

Emotet Analysis Part 1: Unpacking

 pl-v.github.io/plv/posts/Emotet-unpacking/

Player-V

April 2, 2022

Player-V on Apr 2 2022-04-02T15:26:00+08:00

Updated Apr 10 2022-04-10T22:41:15+08:00 3 min read



Introduction

That's will be my first post in the blog, i will make a series of posts about [Emotet](#).

[Emotet](#) is a Trojan that is primarily spread through spam emails (malspam), we're going to digg deep in the anlysis of this Trojan, the first part is about unpacking the malware then we will try to analyse the different modules and techniques used by the malware to compromise a machine, so fire up your virtual machine and let's start.

Triage

The first thing i always do before opening a sample in [IDA](#) or [Xdbg](#) is opening the binary first in a hex editor, in my case i will use [CFF Explorer](#), so opening the sample in CFF explorer shows that we're dealing with 32 bit binary.

Member	Offset	Size	Value	Meaning
Machine	0000010C	Word	014C	Intel 386
NumberOfSections	0000010E	Word	0004	
TimeDateStamp	00000110	Dword	6194732D	
PointerToSymbolT...	00000114	Dword	00000000	
NumberOfSymbols	00000118	Dword	00000000	
SizeOfOptionalHea...	0000011C	Word	00E0	
Characteristics	0000011E	Word	2102	Click here

Let's check import section, the malware use only one library which is `kernel32` that's the first sign which indicate that we're dealing with packed binary.

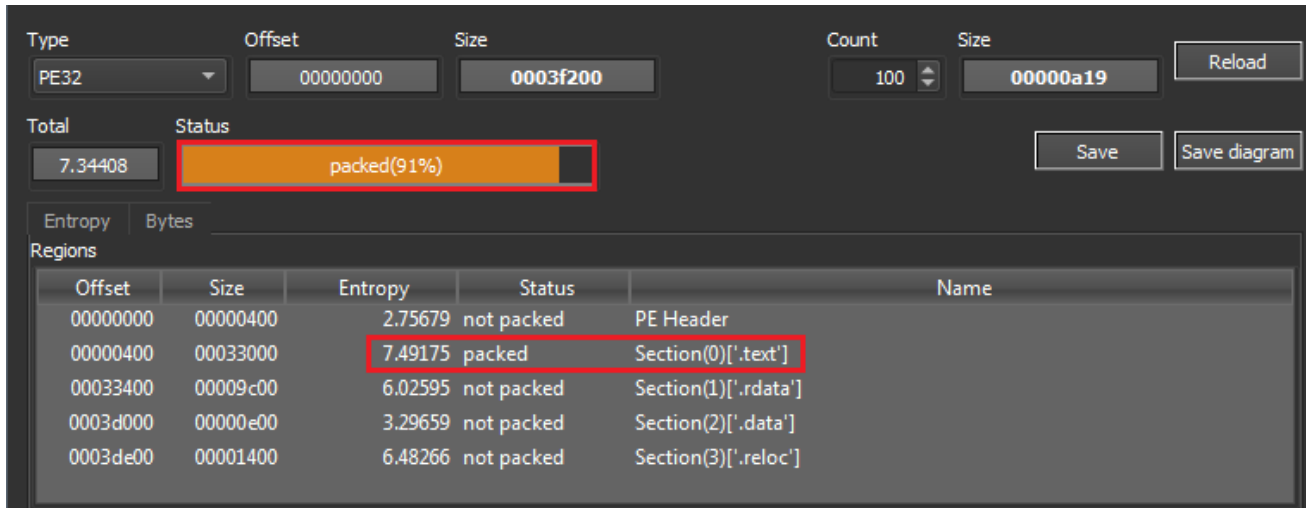
Module Name	Imports	OFTs	TimeDateStamp	ForwarderCh
0003CB7E	N/A	0003C8CC	0003C8D0	0003C8D4
szAnsi	(nFunctions)	Dword	Dword	Dword
KERNEL32.dll	69	0003D4F4	00000000	00000000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
0003D60C	0003D60C	05E0	VirtualAlloc
0003D61C	0003D61C	05E6	VirtualProtect
0003D62E	0003D62E	02BD	GetProcAddress
0003D640	0003D640	03D7	LoadLibraryA
0003D650	0003D650	0461	QueryPerformanceCounter

Two interesting API functions are used:

