# Unpacking Emotet malware part 02

muha2xmad.github.io/unpacking/emotet-part-2/

January 7, 2022

### Muhammad Hasan Ali

Malware Analysis learner

5 minute read

**As-salamu Alaykum**

## Part 01 summary

Download the sample: Here

found `VirtualAlloc` call in `sub_417D50` and its address.

we search for abnormal jumps. we found `jmp ecx` and its address.

## Introduction

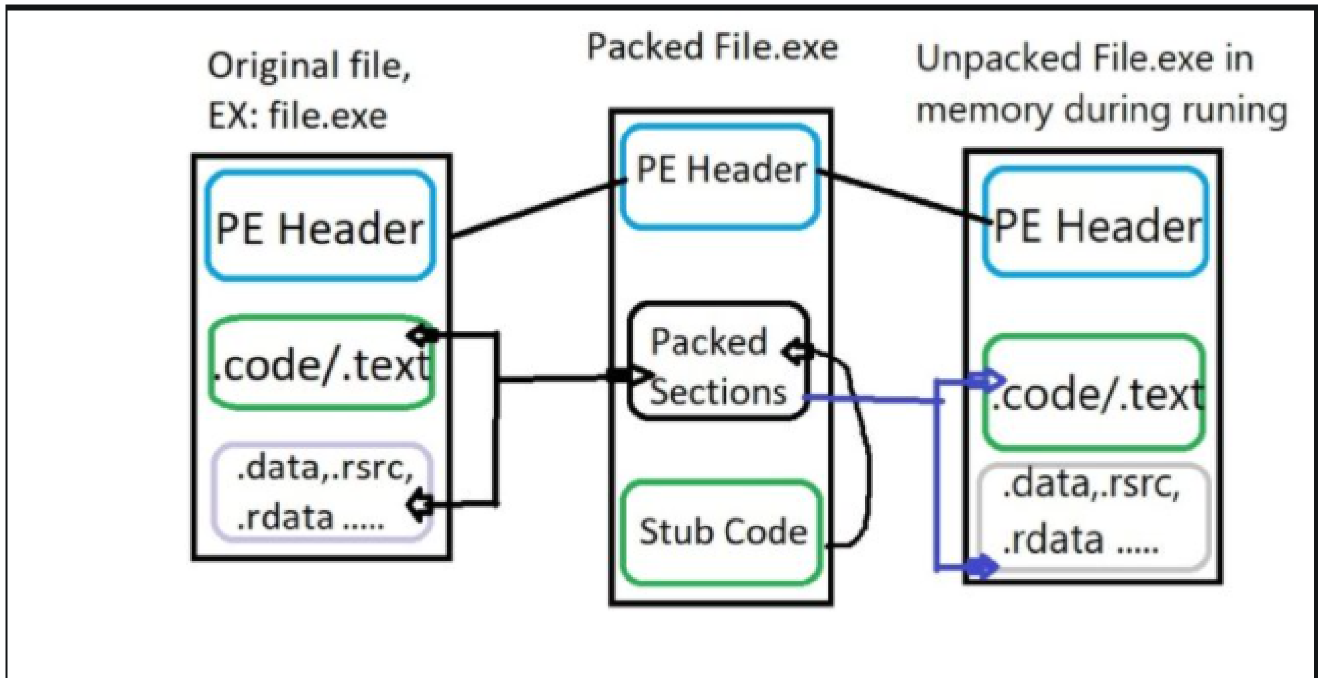
we will debug our sample with x32dbg tool to unpack the Emotet malware

## Notes to be taken

- What is Packing: A trick which is Used to avoid AV detection and analysis.
- What is a packer: A tool that compresses, encrypts, and/or modifies a malicious file format. 1
- Why using packers: To avoid AV detection and analysis to make it harder for researchers to analyze the code
- We need to find the the original entry point (OEP). What is the OEP: is the address of the malware's first instruction (where malicious code begins) before it was packed. 2
- How to find the OEP: find the `tail jump`. the tail jump It's an unconditional jump exists in the tail of stub code , it points to address of unpack file. 3

How to the unpack happen? 3 As we see in the figure (1). OS create stub code with packed file

> What is stub code? 3 Stub code is responsible for unpacking packed sections, when you are running the file ,the address of unpack file exists in the stub code to unpack file. So at the end of the stub code we will see an unconditional jump (tail jump), that is meant after execute the stub code will jump to the address of unpacking file.




Figure(1):

What is stack string? answer

We need to know what is `VirtualAlloc` ? Here

Says that “Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.”

