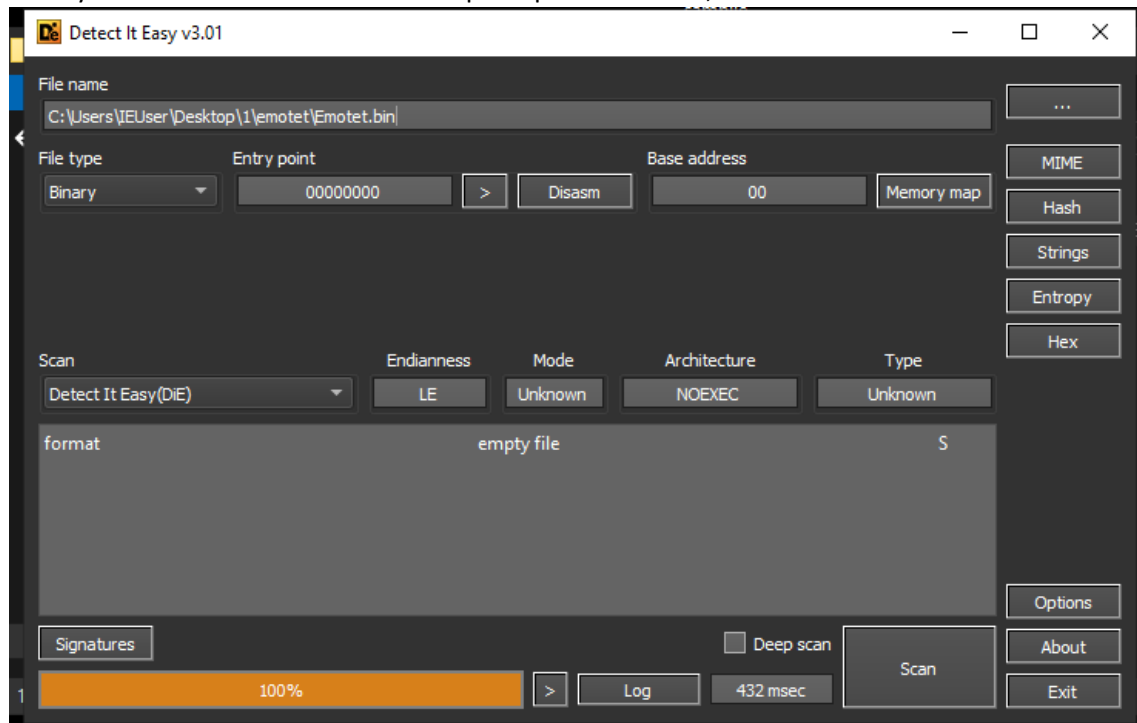
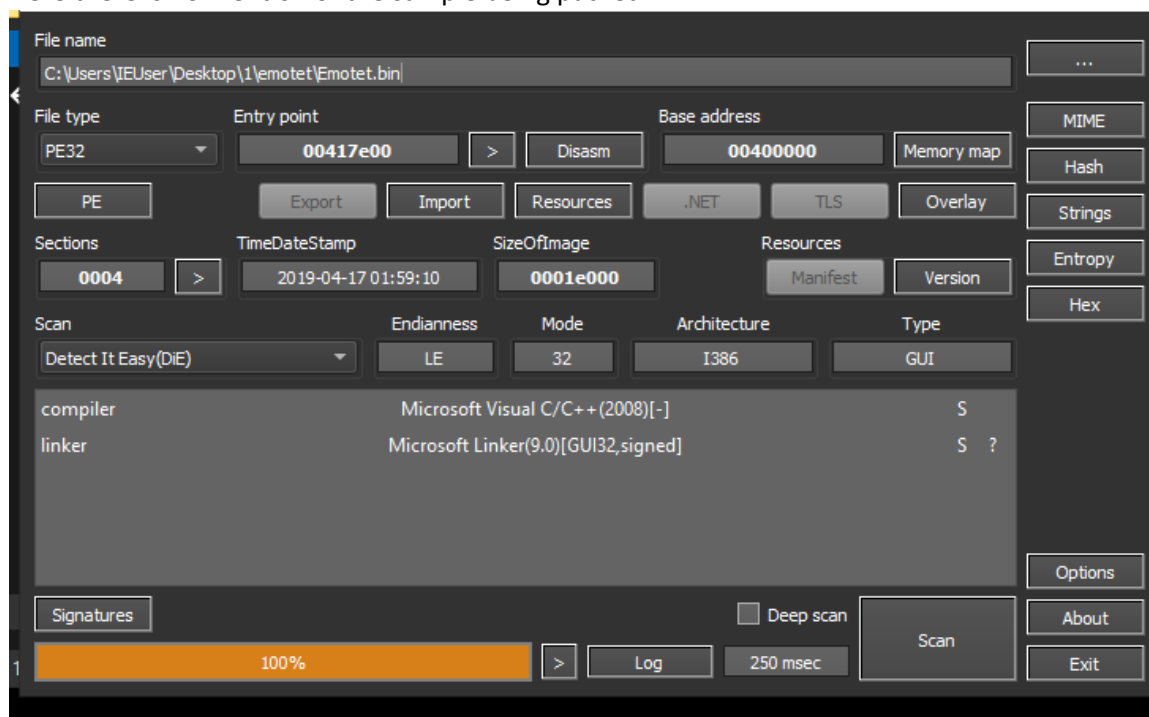