1. VirtualAlloc
2. VirtualProtect

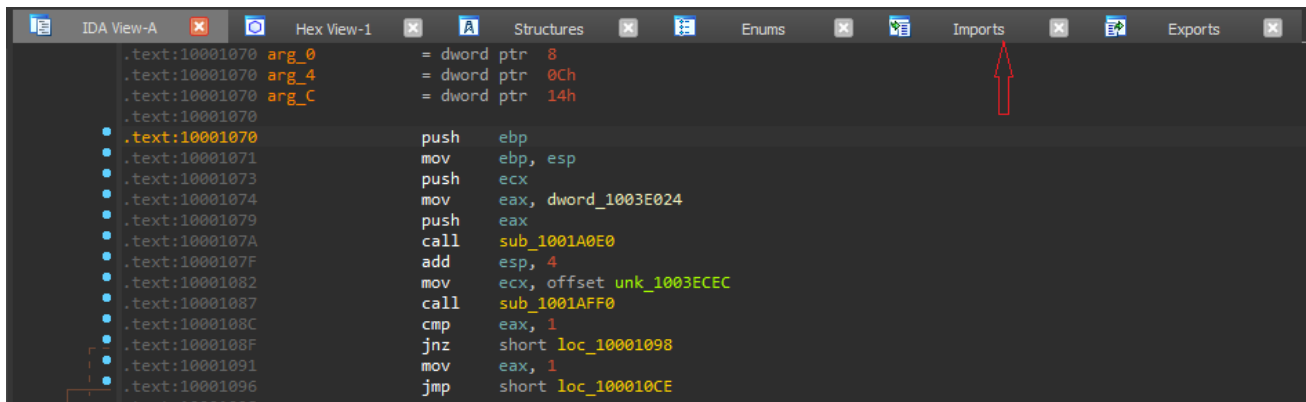
The VirtualAlloc function allocate memory while the VirtualProtect function changes the protection on a region of committed pages in the virtual address space, most of time those two functions are used by malware during the unpacking process. To make sure that our sample is packed Let's open the binary on Die(Detect-It-Easy).



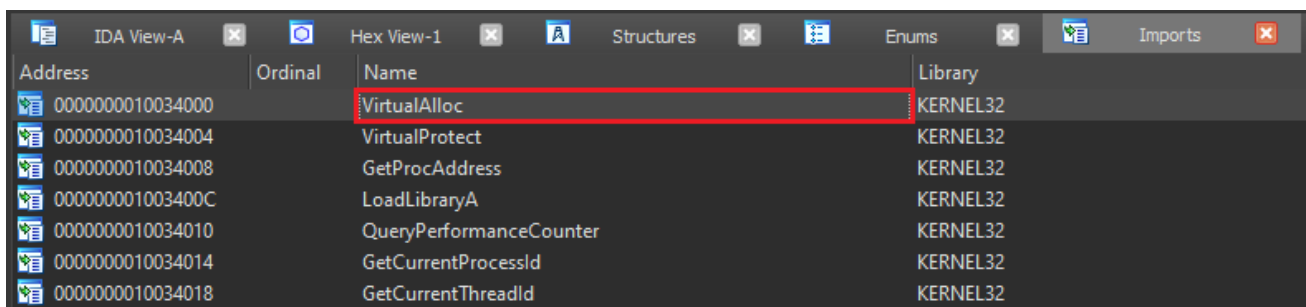
The status bar says that it's 91% packed and `.text` section has a high entropy, that's a strong indication that the malware is packed and we should unpack it for further analysis.

IDA

Now that we're sure that our sample is packed, let's open it in `IDA` and try to find the function which is responsible for unpacking.



Click on `Imports` to reveal all the functions used by the binary.



Search for `VirtualAlloc` and double click on it.

```

.idata:10034000
.idata:10034000 ; Segment type: Externs
.idata:10034000 ; .idata
.idata:10034000 ; LPVOID (__stdcall *VirtualAlloc)(LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect)
.idata:10034000 ; extrn VirtualAlloc:dword
.idata:10034000 ; CODE XREF: sub_1001AE70+D71p
.idata:10034000 ; sub_1001AE70+FA1p
.idata:10034000 ; DATA XREF: ...

```

VirtualAlloc function is used two times by the same function `sub_1001AFF0`, double click on `sub_1001AFF0` and scroll down we notice that the first function called after VirtualAlloc is `sub_10022C40`, so maybe we've found our unpacking function. to make sure let's open it on `Xdbg` and figure out.