**Syntax**

```
 LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

The most important parameter of this function is `lpaddress` , **which returns the starting offset of the newly allocated memory**. where we will extract the malware then dump it.

## Start Debugging

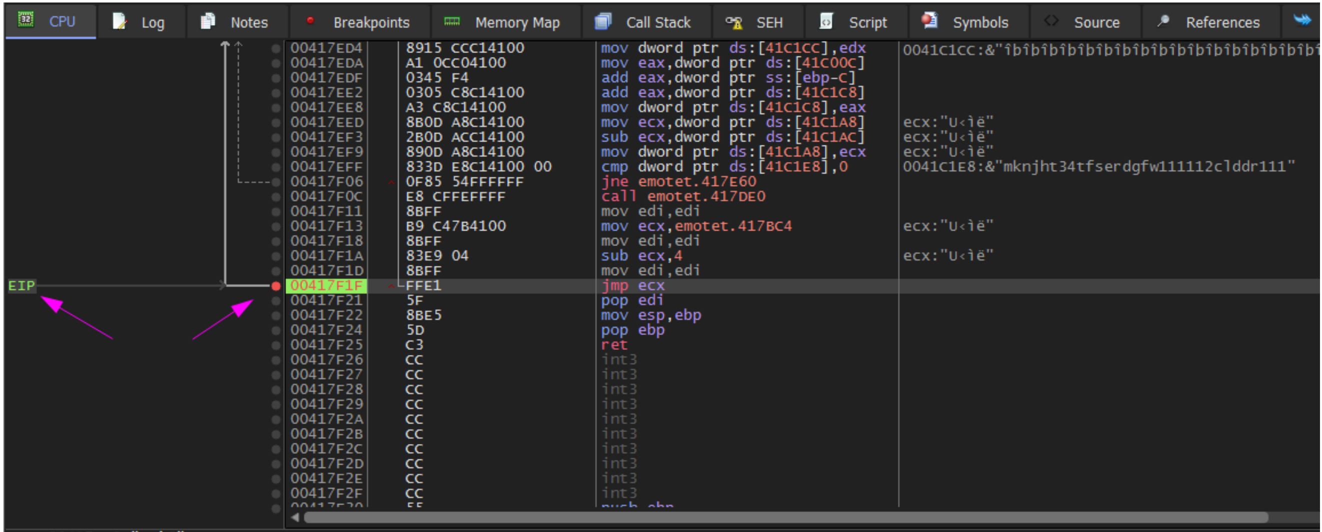
Open our sample by x32dbg and hit the entry point

Figure(2):

We set a breakpoint over the jump instruction `jmp ecx` at the address `00417F1F` by pressing `f2`. because after this jump the unpack process will happen.

Figure(3):

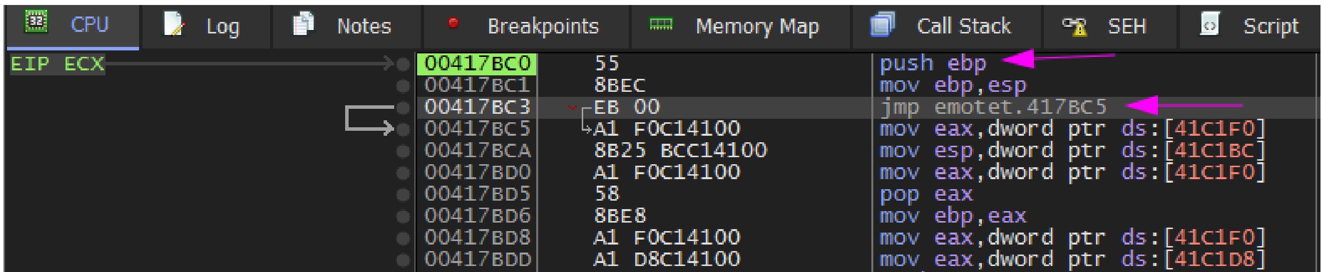
Then we press `f9` to hit the breakpoint.




Figure(4):

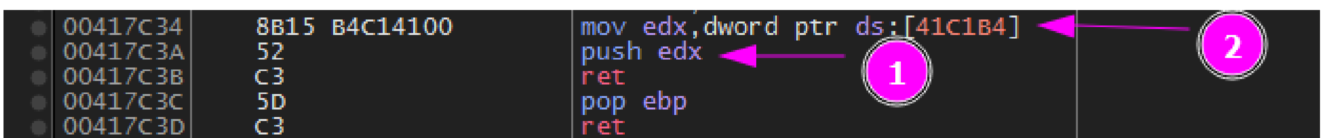
Then press `f7` we jump to another function. If we analyze this function we will notice that:

epilogue




Figure(5):

## Abnormal prologue




Figure(6):

The last figure shows the abnormal prologue (1). And (2) is a suspecious instruction which we will know late that is new `VirtualAlloc` .

So we set a breakpoint over this instruction `mov edx,dword ptr ds:[41C1B4]` by `f2` and press `f9` to hit the breakpoint.