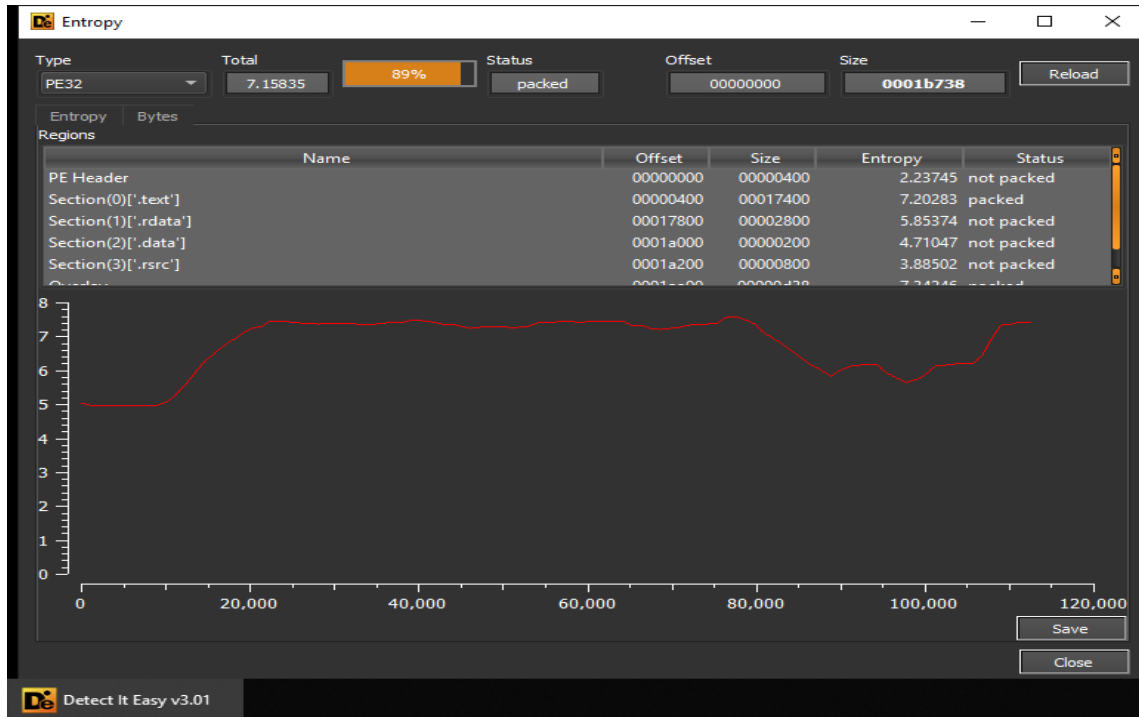
Initially we will check whether the sample is packed or not , for that we can use tool like Detect It Easy.



Here there is no mention of the sample being packed.



Now here we checked the entropy of the sample was 7.15 which is high , and also here it is showing the sample status as being packed.



Now here we load the sample in IDA . We will be looking for unpacking routines in this we noted a call for VirtualAlloc and also it was having an abnormal epilogue (normally it should have pop ebp before ret)but here it has push ecx.

```

var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 14h
mov     [ebp+var_4], 40h
mov     [ebp+var_C], 0
mov     eax, dword_41C1A4
mov     [ebp+var_14], eax
mov     [ebp+var_8], 0FFFFFFFFh
mov     ecx, ds:VirtualAlloc
mov     dword_41C218, ecx
push    [ebp+var_4]
push    3000h
push    [ebp+var_14]
push    [ebp+var_C]
mov     ecx, dword_41C218
push    offset loc_417D9A
push    ecx
retn

add     ecx, 102F0h
mov     dword_41C184, ecx
mov     eax, [ebp+var_10]
mov     esp, ebp
pop     ebp
retn
sub_417D50 endp ; sp-analysis

```

Normally ret execute the code present in top of the stack, so here it is calling VirtualAlloc ,now when VirtualAlloc returns it will return to the loc_41709A and inside it is the final return.

```

; Attributes: bp-based frame
sub_417D50 proc near
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
push ebp
mov ebp, esp
sub esp, 14h
mov [ebp+var_4], 40h
mov [ebp+var_C], 0
mov eax, dword_41C1A4
mov [ebp+var_14], eax
mov [ebp+var_8], 0FFFFFFFh
mov ecx, ds:vi7uall1loc
mov dword_41C218, ecx
push [ebp+var_4]
push 3000h
push [ebp+var_14]
push [ebp+var_C]
mov ecx, dword_41C218
loc_417D9A:
mov [ebp+var_10], eax
mov edx, [ebp+var_10]
mov dword_41C1E8, edx
mov eax, dword_41C1A4
mov dword_41C1A8, eax
mov dword_41C1B4, 0
mov ecx, dword_41C1E8
add ecx, 102F0h
mov dword_41C1B4, ecx
mov eax, [ebp+var_10]
mov esp, ebp
ret

```

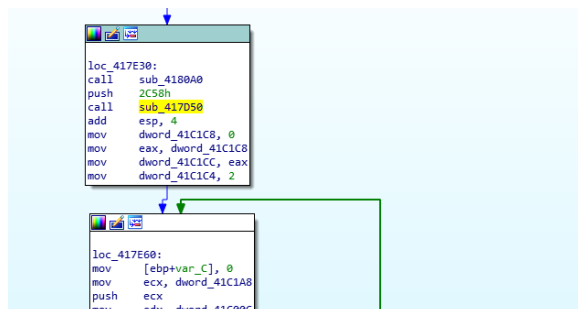
This is where the final return control go.

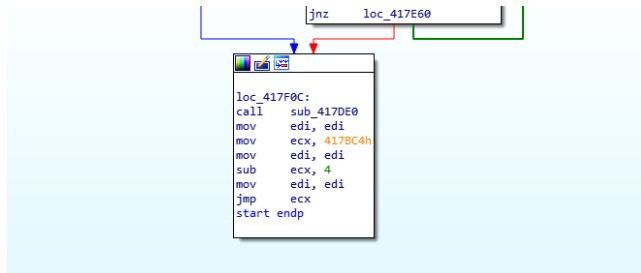
```

.text:00417E30 loc_417E30:
.text:00417E30 call sub_4180A0
.text:00417E35 push 2C58h
.text:00417E3A call sub_417D50
.text:00417E3F add esp, 4
.text:00417E42 mov dword_41C1C8, 0
.text:00417E4C mov eax, dword_41C1C8
.text:00417E51 mov dword_41C1CC, eax
.text:00417E56 mov dword_41C1C4, 2
.text:00417E60 loc_417E60:
.text:00417E60 mov [ebp+var_C], 0
.text:00417E67 mov ecx, dword_41C1A8
.text:00417E6D push ecx
.text:00417E74 mov edx, dword_41C00C
.text:00417E75 push edx
.text:00417E77 call sub_417F30
.text:00417E7A add esp, 8
.text:00417E7D mov dword_41C200, eax
.text:00417E82 mov eax, dword_41C1C8
.text:00417E87 cmp eax, dword_41C1A4
.text:00417E8D jnb short loc_417E91
.text:00417E8F jmp short loc_417F0C

