" "Software\Microsoft\Windows\CurrentVersion\Run" "brbbot" "brbconfig.tmp" "Software\Microsoft\Windows\CurrentVersion\Run" "L'Microsoft Enhanced Cryptographic Provider v1.0'" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)" "HTTP/1.1" "Connection: close\r\n" "brbconfig.tmp" "brbconfig.dll" "ZwQuerySystemInformation" "&(null)" "&(null)" "&(null)" "&(null)" "&(null)" "&(null)" "&(null)" "CorExitProcess" "L'RunProgramW\%1\%Program:" "L'program name unknown'" "L'<null>" "L'\n'"
- Search:** Type here to filter results...
- Region Search:** Region Search 100%
- Total Progress:** 00%
- Command:** Commands are comma separated (like assembly instructions): mov eax, ebx
- Pause:** Paused 65 string(s) in 31ms
- Bottom Taskbar:** Icons for various applications like File Explorer, Task Manager, Task View, and Start.

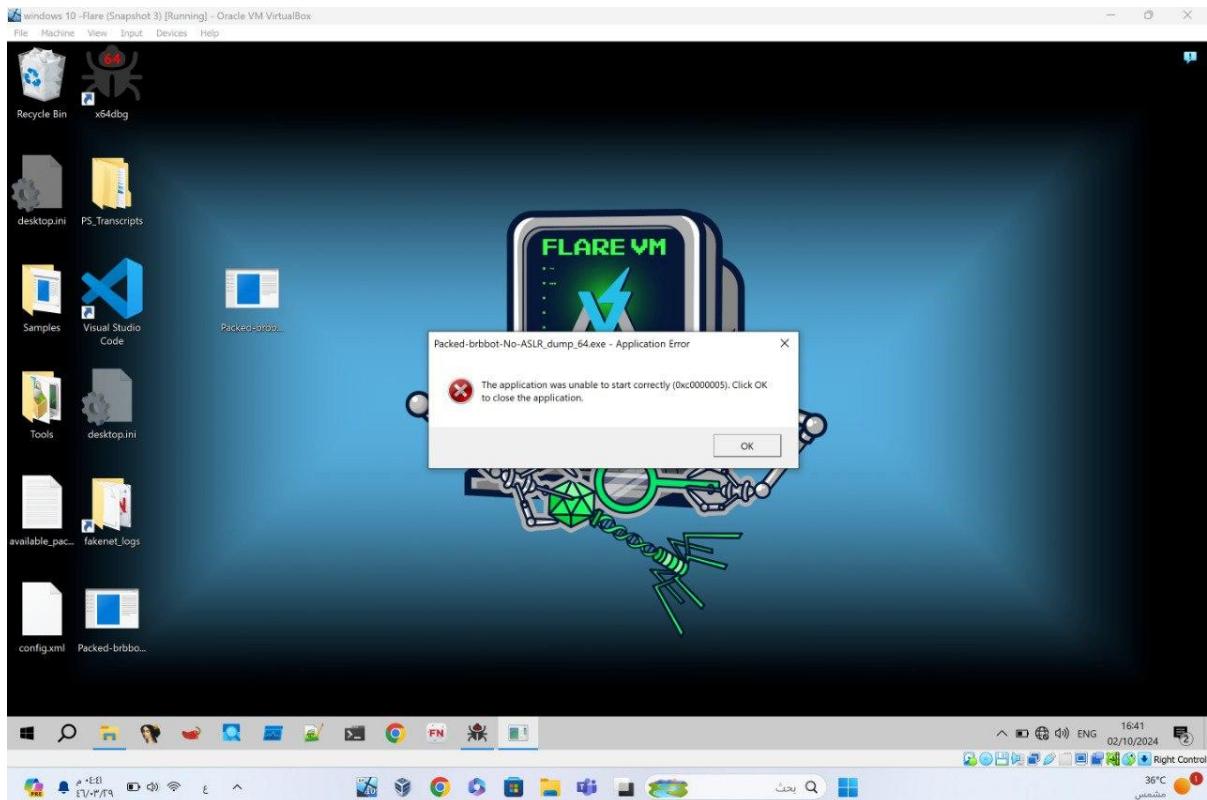
8- Now from the first instruction on the unpacked part I will create a dump.