```
EIP  00417C30  8BD2           mov edx,edx
     00417C32  8BD2           mov edx,edx
     00417C34  8B15 B4C14100  mov edx,dword ptr ds:[41C1B4]
     00417C3A  52             push edx
     00417C3B  C3             ret
     00417C3C  5D             pop ebp
     00417C3D  C3             ret
```

Figure(7):

Now if we `follow in dump` We see that it’s allocating memory.

Then Press `f8` it will `push edx` to the stack which is the value of `mov edx,dword ptr ds:[41C1B4]` .

Then Press `f8` . There is **abnormal** **`ret`** . Normal `ret` value will get back to wherever it was called from.

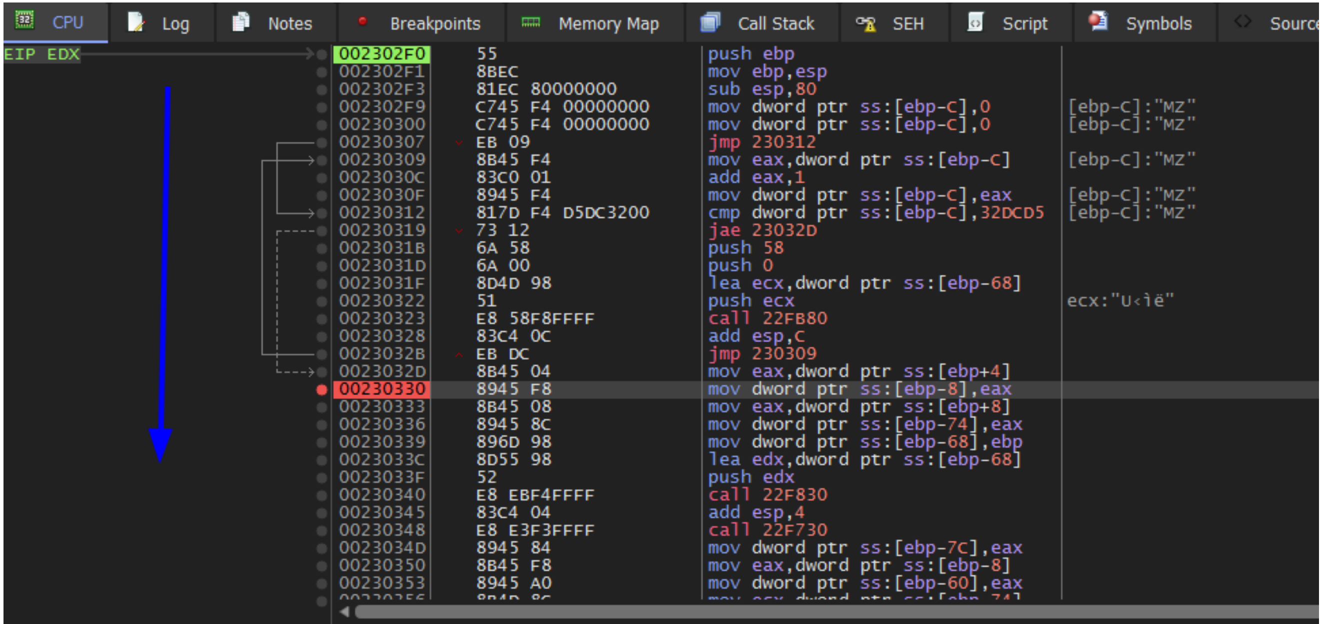
Here it return to this address `002302F0` . Which will be the address of the unpacking section.