```

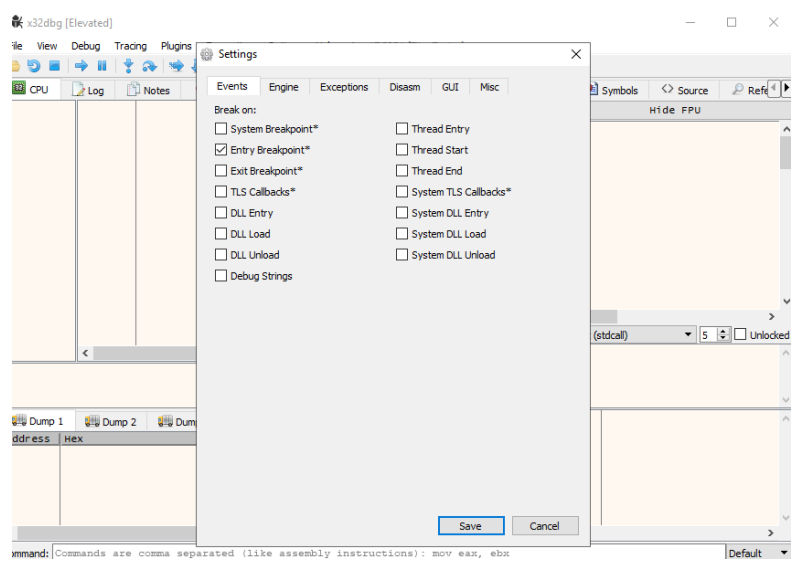
Here we can put breakpoint and this can be unpacking code location and once the code is unpacked we will be looking for that , like any abnormal jump.

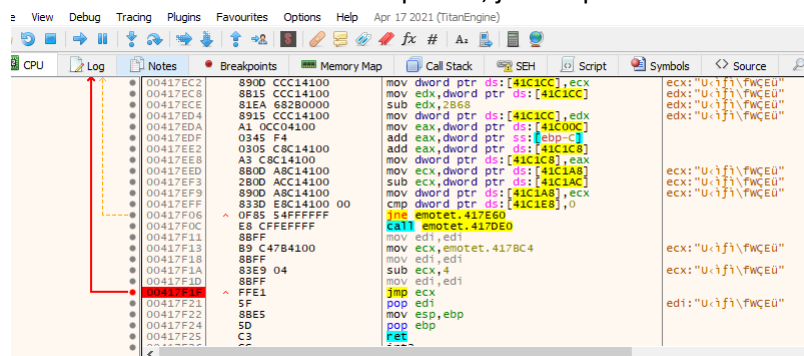




Here we find an abnormal jump.

Now we will load the sample in x32dbg. Here in Options->Preferences->Uncheck the System Breakpoint and TLS Callbacks.





Now here we have taken the jump to ecx. This is the location where the part of unpacked code resides.

```
004302F0 55      push ebp
004302F1 8BEC    mov ebp,esp
004302F3 81EC    sub esp,80
004302F9 C745 F4 00000000 mov dword ptr ss:[ebp-C],0
00430300 C745 F4 00000000 mov dword ptr ss:[ebp-C],0
00430307 EB 09    jmp 430312
00430309 8B45 F4  mov eax,dword ptr ss:[ebp-C]
0043030C 83C0 01  add eax,1
0043030F 8945 F4  mov dword ptr ss:[ebp-C],eax
00430312 817D F4 D5DC3200 cmp dword ptr ss:[ebp-C],320CD5
00430319 73 12    jae 430320
0043031B 6A 58    push 58
0043031D 6A 00    push 0
0043031F 8D40 98  lea ecx,dword ptr ss:[ebp-68]
00430322 51      push ecx
00430323 E8 5BF8FFFF call 42F880
00430328 83C4 0C  add esp,C
0043032B EB DC    jmp 430309
0043032D 8B45 04  mov eax,dword ptr ss:[ebp+4]
00430330 8945 F8  mov dword ptr ss:[ebp-8],eax
00430333 8B45 08  mov eax,dword ptr ss:[ebp+8]
00430336 8945 0C  mov dword ptr ss:[ebp-7],eax
00430339 89D0 98  mov dword ptr ss:[ebp-68],ebp
0043033C 8D55 98  lea edx,dword ptr ss:[ebp-68]
0043033F 52      push edx
00430340 EB F8FFFF call 42F880
```

Here in this unpacked code there is an abnormal epilogue i.e. push edx before ret. In edx there can be

```
00417C1A 8BD2    mov edx,edx
00417C1C 8BD2    mov edx,edx
00417C1E 8BD2    mov edx,edx
00417C20 8BD2    mov edx,edx
00417C22 8BD2    mov edx,edx
00417C24 8BD2    mov edx,edx
00417C26 8BD2    mov edx,edx
00417C28 8BD2    mov edx,edx
00417C2A 8BD2    mov edx,edx
00417C2C 8BD2    mov edx,edx
00417C2E 8BD2    mov edx,edx
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
00417C3D 5D      pop ebp
00417C3E C3      ret
00417C3F CC      int3
00417C41 CC      int3
00417C43 55      push ebp
00417C44 8BEC    mov ebp,esp
00417C46 8B45 08  mov eax,dword ptr ss:[ebp+8]
00417C48 8B45 0C  mov eax,dword ptr ss:[ebp+8]
```

another unpacked code,put a breakpoint here.