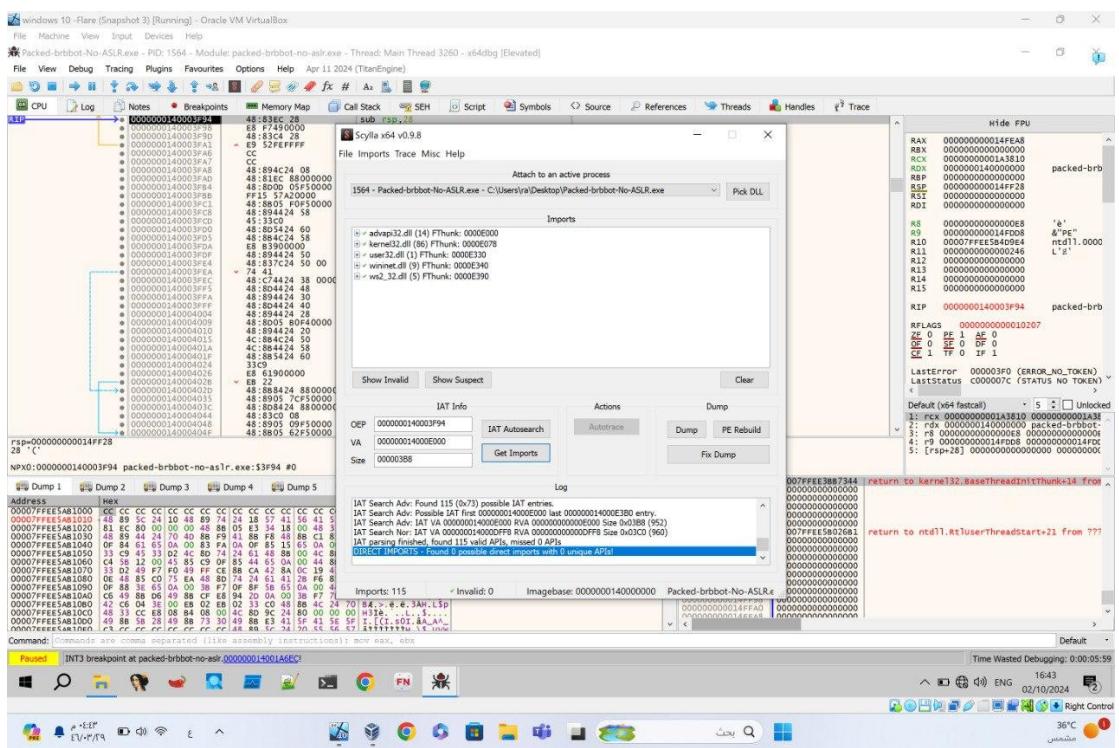
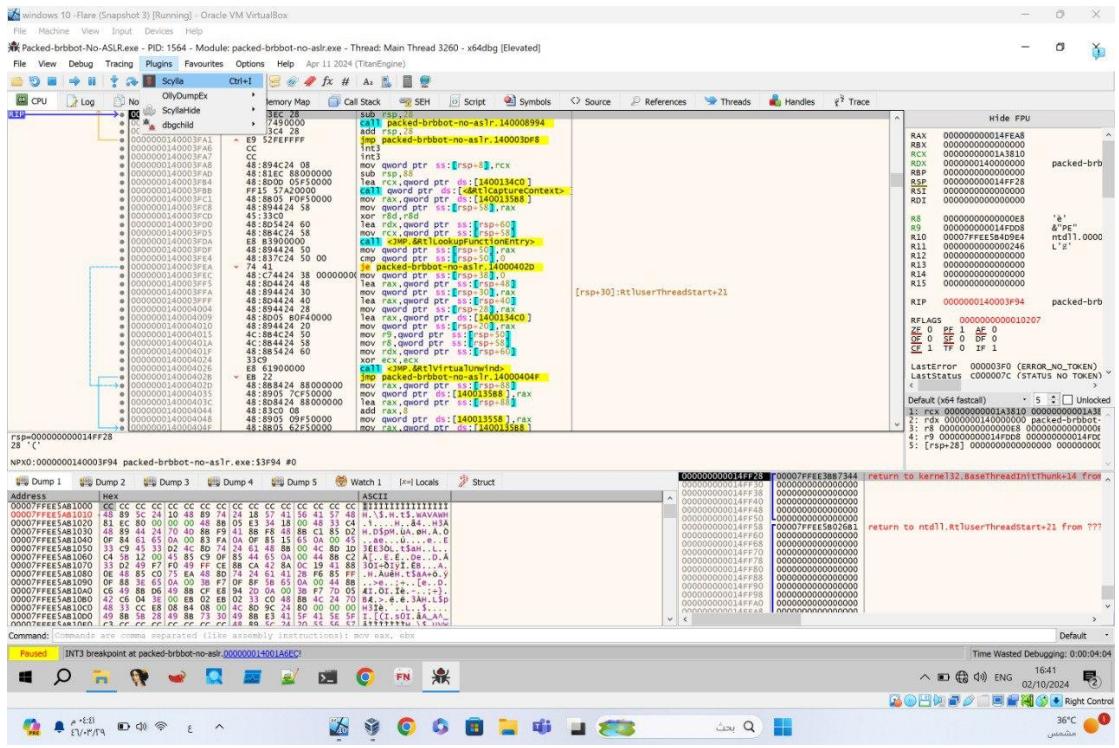
- 9- Press correcting entry point to be the start of the first instruction from unpacked part rather than original entry point of the file.
 10- Then give every section memory write privilege.