```
0018FF80  002302F0
0018FF84  00000000
0018FF88  00400000  emotet.00400000
0018FF8C  75C2343D  return to kernel32.75C2343D from ???
0018FF90  7EFDE000
0018FF94  0018FFD4
0018FF98  778F9832  return to ntdll.778F9832 from ???
0018FF9C  7EFDE000
0018FFA0  77096289  rpcrt4.77096289
0018FFA4  00000000
0018FFA8  00000000
0018FFAC  7EFDE000
```

Figure(9):

So step over it.

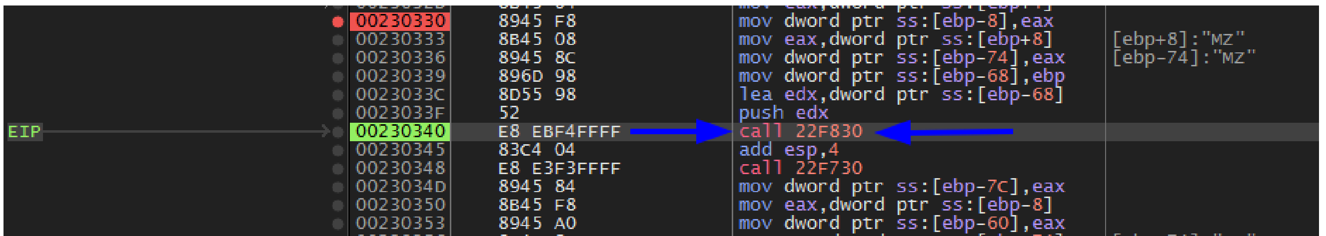
In the next part we will see functions (Unpacking routine) and we will explain it on the fly in the next figure




Figure(10):

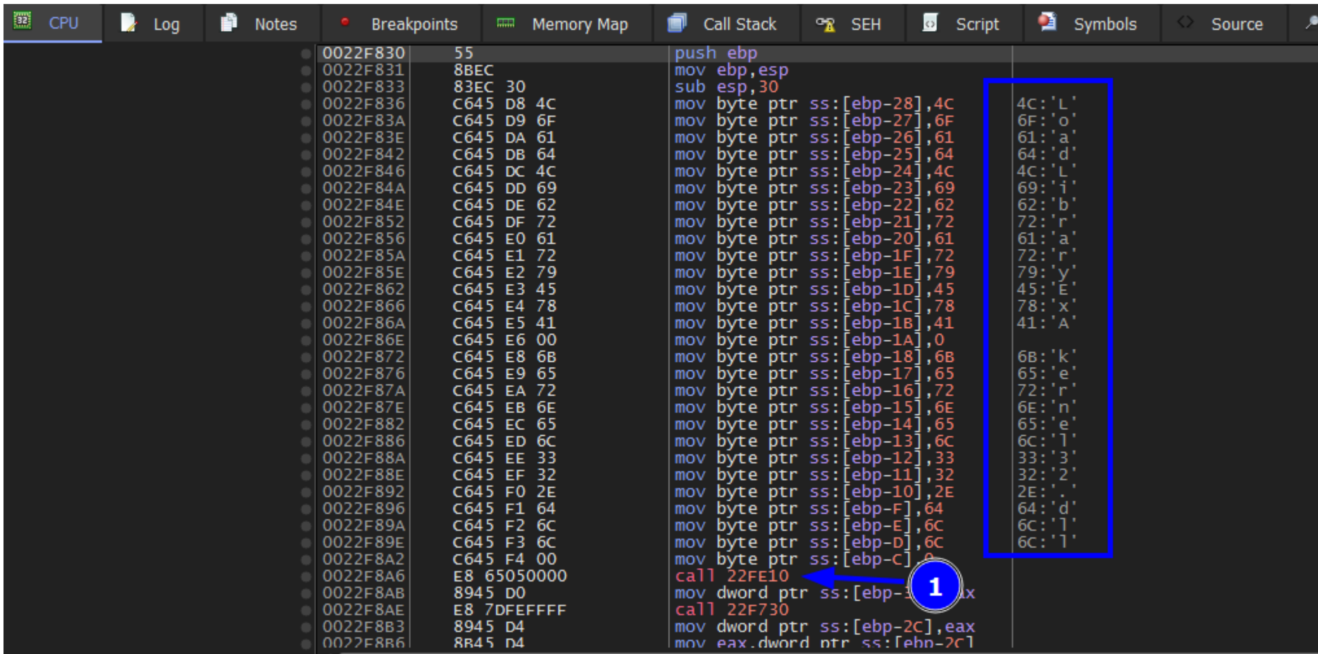
Keep stepping over untill you reach the breakpoint.