Unpacking

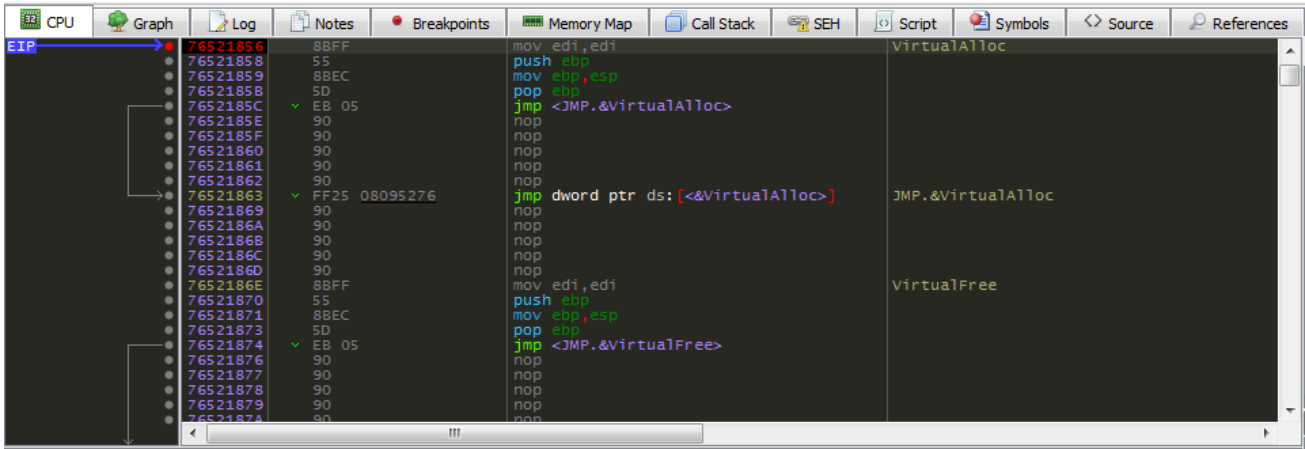
Open your `X32dbg` and paste and paste your sample to it.

The screenshot shows the X32dbg interface with the assembly view of the `emotet.EntryPoint` function. The assembly code includes instructions like `push ebp`, `mov ebp, esp`, `cmp dword ptr ssi, [ebp+C]`, `call emotet.705314E0`, `push dword ptr ssi, [ebp+10]`, `push dword ptr ssi, [ebp+C]`, `push dword ptr ssi, [ebp+8]`, `call emotet.70531219`, `add esp, C`, `pop ebp`, and `ret C`. The registers window shows `EAX: 00000000`, `EBX: 00000001`, `ECX: 0036F564`, `EDX: 00000020`, `EBP: 0036F4D8`, `ESI: 0036F4C0`, `EDI: 0036F588`, and `EIP: 7053134F <emotet.EntryPoint>`. The stack window shows `1: [esp+4] 70510000 emotet.70510000`, `2: [esp+8] 00000001`, `3: [esp+C] 00000000`, `4: [esp+10] 00000001`, and `5: [esp+14] 0036F588`. The memory dump window shows the ASCII representation of the stack data.

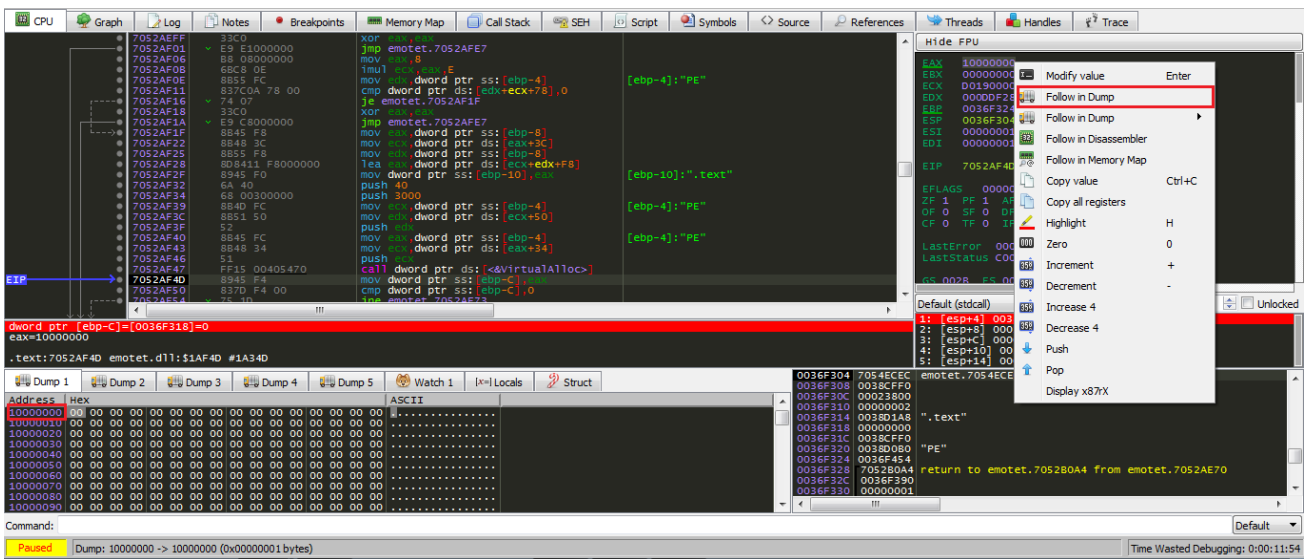
Place a breakpoint on VirtualAlloc and hit run.