This is the another newly unpacked section.

```
004302F0 55      push ebp
004302F1 8BEC    mov ebp,esp
004302F3 81EC    sub esp,80
004302F9 C745 F4 00000000 mov dword ptr ss:[ebp-C],0
00430300 C745 F4 00000000 mov dword ptr ss:[ebp-C],0
00430307 EB 09    jmp 430312
00430309 8B45 F4  mov eax,dword ptr ss:[ebp-C]
0043030C 83C0 01  add eax,1
0043030F 8945 F4  mov dword ptr ss:[ebp-C],eax
00430312 817D F4 D5DC3200 cmp dword ptr ss:[ebp-C],320CD5
00430319 73 12    jae 430320
0043031B 6A 58    push 58
0043031D 6A 00    push 0
0043031F 8D40 98  lea ecx,dword ptr ss:[ebp-68]
00430322 51      push ecx
00430323 E8 5BF8FFFF call 42F880
00430328 83C4 0C  add esp,C
0043032B EB DC    jmp 430309
0043032D 8B45 04  mov eax,dword ptr ss:[ebp+4]
00430330 8945 F8  mov dword ptr ss:[ebp-8],eax
00430333 8B45 08  mov eax,dword ptr ss:[ebp+8]
00430336 8945 0C  mov dword ptr ss:[ebp-7],eax
00430339 89D0 98  mov dword ptr ss:[ebp-68],ebp
0043033C 8D55 98  lea edx,dword ptr ss:[ebp-68]
0043033F 52      push edx
00430340 EB F8FFFF call 42F880
```

Here we can see it is moving the strings in stack and then using them as parameters for the call .

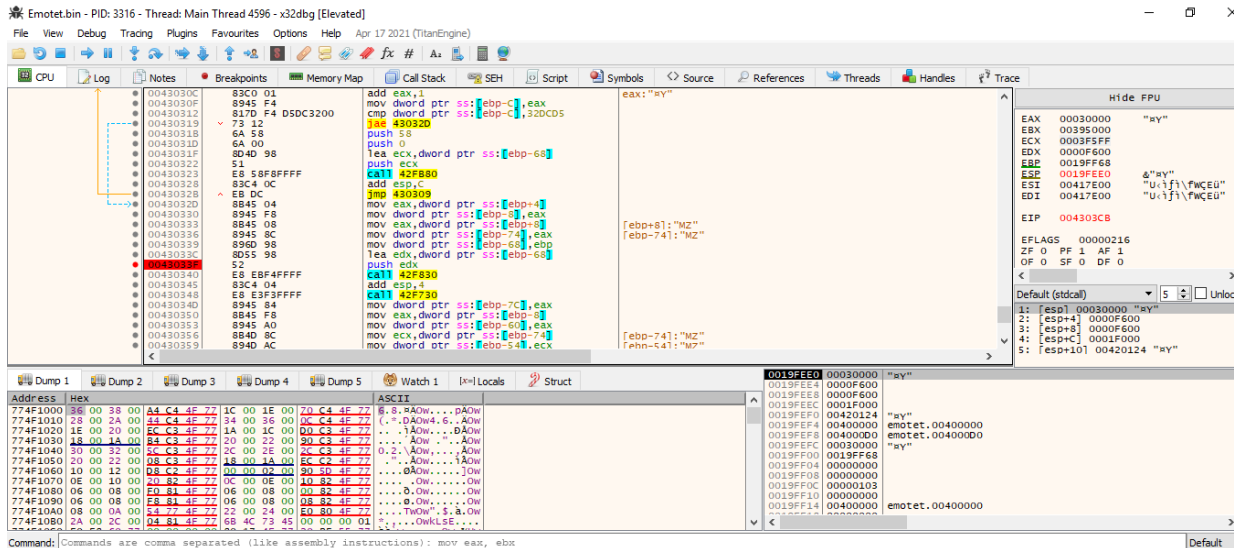
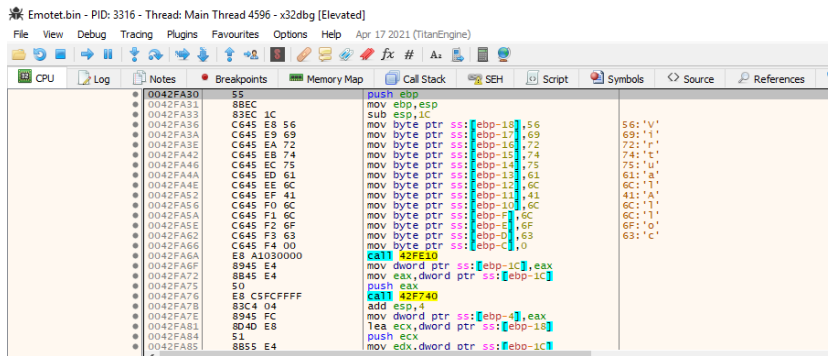
```
0042F830 55      push    ebp
0042F831 8BEC    mov     ebp,esp
0042F832 30      sub     esp,30
0042F833 8BEC    mov     ptr [ebp-28],4C
0042F834 4C      mov     byte ptr [ebp-27],6F
0042F835 61      mov     byte ptr [ebp-26],61
0042F836 64      mov     byte ptr [ebp-25],64
0042F837 4C      mov     byte ptr [ebp-24],4C
0042F838 69      mov     byte ptr [ebp-23],69
0042F839 62      mov     byte ptr [ebp-22],62
0042F83A 72      mov     byte ptr [ebp-21],72
0042F83B 61      mov     byte ptr [ebp-20],61
0042F83C 72      mov     byte ptr [ebp-1F],72
0042F83D 79      mov     byte ptr [ebp-1E],79
0042F83E 45      mov     byte ptr [ebp-1D],45
0042F83F 78      mov     byte ptr [ebp-1C],78
0042F840 41      mov     byte ptr [ebp-1B],41
0042F841 68      mov     byte ptr [ebp-1A],68
0042F842 65      mov     byte ptr [ebp-19],65
0042F843 72      mov     byte ptr [ebp-18],72
0042F844 6E      mov     byte ptr [ebp-17],6E
0042F845 65      mov     byte ptr [ebp-16],65
0042F846 6C      mov     byte ptr [ebp-15],6C
0042F847 33      mov     byte ptr [ebp-14],33
0042F848 32      mov     byte ptr [ebp-13],32
```

Here now it will return to the caller function.