Then we see this funtion and step into `f7`.




Figure(11):

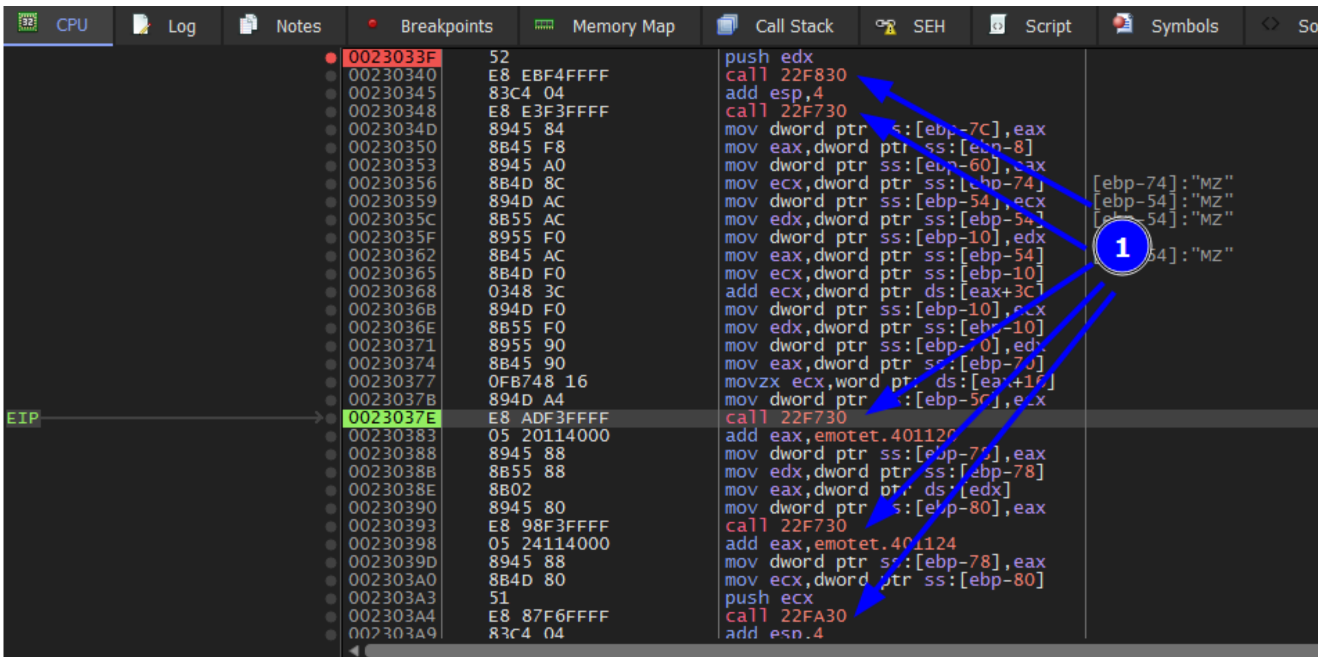
It uses stack strings. which is mentioned above In introduction. **(1) pushes them on the stack.**




Figure(12):

To get out from this function find `ret` and set a breakpoint then press `f9`

And these functions do the same as above. So step over them `f8`. to see what inside a function without executing it: Double click over a function and press `-` button to get out.




Figure(13):

Untill we get to this last function. step into `f7`.

Figure(14):

After we get into the function we need to analyze it **carefully**