The screenshot shows the X32dbg interface with a breakpoint set on the `VirtualAlloc` function. The memory dump window shows the ASCII representation of the stack data, including the string `..A..S..$.oc...`. The command window shows `bp VirtualAlloc` and the status bar indicates `Paused` and `INT3 breakpoint "entry breakpoint" at <emotet.EntryPoint> (7053134F)`.

`Xdbg` will keep running until it hit the breakpoint, after click two times on `Execute till run`.



Check the **EAX** register it contain the return address of the allocated memory by VirtualAlloc, right click on that value and click on **Follow in Dump** .



As we said earlier that the function after VirtualAlloc is responsible for unpacking, step over it and keep your eyes on the dump window at the bottom.

The screenshot shows a debugger window with the following components:

- Assembly View:** A list of instructions with their addresses and hex values. The instruction at address 7052AF90, `call emotet.70532C40`, is highlighted with a red box. Other instructions include `push eax`, `add esp, C`, `mov dword ptr ss:[ebp-14], 0`, `jmp emotet.7052AFAA`, `mov ecx, dword ptr ss:[ebp-14]`, `add ecx, 1`, `mov dword ptr ss:[ebp-14], ecx`, `mov edx, dword ptr ss:[ebp-4]`, `movzx eax, word ptr ds:[edx+6]`, `cmp dword ptr ss:[ebp-14], eax`, `jge emotet.7052AFE4`, `mov ecx, dword ptr ss:[ebp-10]`, `mov edx, dword ptr ds:[ecx+10]`, `push edx`, `mov eax, dword ptr ss:[ebp-10]`, `mov ecx, dword ptr ds:[eax+14]`, `add ecx, dword ptr ss:[ebp-8]`, `push ecx`, `mov edx, dword ptr ss:[ebp-10]`, `mov eax, dword ptr ds:[edx+C]`, `add eax, dword ptr ss:[ebp-C]`, `push eax`, `call emotet.70532C40`, `add esp, C`, and `mov dword ptr ss:[ebp-10], 0`.
- Memory Dump:** A table showing memory addresses, hex values, and ASCII characters. The first three rows are highlighted with a red box:

Address	Hex	ASCII
10000000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

After executing `sub_10022C40` function we can finally see our unpacked malware, dump it and save it somewhere in your machine.

CPU Graph Log Notes Breakpoints Memory Map Call

EIP → 7052AFEA C2 0400 ret 4
 7052AFED CC int3
 7052AFEE CC int3
 7052AFEF CC int3
 7052AFF0 55 push ebp
 7052AFF1 8BEC mov ebp, esp
 7052AFF3 6A FF push FFFFFFFF
 7052AFF5 68 823D5470 push emotet.70543D82