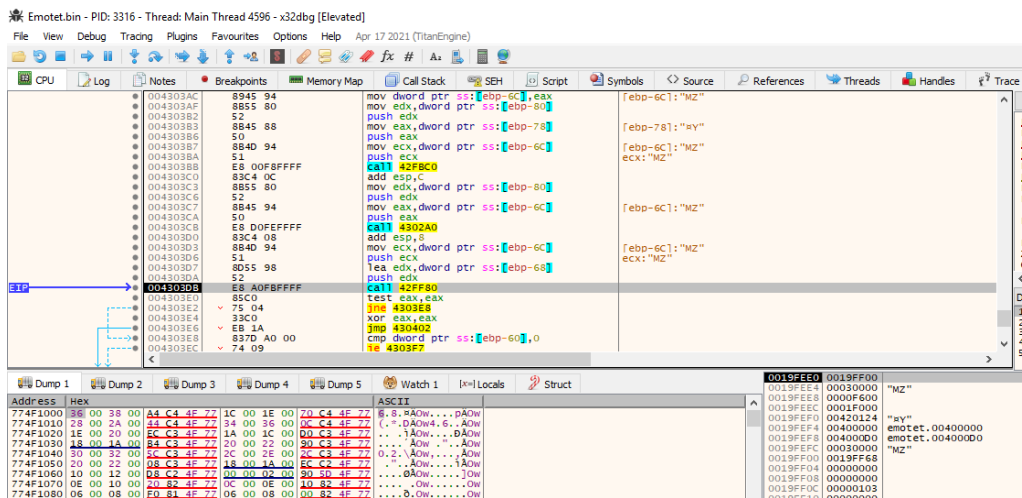
```
0042FA0F 8955    mov     dword ptr [ebp-4],edx
0042FA10 3BFC    cmp     dword ptr [ebp-4],F
0042FA11 73      jbe     42FA1D
0042FA12 8B45    mov     eax,dword ptr [ebp-4]
0042FA13 1B      imul    eax,eax,11
0042FA14 0345    add     eax,dword ptr [ebp-4]
0042FA15 50      push    eax
0042FA16 8B4D    mov     ecx,dword ptr [ebp-30]
0042FA17 51      push    ecx
0042FA18 8B55    mov     edx,dword ptr [ebp+8]
0042FA19 8B42    mov     eax,dword ptr [edx+20]
0042FA1A 50      push    eax
0042FA1B 8B4D    mov     ecx,dword ptr [ebp-4]
0042FA1C 8B55    mov     edx,dword ptr [ebp+8]
0042FA1D 66      mov     dword ptr [edx+ecx*4+1C],eax
0042FA1E 50      jmp     42F9EC
0042FA1F 50      pop     esp
0042FA20 5D      pop     ebp
0042FA21 C3      ret
0042FA22 CC      int3
0042FA23 CC      int3
0042FA24 CC      int3
0042FA25 CC      int3
0042FA26 CC      int3
0042FA27 CC      int3
```

Here the another similar function doing the same approach.

```
0042F730 58      mov     eax,00000000
0042F731 2D      sub     eax,35074100
0042F732 C3      jmp     42F735
0042F733 CC      int3
0042F734 CC      int3
0042F735 CC      int3
0042F736 CC      int3
0042F737 55      push    ebp
0042F738 8BEC    mov     ebp,esp
0042F739 30      sub     esp,30
0042F73A 47      mov     byte ptr [ebp-28],47
0042F73B 65      mov     byte ptr [ebp-27],65
0042F73C 74      mov     byte ptr [ebp-26],74
0042F73D 50      mov     byte ptr [ebp-25],50
0042F73E 72      mov     byte ptr [ebp-24],72
0042F73F 6F      mov     byte ptr [ebp-23],6F
0042F740 63      mov     byte ptr [ebp-22],63
0042F741 41      mov     byte ptr [ebp-21],41
0042F742 64      mov     byte ptr [ebp-20],64
0042F743 72      mov     byte ptr [ebp-1F],72
0042F744 6E      mov     byte ptr [ebp-1E],6E
0042F745 73      mov     byte ptr [ebp-1D],73
0042F746 73      mov     byte ptr [ebp-1C],73
```



In the below image we have reached the last function call, we will step into it



Here in this image below we can see the call edx (here it is VirtualAlloc) and for VirtualAlloc it has pushed 40 normally the 4th parameter of VirtualAlloc.


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

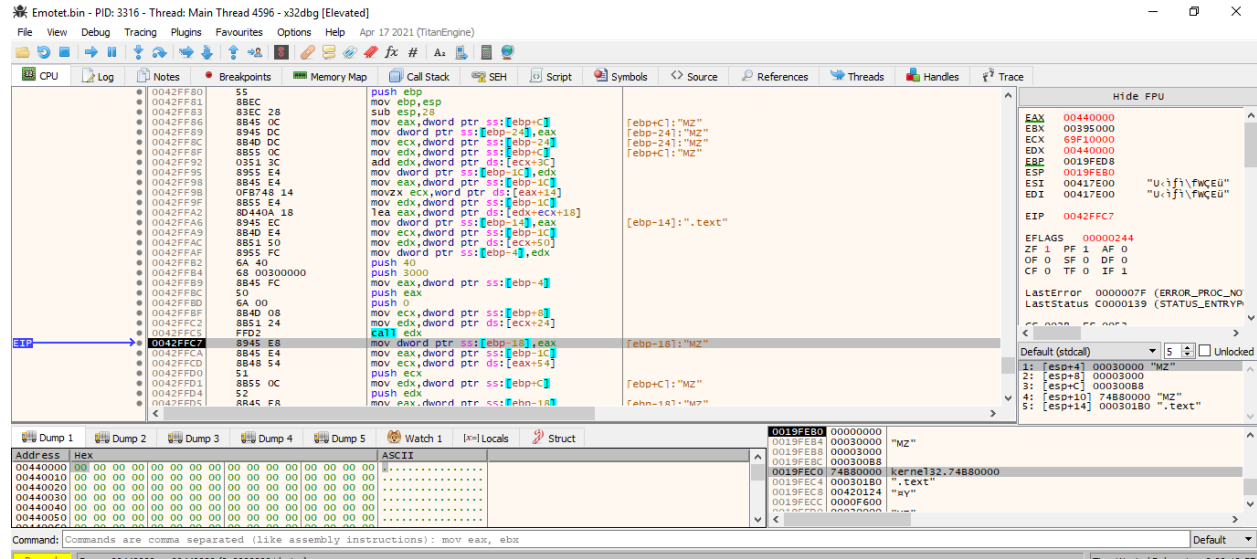
PAGE_EXECUTE_READWRITE Enables execute, read-only, or read/write access to the committed region of pages.