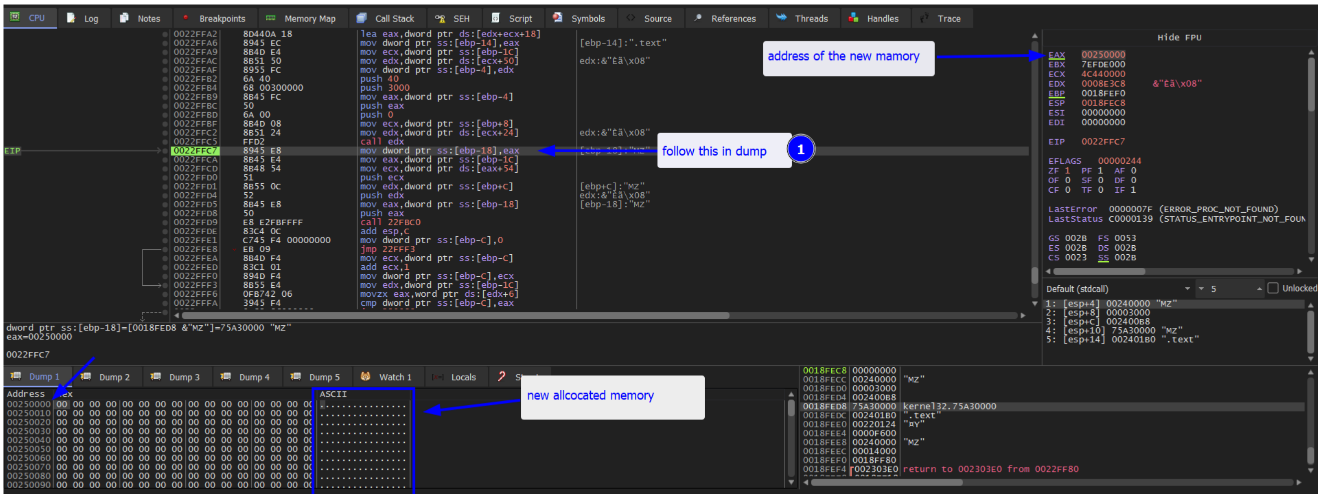
Figure(14):

As we can see `call edx` is calling `VirtualAlloc` :

`push 40` **RWX** which is our indicator to know that this call could be `VirtuallAlloc`

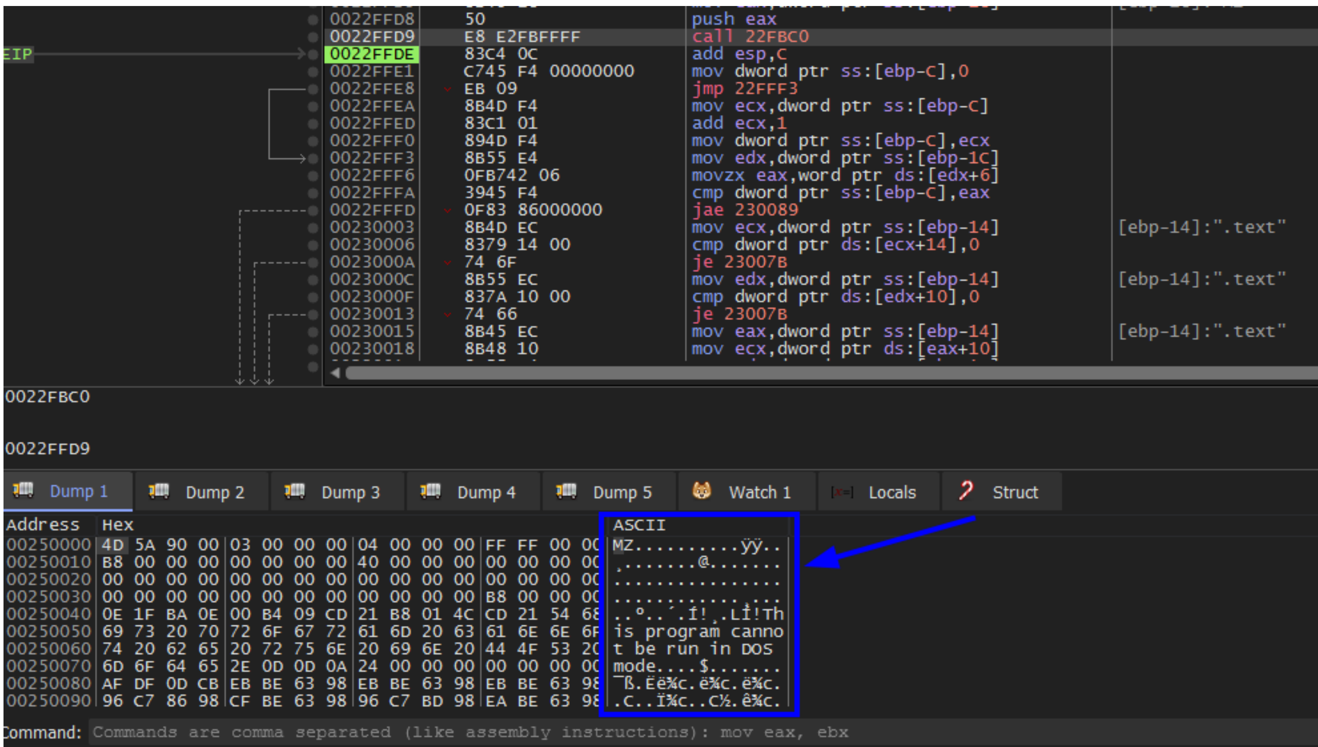
Figure(15):

One step over `f8` and we will get the adress of newly memory allocated in `eax`




Figure(16):

Then keep stepping over and get to this function `call 22FBC0` and then one more step over. As we see in the dump section, the function writes over the newly memory allocate with the exe file.




Figure(17):

When keep stepping we see that it's copying files to the exe file