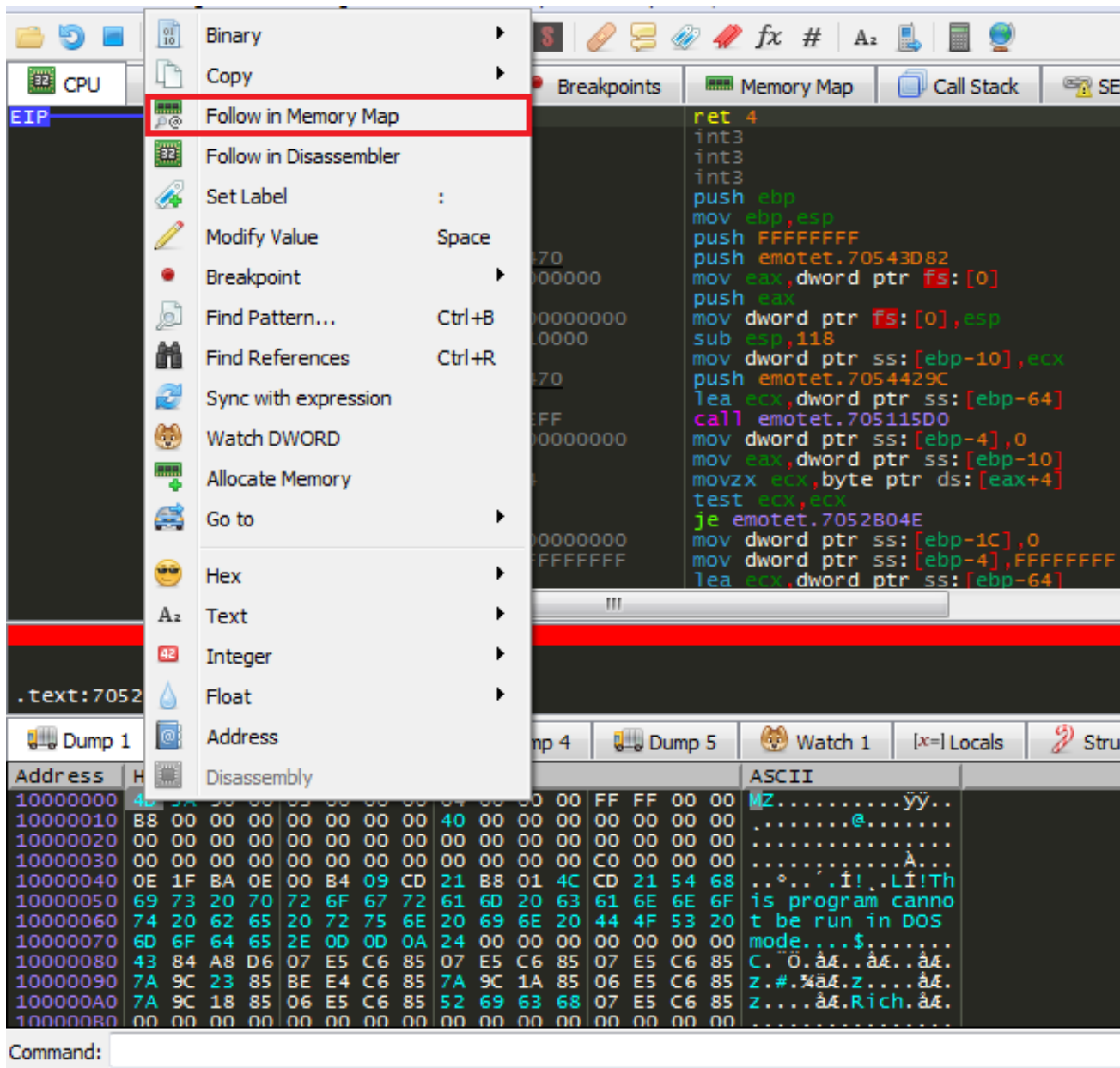
.text:7052AFEA emotet.dll:\$1AFEA #1A3EA