Windows Server 2003 and Windows XP: This attribute is not supported by the [CreateFileMapping](#) function until Windows XP with SP2 and Windows Server 2003 with SP1.

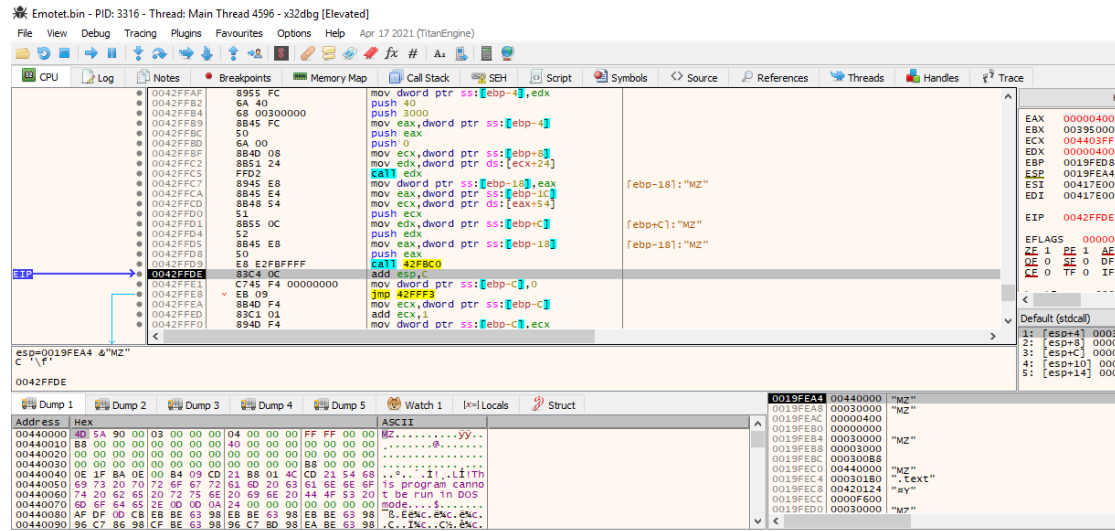
VirtualAlloc Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.

<https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc>.

In the image below after stepping over the call edx now we can see the value return base address of the new allocated memory region in the eax i.e 440000 , now we will follow the allocated region in dump.



After this call we can see that it has dumped an PE file in the allocated region.



Till now it has allocated different sections of pe in that location.

Emotet.bin - PID: 3316 - Thread: Main Thread 4596 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Apr 17 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

004301B2 8B40 08 mov ecx,dword ptr ss:[ebp+8]
004301B5 8B51 18 mov edx,dword ptr ds:[ecx+18]
004301B8 52 push edx
004301B9 8B45 08 mov eax,dword ptr ss:[ebp+8]
004301BC 8B48 10 mov ecx,dword ptr ds:[eax+10]
004301BF 51 push ecx
004301C0 8B55 08 mov edx,dword ptr ss:[ebp+8]
004301C3 8B42 14 mov eax,dword ptr ds:[edx+14]
004301C6 50 push eax
004301C7 E8 04DFFFFF call 42FED0
004301CC 83C4 10 add esp,10
004301CF 8B40 08 mov ecx,dword ptr ss:[ebp+8]
004301D2 8379 08 00 cmp dword ptr ds:[ecx+8],0
004301D6 75 0F jne 4301E8
004301D8 8B55 08 mov ecx,dword ptr ss:[ebp+8]
004301DB 8B42 18 mov eax,dword ptr ds:[ecx+18]
004301DE 50 push eax
004301DF E8 7CF9FFFF call 42F860
004301E4 83C4 04 add esp,4
004301E7 B8 01000000 mov eax,1
004301EC 8BE5 mov esp,ebp
004301ED 5D pop ebp
004301F2 CC int3

004301EF

0019FEDC 004303E0 return to 004303E0 from 0042FF80
0019FE00 0019FF00
0019FEE4 00030000
0019FEEB 0000F600
0019FEEC 0001F000
0019FEF0 00420124
0019FEF4 00400000
0019FEF8 00400000
0019FEFC 00030000
0019FF00 0019FF68
0019FF04 00000000
0019FF08 00000000

Address Hex
00440000 80 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
00440010 B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00440020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00440030 00 00 00 00 00 00 00 00 00 00 00 00 8B 00 00 00
00440040 0E 1F 8A 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68
00440050 69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F
00440060 74 20 62 65 2E 0D 00 0A 24 00 00 00 00 00 00 00
00440070 6D 6F 64 65 2E 0D 00 0A 24 00 00 00 00 00 00 00
00440080 AF DF 00 CB EB BE 63 98 EB BE 63 98 EB BE 63 98
00440090 96 C7 86 98 CF BE 63 98 96 C7 80 98 EA BE 63 98

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

now it has finished the unpacking , now we can dump that into the file.

Emotet.bin - PID: 3316 - Thread: Main Thread 4596 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Apr 17 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