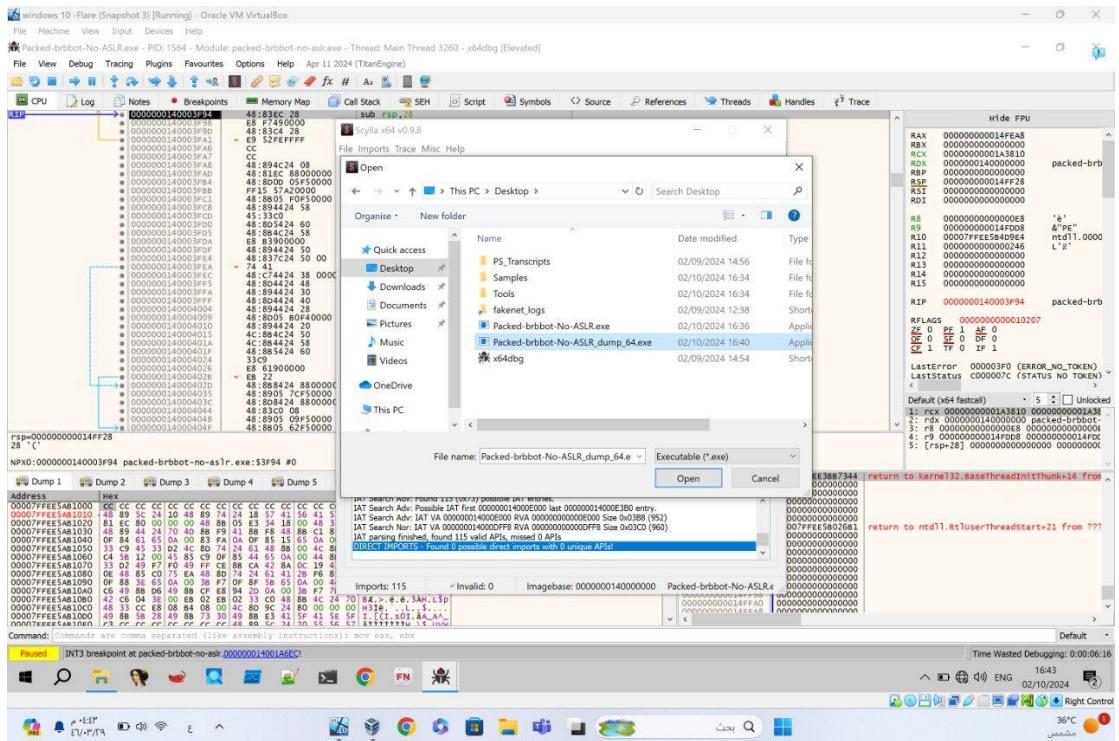




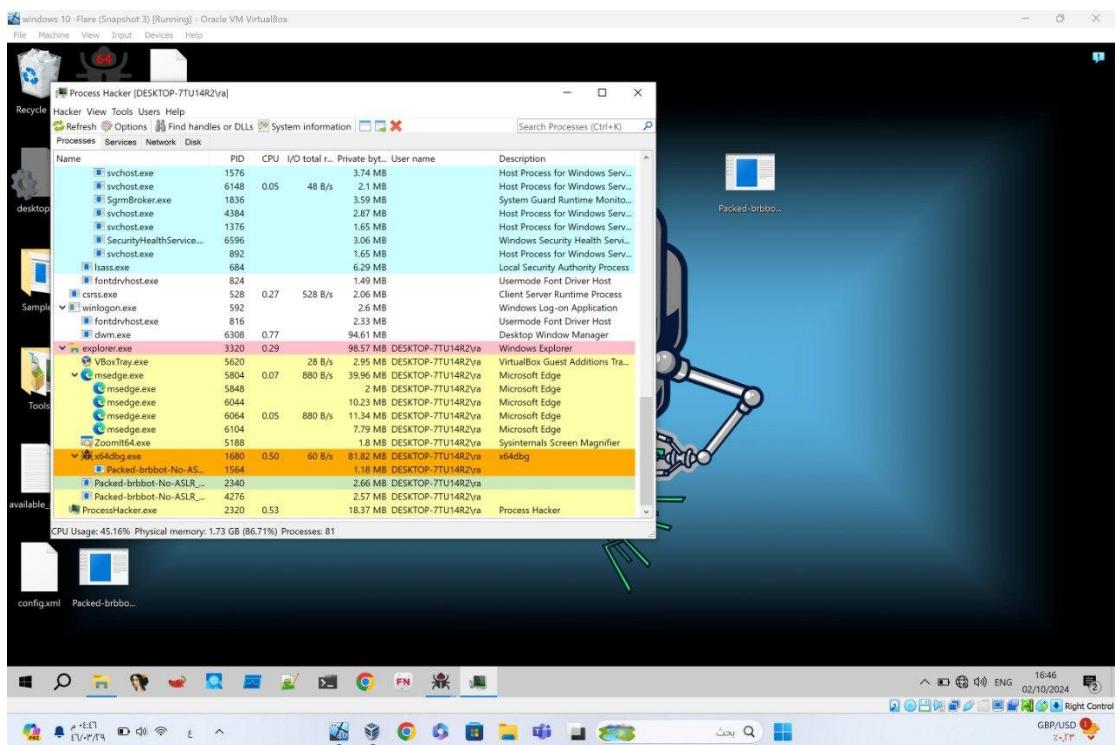
11- I could see it is not working so I will fix it by using Scylla.







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



File Machine View Input Devices Help

Results - Packed-brbbot-No-ASLR.dump.64.SCY.exe (5792)

733 results.