`.text`

Figure(18):

then `.rdata` then `.data` then `.reloc`

Untill we get to the last `ret 8` as shown.

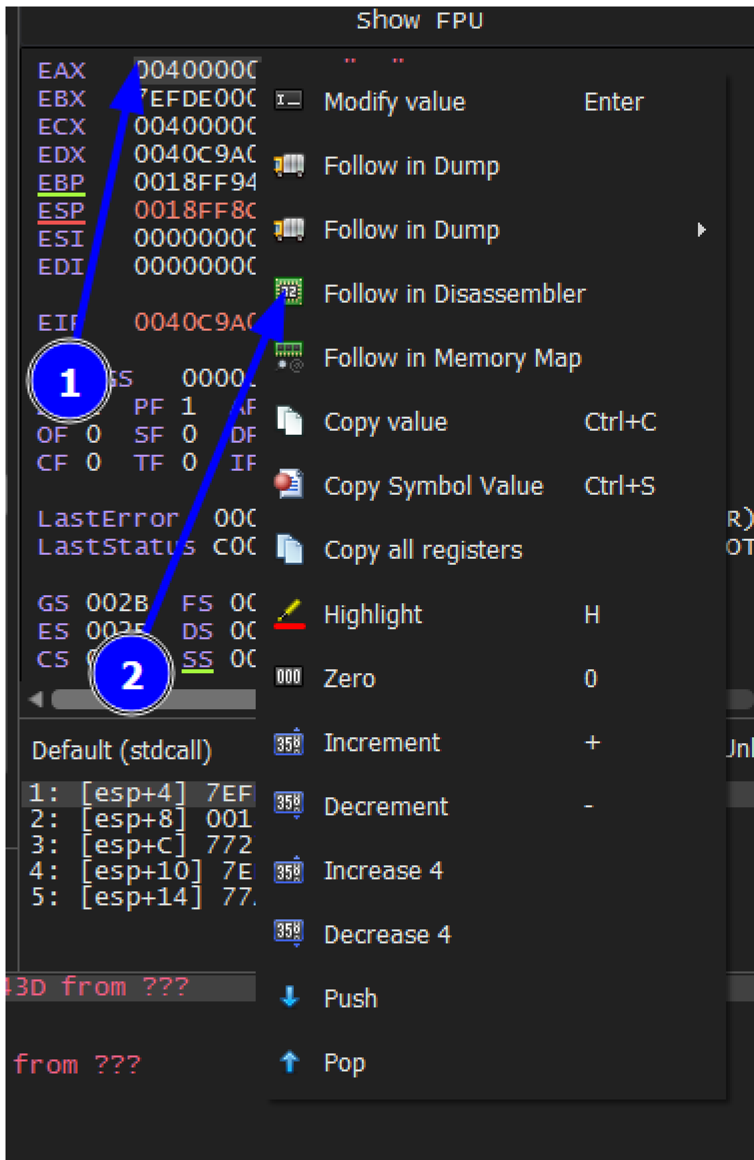
Figure(19):

**Stay awake** our file is almost finished. After the second `ret`.

Figure(20):

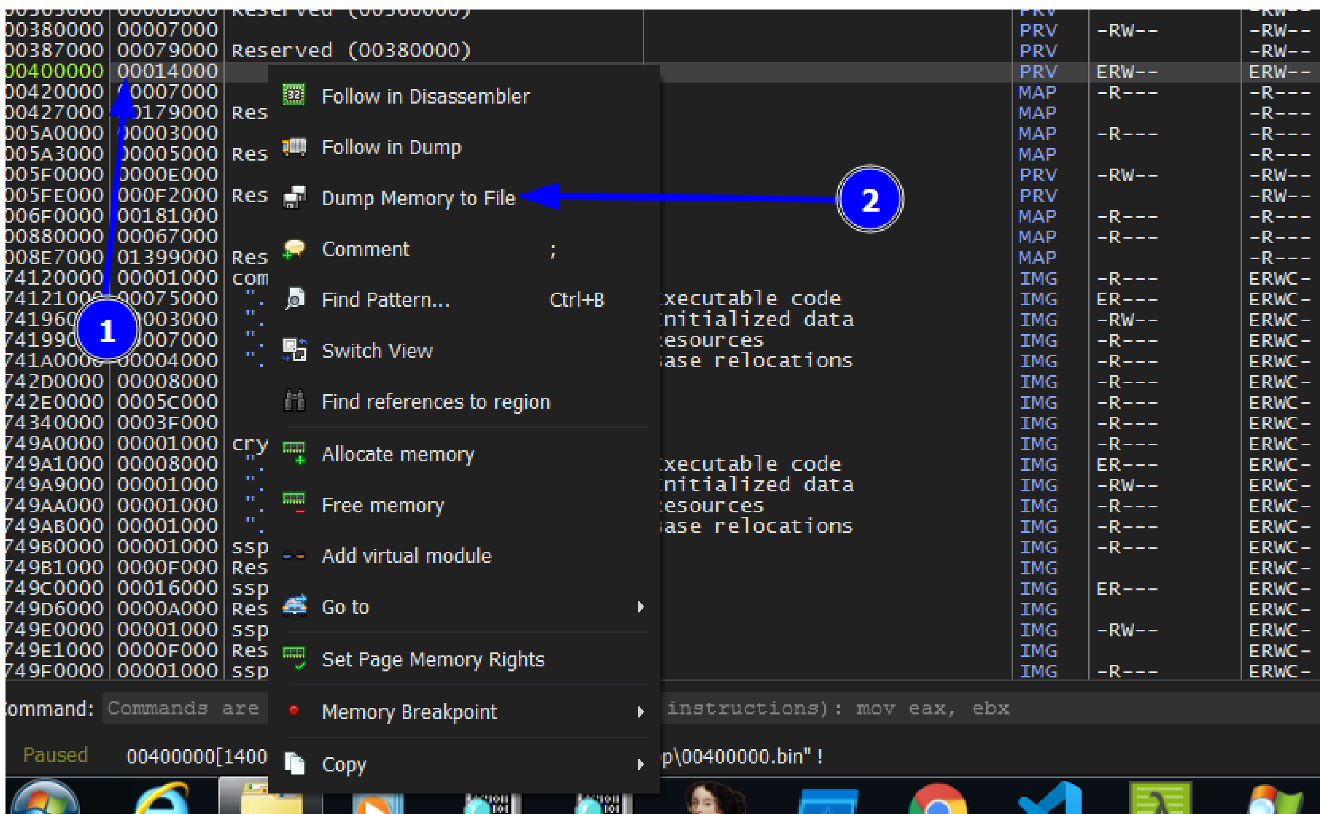
Now we can dump the unpacked exe. `right click` over `eax` and press `Follow in Mwmory map`

**Sorry for this Mistake in the next figure. It's `Follow in Mwmory map`**




Figure(21):

Then `right click` and then press `Dump memory to File`




Figure(22):

Now if we tried to open it in IDA. **We will notice that’s can’t be analyzed**