004303E0 85C0 test eax,ecx
004303E2 75 04 jne 4303E8
004303E4 3BC0 xor eax,ecx
004303E6 EB 1A jmp 430402
004303E8 837D A0 00 cmp dword ptr ss:[ebp-60],0
004303EC 74 09 jle 4303E7
004303EE 8B45 80 mov eax,dword ptr ss:[ebp-50]
004303F1 8B40 98 mov ecx,dword ptr ds:[ebp-68]
004303F4 8B41 10 mov dword ptr ds:[ecx+10],eax
004303F7 8B55 A8 mov edx,dword ptr ss:[ebp-58]
004303FA 8B55 98 mov esp,dword ptr ss:[ebp-68]
004303FD 5D pop ebp
004303FE 58 pop eax
004303FF 58 pop eax
00430400 52 push edx
00430401 C3 ret
00430402 8BE5 mov esp,ebp
00430403 5D pop ebp
00430404 5D pop ebp
00430405 C3 ret
00430406 0000 add byte ptr ds:[eax],al
00430408 0000 add byte ptr ds:[eax],al
0043040A 0000 add byte ptr ds:[eax],al
0043040C 0000 add byte ptr ds:[eax],al
0043040E 0000 add byte ptr ds:[eax],al

eax=1

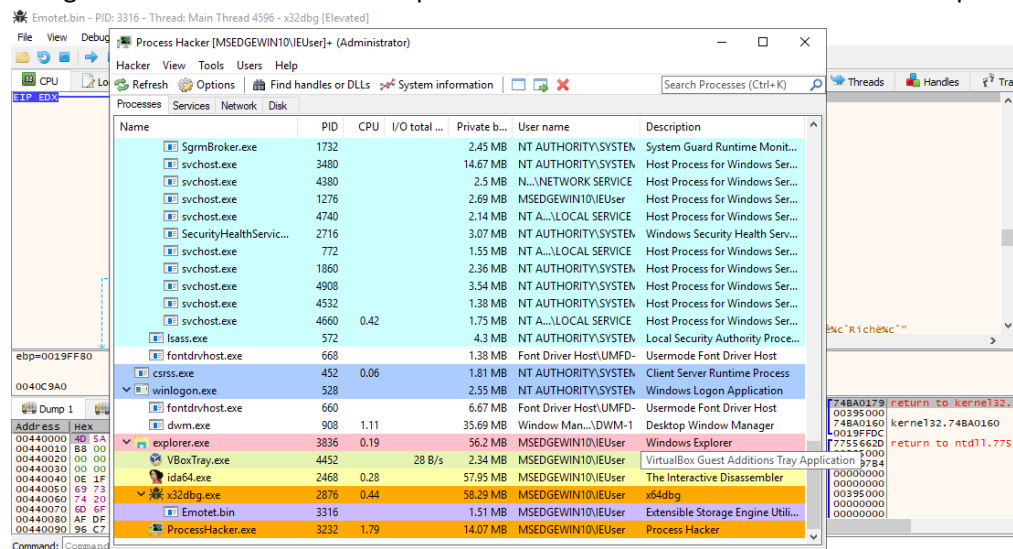
004303E0

0019FEE8 0000F600
0019FEEC 0001F000
0019FEF0 00420124
0019FEF4 00400000
0019FEF8 00400000
0019FEFC 00030000
0019FF00 0019FF68
0019FF04 00000000
0019FF08 00000000
0019FF0C 00000103
0019FF10 0040C9A0
0019FF14 00400000

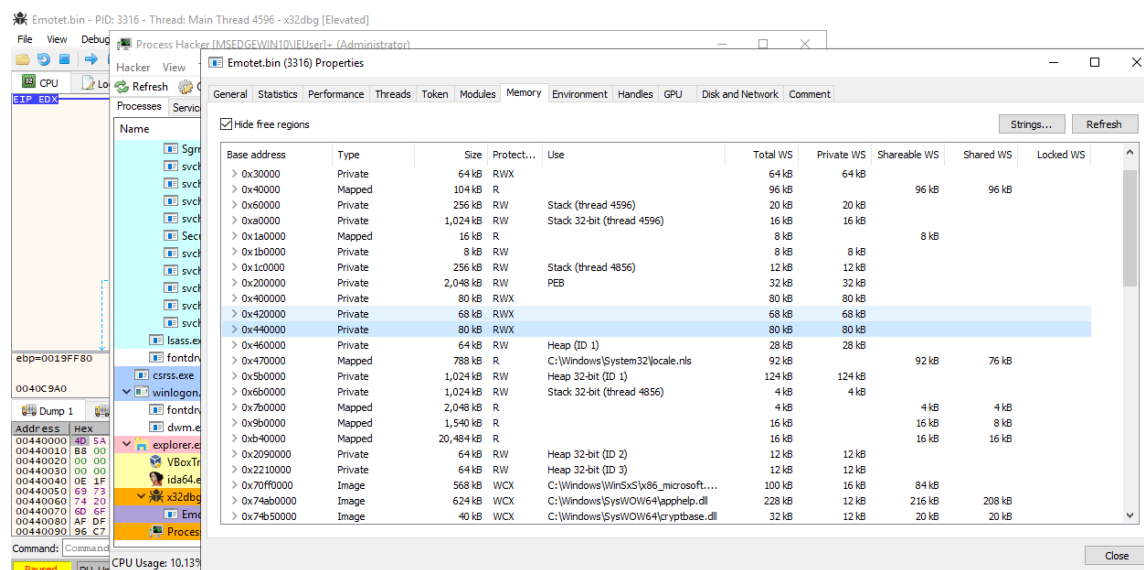
Address Hex
00440000 80 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
00440010 B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00440020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00440030 00 00 00 00 00 00 00 00 00 00 00 00 8B 00 00 00
00440040 0E 1F 8A 0E 00 84 09 CD 21 B8 01 4C CD 21 54 68
00440050 69 73 20 70 72 6F 67 72 61 60 20 63 61 6E 6E 6F
00440060 74 20 62 65 2E 0D 00 0A 24 00 00 00 00 00 00 00
00440070 6D 6F 64 65 2E 0D 00 0A 24 00 00 00 00 00 00 00
00440080 AF DF 00 CB EB BE 63 98 EB BE 63 98 EB BE 63 98
00440090 96 C7 86 98 CF BE 63 98 96 C7 80 98 EA BE 63 98

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Using Process Hacker we can dump that file. Double click on emotet and then open the memory tab.