Address	Length	Result
0x47998	26	r7Tn!f!l!b!k
0x479c0	48	RPC-41137625e2a3f901
0x479d0	46	RPC-41137625e2a3f901
0x479e0	46	RPC-41137625e2a3f901
0x479f0	46	RPC-41137625e2a3f901
0x479g0	46	RPC-41137625e2a3f901
0x479h0	46	RPC-41137625e2a3f901
0x479i0	46	RPC-41137625e2a3f901
0x479j0	46	RPC-41137625e2a3f901
0x479k0	46	RPC-41137625e2a3f901
0x479l0	46	RPC-41137625e2a3f901
0x479m0	46	RPC-41137625e2a3f901
0x479n0	46	RPC-41137625e2a3f901
0x479o0	46	RPC-41137625e2a3f901
0x479p0	46	RPC-41137625e2a3f901
0x479q0	46	RPC-41137625e2a3f901
0x479r0	46	RPC-41137625e2a3f901
0x479s0	46	RPC-41137625e2a3f901
0x479t0	46	RPC-41137625e2a3f901
0x479u0	46	RPC-41137625e2a3f901
0x479v0	46	RPC-41137625e2a3f901
0x479w0	46	RPC-41137625e2a3f901
0x479x0	46	RPC-41137625e2a3f901
0x479y0	46	RPC-41137625e2a3f901
0x479z0	46	RPC-41137625e2a3f901
0x479{0	46	RPC-41137625e2a3f901
0x479 0	46	RPC-41137625e2a3f901
0x479~0	46	RPC-41137625e2a3f901
0x479`0	46	RPC-41137625e2a3f901
0x479@0	46	RPC-41137625e2a3f901
0x479#0	46	RPC-41137625e2a3f901
0x479\$0	46	RPC-41137625e2a3f901
0x479%0	46	RPC-41137625e2a3f901
0x479<0	46	RPC-41137625e2a3f901
0x479>0	46	RPC-41137625e2a3f901
0x479^0	46	RPC-41137625e2a3f901
0x479_0	46	RPC-41137625e2a3f901
0x479{10	106	C:\Windows\Temp\{AppData}\Local\Microsoft...
0x479{11	118	\??\C:\Users\r4\{AppData}\Local\Microsoft...
0x479{12	106	\??\C:\Users\r4\{AppData}\Local\Microsoft...
0x479{13	110	C:\Users\r4\{AppData}\Local\Microsoft\W...
0x479{14	114	\??\C:\Users\r4\{AppData}\Local\Microsoft...
0x479{15	116	System\CurrentControlSet\Services\...
0x479{16	119	C:\Windows\Temp\{AppData}\Local\Microsoft\W...
0x479{17	116	System\CurrentControlSet\Services\Te...
0x479{18	110	C:\Users\r4\{AppData}\Local\Microsoft\W...
0x47d34	23	RPC-41137625e2a3f901
0x47e10	68	32B799-3391540652-329561412-1001
0x47e20	212	[RPC-Content]Webcache_(7329e2-08...
0x47e30	212	[RPC-Content]Webcache_(7329e2-09...
0x47e40	20	urimon.dll
0x47e50	20	WINNSI.DLL
0x47e60	20	ROFILE%Ap
0x47e70	20	svd.dll
0x47e80	70	Security=Impersonation Dynamic True
0x47e90	36	S2-329961412-1001
0x47e{0	70	Security=Impersonation Dynamic True
0x47e{1	64	C:\Windows\SYSTEM32\dhcpcsvc.DLL
0x47e{2	62	C:\Windows\SYSTEM32\twinkts.DLL
0x47e{3	70	Security=Impersonation Dynamic True
0x47e{4	13	brb_3duts_by
0x47f{0	60	C:\Windows\SYSTEM32\urimon.dll