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Lc

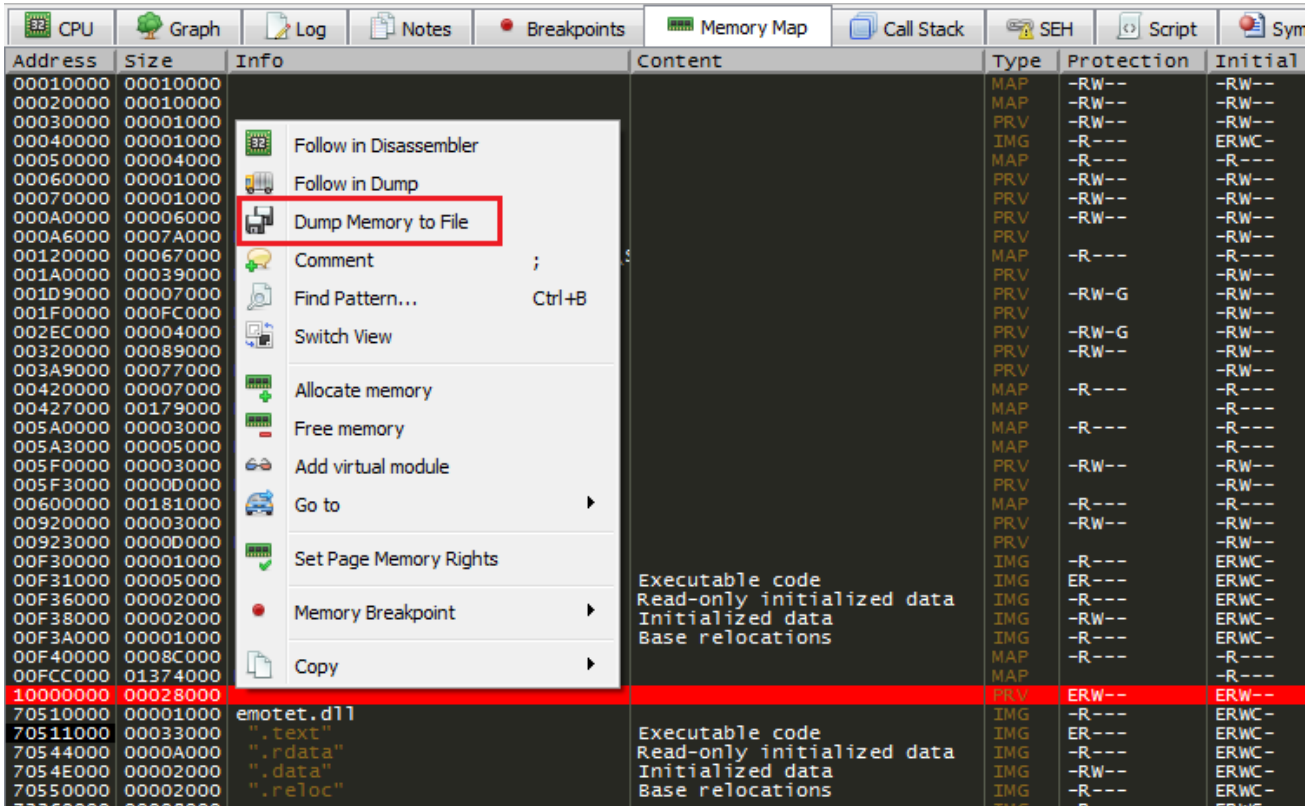
Address	Hex	ASCII
10000000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿÿ..
10000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
10000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00A...
10000030	00 00 00 00 00 00 00 00 00 00 00 00 C0 00 00 00
10000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°...i!..Li!Th
10000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program cannot
10000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	be run in DOS
10000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode...\$.
10000080	43 84 A8 D6 07 E5 C6 85 07 E5 C6 85 07 E5 C6 85	C. Ö.ä.ä.ä.ä.
10000090	7A 9C 23 85 BE E4 C6 85 7A 9C 1A 85 06 E5 C6 85	Z.#.%ä.z...ä.
100000A0	7A 9C 18 85 06 E5 C6 85 52 69 63 68 07 E5 C6 85	Z...ä.Rich.ä.
100000B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100000C0	50 45 00 00 4C 01 04 00 9A 76 91 61 00 00 00 00	PE..L....v.a...
100000D0	00 00 00 00 E0 00 02 21 0B 01 0C 00 00 2A 02 00	...à..!.....*
100000E0	00 1A 00 00 00 00 00 00 20 1A 02 00 00 10 00 00
100000F0	00 40 02 00 00 00 00 10 00 10 00 00 00 02 00 00@.....
10000100	06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00
10000110	00 80 02 00 00 04 00 00 00 00 00 00 02 00 40 01@.....
10000120	00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
10000130	00 00 00 00 10 00 00 00 10 40 02 00 47 00 00 00@.G...
10000140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000150	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000160	00 70 02 00 6C 02 00 00 00 00 00 00 00 00 00 00	..p..l.....
10000170	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000180	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10000190	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100001A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
100001B0	00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00text..

Command:

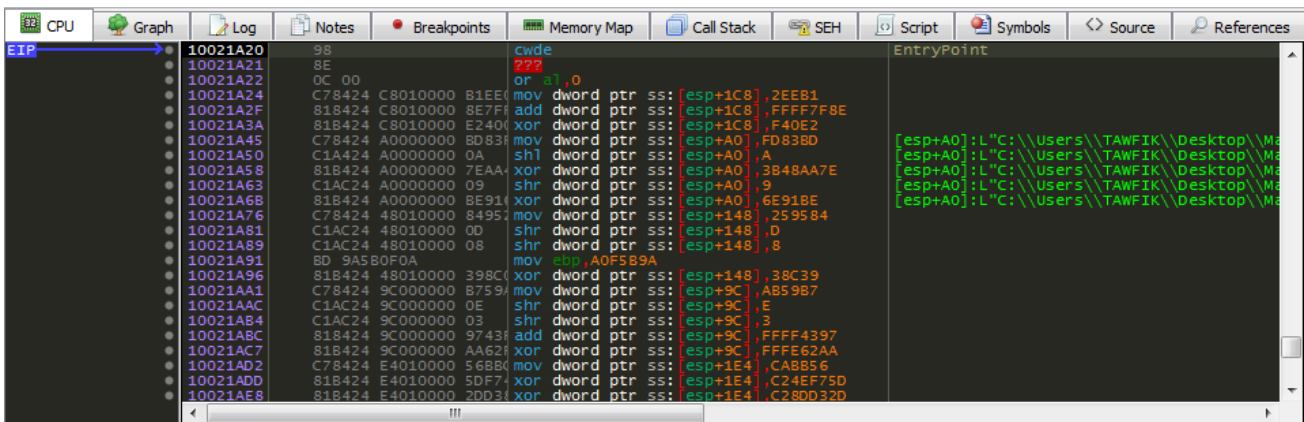
Right click on dump windows and `Follow in memory map`.