```
public start
start proc near
push    esi
lahf
popf
mov     dword ptr [ebp-4DCh], 587279C7h
mov     dword ptr [ebp-4D8h], 0ED7EE61Fh
mov     dword ptr [ebp-4D4h], 0AD05AF9Ch
mov     dword ptr [ebp-4D0h], 0B2015FBCh
mov     dword ptr [ebp-4CCh], 645D1A4Eh
mov     dword ptr [ebp-4C8h], 2E4F310Dh
mov     dword ptr [ebp-4C4h], 5A11C3CDh
mov     dword ptr [ebp-4C0h], 0A4FF6433h
mov     dword ptr [ebp-4BCh], 8BF1A0E2h
mov     dword ptr [ebp-4B8h], 0A69D0C0Fh
mov     dword ptr [ebp-4B4h], 0BC5118DCh
mov     dword ptr [ebp-4B0h], 0E7B92C0Ch
mov     dword ptr [ebp-4ACh], 0AD9CB0F4h
mov     dword ptr [ebp-4A8h], 8CD776B8h
mov     dword ptr [ebp-4A4h], 0B60EE1B9h
mov     dword ptr [ebp-4A0h], 6682D4AEh
mov     dword ptr [ebp-49Ch], 7CA422F3h
mov     dword ptr [ebp-498h], 0A37A83F0h
mov     dword ptr [ebp-494h], 647CC756h
mov     dword ptr [ebp-490h], 0E5155930h
mov     dword ptr [ebp-48Ch], 4E97325Fh
mov     dword ptr [ebp-488h], 0D2A17ECh
mov     dword ptr [ebp-484h], 0D1FFE464h
mov     dword ptr [ebp-480h], 710B8734h
mov     dword ptr [ebp-47Ch], 0F493AA9Fh
```
vith Hex View-1)

Figure(23):

So we need to repair section headers using `PE bear` tool.

**Before**

Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | BaseReloc.

| Name | Raw Addr. | Raw size | Virtual Addr. | Virtual Size | Characteristics | Ptr to Reloc. | Num. of Reloc. | Num. of Linenur |
|---|---|---|---|---|---|---|---|---|
| .text | 400 | CE00 | 1000 | CC24 | 60000020 | 0 | 0 | 0 |
| .rdata | D200 | C00 | E000 | B00 | 40000040 | 0 | 0 | 0 |
| .data | DE00 | 1200 | F000 | 3DE4 | C0000040 | 0 | 0 | 0 |
| .reloc | F000 | 600 | 13000 | 58C | 42000040 | 0 | 0 | 0 |

Figure(24):

**After editing**

Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | BaseReloc.

| Name | Raw Addr. | Raw size | Virtual Addr. | Virtual Size | Characteristics | Ptr to Reloc. | Num. of Reloc. | Num. of Linen |
|---|---|---|---|---|---|---|---|---|
| .text | 1000 | D000 | 1000 | D000 | 60000020 | 0 | 0 | 0 |
| .rdata | E000 | 1000 | E000 | 1000 | 40000040 | 0 | 0 | 0 |
| .data | F000 | 4000 | F000 | 4000 | C0000040 | 0 | 0 | 0 |
| .reloc | 13000 | 600 | 13000 | 600 | 42000040 | 0 | 0 | 0 |

Figure(25):

Then change the image base: if it’s different value of the OEP.

| Offset | Name | Value | Value |
|---|---|---|---|
| D0 | Magic | 10B | NT32 |
| D2 | Linker Ver. (Major) | C | |
| D3 | Linker Ver. (Minor) | 0 | |
| D4 | Size of Code | CE00 | |
| D8 | Size of Initialized Data | 5000 | |
| DC | Size of Uninitialized Data | 0 | |
| E0 | Entry Point | C9A0 | |
| E4 | Base of Code | 1000 | |
| E8 | Base of Data | E000 | |
| EC | Image Base | 400000 | |
| F0 | Section Alignment | 1000 | |
| F4 | File Alignment | 200 | |
| F8 | OS Ver. (Major) | 6 | Windows Vista / Server 2008 |
| FA | OS Ver. (Minor) | 0 | |
| FC | Image Ver. (Major) | 0 | |
| FE | Image Ver. (Minor) | 0 | |
| 100 | Subsystem Ver. (Major) | 6 | |
| 102 | Subsystem Ver. Minor) | 0 | |
| 104 | Win32 Version Value | 0 | |
| 108 | Size of Image | 14000 | |
| 10C | Size of Headers | 400 | |

Figure(26):

## Unmap the unpacked file

How we edit the section headers? ordered steps.

first: copy `Virtuall address` values into `Raw address` values.

second: `Raw size` Raw size of `.test` = Raw adress of `.rdata` - Raw adress of `.text`

`E000` - `1000` = `D000`

Raw size of `.rdata` = Raw adress of `.data` - Raw adress of `.rdata`

`F000` - `E000` = `1000`

Raw size of `.data` = Raw adress of `.reloc` - Raw adress of `.data`

`13000` - `F000` = `4000`

Raw size of `.reloc` = still the same third: copy `Raw size` values into `Virtual size` values.

**After changing save the file. This is our unpacked malware**

**See you in the next article. inshAllah**

## Article quote

> على الضفةِ الأخرى لن نخشى الغرق

## Refernces


1- https://www.mcafee.com/blogs/enterprise/malware-packers-use-tricks-avoid-analysis-detection/

2-https://www.oreilly.com/library/view/learning-malware-analysis/9781788392501/12556df2-7825-4e43-8811-c0fabeab78d8.xhtml

3- https://www.0xbyte.com/unpacking-mzp-ransomware-manually/

4- https://isc.sans.edu/forums/diary/Stackstrings+type+2/26192/#:~:text=This%20is%20a%20technique%20that,the%20allocated%20chunk%20of%20memory

5- https://msdn.microsoft.com/en-us/library/windows/desktop/aa366887(v=vs.85).aspx