0x47f{1	42	val\SYSTEM32\Zlib.dll
0x47f{2	66	C:\Windows\SYSTEM32\dhcpcsvc.dll
0x480{0	34	Va\{AppData}\Local
0x480{10	60	C:\Windows\SYSTEM32\WINNSI.DLL
0x480{450	58	Microsoft\Windows\{NetCookies}
0x480{460	13	brb_3duts_by
0x480{470	70	Security=Impersonation Dynamic True
0x480{50	64	c:\Windows\SYSTEM32\netutils.dll

Save... Copy Close

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

File Machine View Input Devices Help

pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)

file settings about

File indicators (sections > executable)

- 0% footprints (type > sha256)
- 0% virtual (status > offline)
- 0% dos-headers (size > 64 bytes)
- 0% dos-stubs (size > 168 bytes)
- 0% rich-header (tooling > Visual Studio 2010)
- 0% file-header (executable > 64-bit)
- 0% optional-header (subsystem > GUI)
- 0% directories (count > 3)
- 0% sections (File > unknown)
- 0% libraries (group > network)
- 0% imports (Flag > 115)
- 0% exports (n/a)
- 0% thread-local-storage (n/a)
- 0% .NET (n/a)
- 0% resources (signature > unknown)
- 0% strings (Flag > 76)
- 0% debug (n/a)
- 0% manifest (n/a)
- 0% version (n/a)
- 0% certificate (n/a)
- 0% overlay (n/a)

property value

file	sha256: FCED85D1DCF5386257C1512991EF673FFD74C7B009E0C01E47C1D983C88234AD8
file > first-bytes-hex	4D 5A 90 00 03 00 00 04 00 00 FF 00 00 B8 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00
file > first-bytes-text	M Z@.....
size	105472 bytes
entropy	5.048
file > type	executable
cpu	64-bit
subsystem	GUI
version	n/a
description	n/a
entry-point > first-bytes-hex	4B 83 EC 28 E8 F9 00 00 83 C4 28 E9 52 FE FF CC CC 4B 89 4C 24 08 48 81 EC 88 00 00 48
entry-point > location	0x00003F4 (section[NP0])
signature tooling	Visual Studio 2010

stamps

compiler-stamp	Wed Feb 25 06:12:18 2015 (UTC)
debug-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a

names

file	c:\users\r4\desktop\packed-brbbot-no-aslr_dump.64.scy.exe
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET > module	n/a
certificate > program-name	n/a

sha256: FCED85D1DCF5386257C1512991EF673FFD74C7B009E0C01E47C1D983C88234AD8

cpu: 64-bit file: type: executable subsystem: GUI entry-point: 0x00003F4

File Machine View Input Devices Help

pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)

file settings about

File indicators (sections > executable)