Another right click on the address of the unpacked binary then `Dump memory to file` .



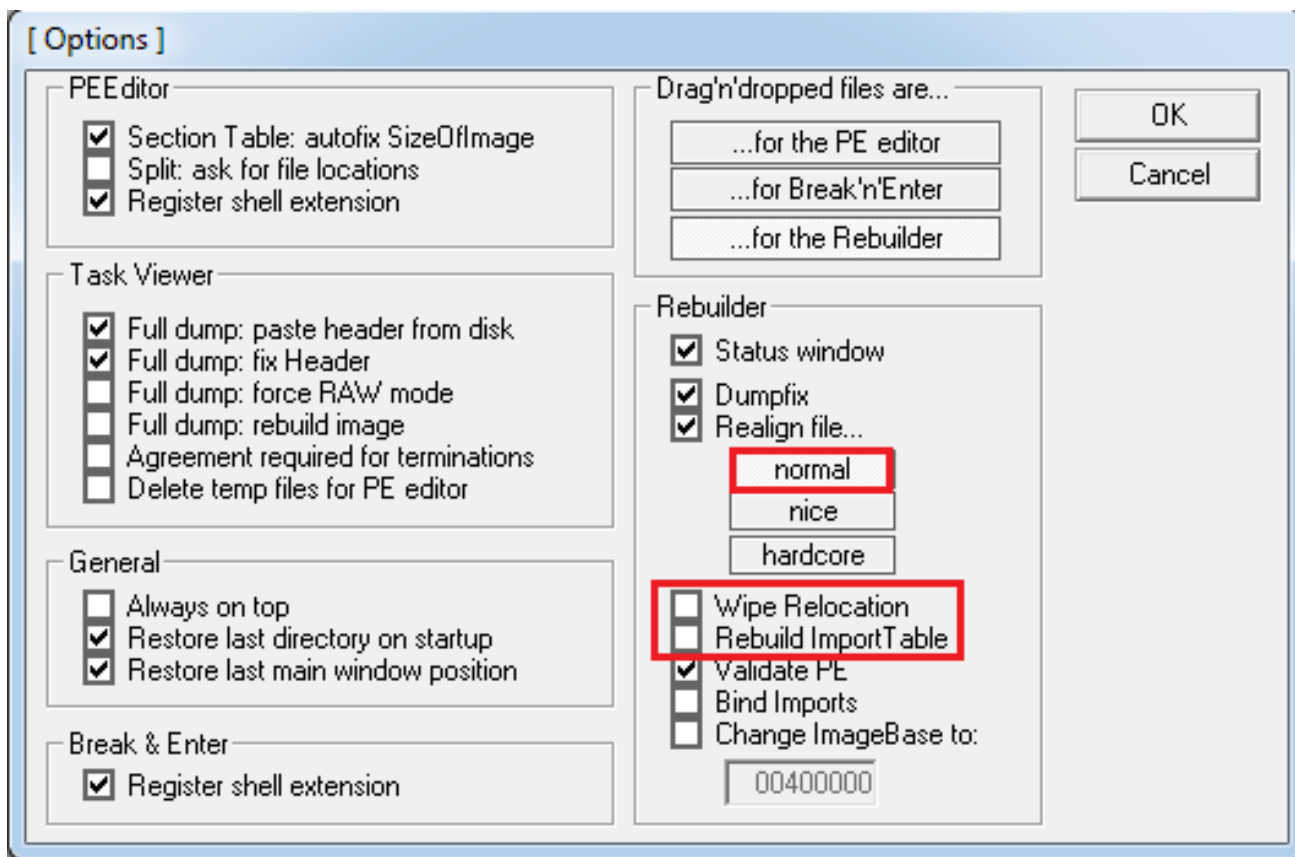
Now that we have our sample unpacked and ready for analysis let's open it in `X32dbg` .



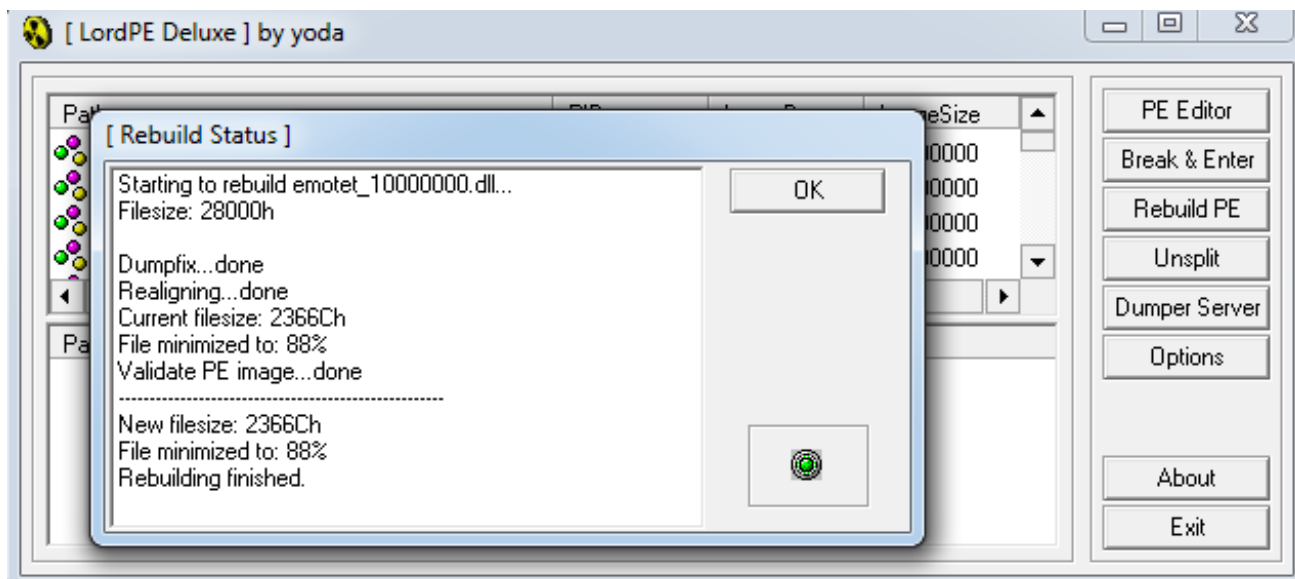
It seems that our unpacked binary is missed and it should be fixed.

Fixing

To fix the unpacked binary there are several methods to do that, we will use `LordPE` to automate the fixing, so all we should do is to open LordPe and click on options, then uncheck `Wipe Relocation` and `Rebuild ImportTable` options, finally click on `normal` then `OK` .



Drag your unpacked sample to [LordPe](#) and it will be fixed automatically.



Finally open your fixed binary in [X32dbg](#) and notice that it's more readable right now.

Address	Disassembly	Comment
70561A20	55	push ebp
70561A21	8BEC	mov ebp, esp
70561A23	83EC 14	sub esp, 14
70561A26	C745 FC 6EAB9500	mov dword ptr ss:[ebp-4], 95AB6E
70561A2D	C165 FC 04	shl dword ptr ss:[ebp-4], 4
70561A31	C16D FC 10	shr dword ptr ss:[ebp-4], 10
70561A35	8175 FC 9E0EAB3C	xor dword ptr ss:[ebp-4], 3CA80E9E
70561A3C	8175 FC 9865A13C	xor dword ptr ss:[ebp-4], 3CA16598
70561A43	C745 F0 7E3D1D00	mov dword ptr ss:[ebp-10], 1D3D7E
70561A4A	814D F0 27FEC6F2	or dword ptr ss:[ebp-10], F2C6FE27
70561A51	8175 F0 4E28DDF2	xor dword ptr ss:[ebp-10], F2DD284E
70561A58	C745 EC EB152200	mov dword ptr ss:[ebp-14], 2215EB
70561A5F	8145 EC 5164FFFF	add dword ptr ss:[ebp-14], FFFF6451
70561A66	8175 EC 9A862700	xor dword ptr ss:[ebp-14], 27869A
70561A6D	C745 F4 95777700	mov dword ptr ss:[ebp-C], 777795
70561A74	8175 F4 E9335E0A	xor dword ptr ss:[ebp-C], A5E33E9
70561A78	814D F4 1C46D843	or dword ptr ss:[ebp-C], 43D8461C
70561A82	8175 F4 93C7F24B	xor dword ptr ss:[ebp-C], 48F2C793
70561A89	C745 F8 A5BC4200	mov dword ptr ss:[ebp-8], 42BCA5
70561A90	C16D F8 08	shr dword ptr ss:[ebp-8], 8
70561A94	8175 F8 A8845F6B	xor dword ptr ss:[ebp-8], 685F84A8
70561A98	8175 F8 C95D5E6B	xor dword ptr ss:[ebp-8], 685E5DC9
70561AA2	FF4D 0C	dec dword ptr ss:[ebp+C]
70561AA5	75 29	jne emotet_10000000.70561AD0

Reference

1. [New Emotet 11/2021 - Reverse Engineering_VBA Obfuscation + Unpacking](#)
2. [How to Unpack Malware with x64dbg](#)