- 0% footprints (type > sha256)
- 0% virtual (status > offline)
- 0% dos-headers (size > 64 bytes)
- 0% dos-stubs (size > 168 bytes)
- 0% rich-header (tooling > Visual Studio 2010)
- 0% file-header (executable > 64-bit)
- 0% optional-header (subsystem > GUI)
- 0% directories (count > 3)
- 0% sections (File > unknown)
- 0% libraries (group > network)
- 0% imports (Flag > 115)
- 0% exports (n/a)
- 0% thread-local-storage (n/a)
- 0% .NET (n/a)
- 0% resources (signature > unknown)
- 0% strings (Flag > 76)
- 0% debug (n/a)
- 0% manifest (n/a)
- 0% version (n/a)
- 0% certificate (n/a)
- 0% overlay (n/a)

property value

file	sha256: FCED85D1DCF5386257C1512991EF673FFD74C7B009E0C01E47C1D983C88234AD8
file > first-bytes-hex	4D 5A 90 00 03 00 00 04 00 00 FF 00 00 B8 00 00 00 00 00 40 00 00 00 00 00 00 00 00
file > first-bytes-text	M Z@.....
size	105472 bytes
entropy	5.048
file > type	executable
cpu	64-bit
subsystem	GUI
version	n/a
description	n/a
entry-point > first-bytes-hex	4B 83 EC 28 E8 F9 00 00 83 C4 28 E9 52 FE FF CC CC 4B 89 4C 24 08 48 81 EC 88 00 00 48
entry-point > location	0x00003F4 (section[NP0])
signature tooling	Visual Studio 2010

stamps

compiler-stamp	Wed Feb 25 06:12:18 2015 (UTC)
debug-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a

names

file	c:\users\r4\desktop\packed-brbbot-no-aslr_dump.64.scy.exe
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET > module	n/a
certificate > program-name	n/a

File Machine View Input Devices Help

pestudio 9.59 - Malware Initial Assessment - www.winitor.com (read-only)

file settings about

File indicators (sections > executable)

- 0% footprints (type > sha256)
- 0% virtual (status > offline)
- 0% dos-headers (size > 64 bytes)
- 0% dos-stubs (size > 168 bytes)
- 0% rich-header (tooling > Visual Studio 2010)
- 0% file-header (executable > 64-bit)
- 0% optional-header (subsystem > GUI)
- 0% directories (count > 3)
- 0% sections (File > unknown)
- 0% libraries (group > network)
- 0% imports (Flag > 115)
- 0% exports (n/a)
- 0% thread-local-storage (n/a)
- 0% .NET (n/a)
- 0% resources (signature > unknown)
- 0% strings (Flag > 76)
- 0% debug (n/a)
- 0% manifest (n/a)
- 0% version (n/a)
- 0% certificate (n/a)
- 0% overlay (n/a)

property value

file	sha256: FCED85D1DCF5386257C1512991EF673FFD74C7B009E0C01E47C1D983C88234AD8
file > first-bytes-hex	4D 5A 90 00 03 00 00 04 00 00 FF 00 00 B8 00 00 00 00 00 40 00 00 00 00 00 00 00
file > first-bytes-text	M Z@.....
size	105472 bytes
entropy	5.048
file > type	executable
cpu	64-bit
subsystem	GUI
version	n/a
description	n/a
entry-point > first-bytes-hex	4B 83 EC 28 E8 F9 00 00 83 C4 28 E9 52 FE FF CC CC 4B 89 4C 24 08 48 81 EC 88 00 00 48
entry-point > location	0x00003F4 (section[NP0])
signature tooling	Visual Studio 2010

stamps

compiler-stamp	Wed Feb 25 06:12:18 2015 (UTC)
debug-stamp	n/a
resource-stamp	n/a
import-stamp	n/a
export-stamp	n/a

names

file	c:\users\r4\desktop\packed-brbbot-no-aslr_dump.64.scy.exe
debug	n/a
export	n/a
version	n/a
manifest	n/a
.NET > module	n/a
certificate > program-name	n/a

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.