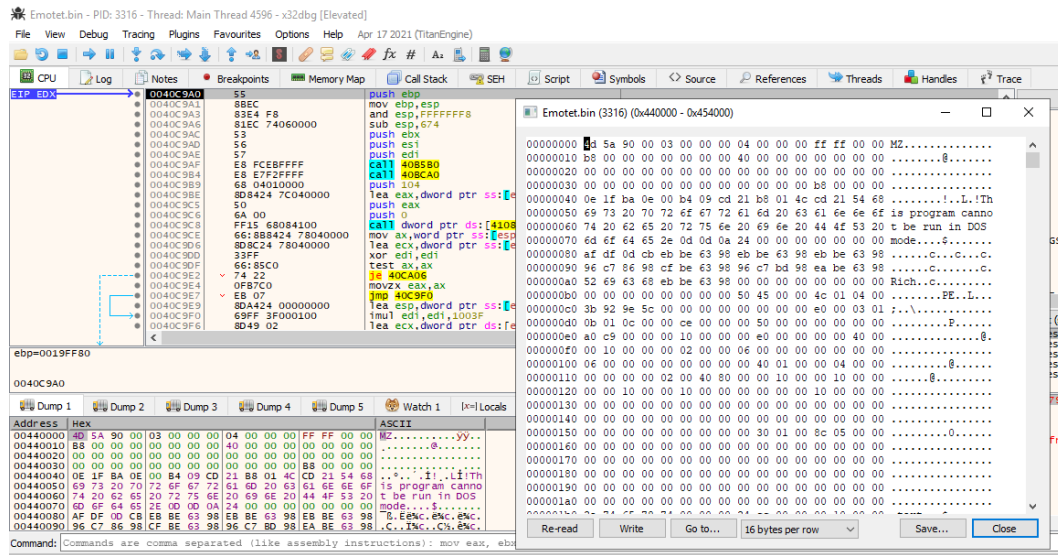


Now we will look the region return earlier by VirtualAlloc i.e. 440000 here we can see the permission i.e. rwx and then double click on it.

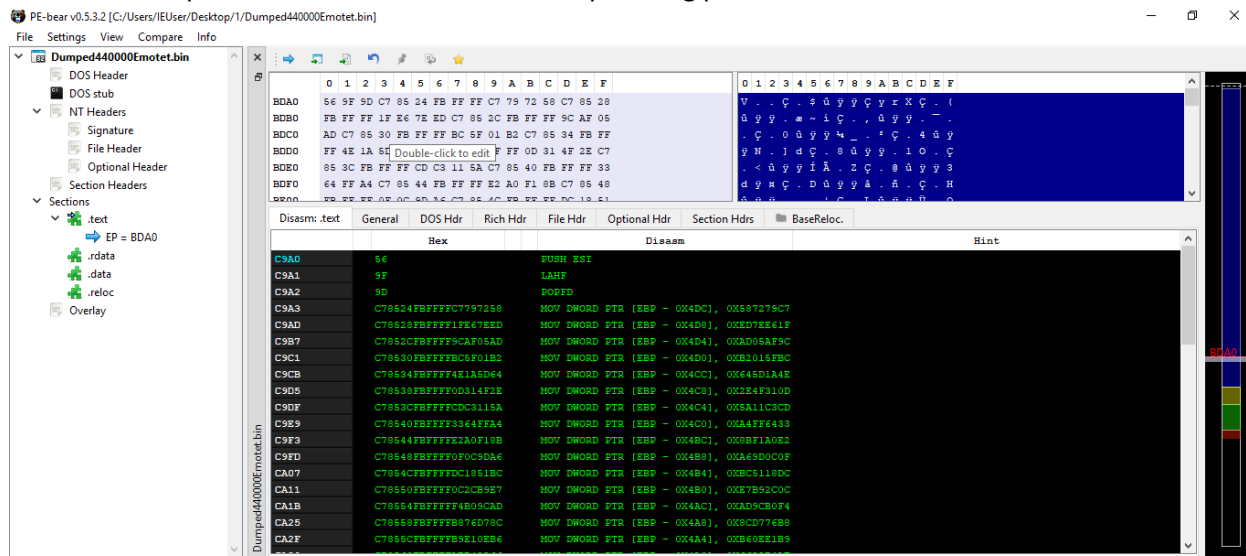


Double click on that memory region. This exe is same as the one we dumped previously in x32dbg.

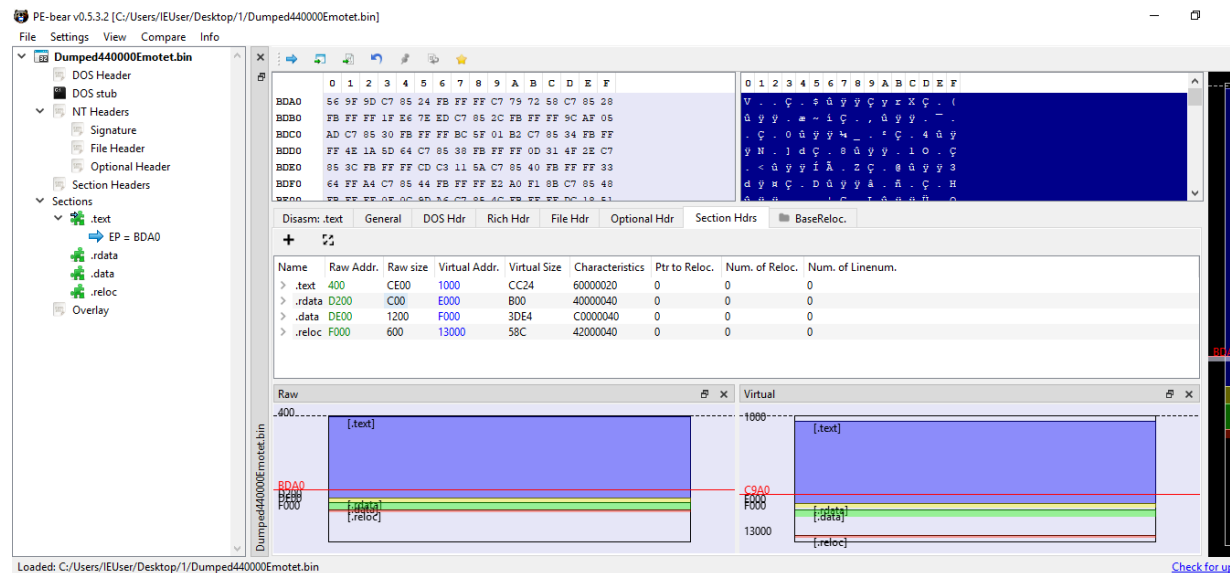
Now right click on it and save it desired location.



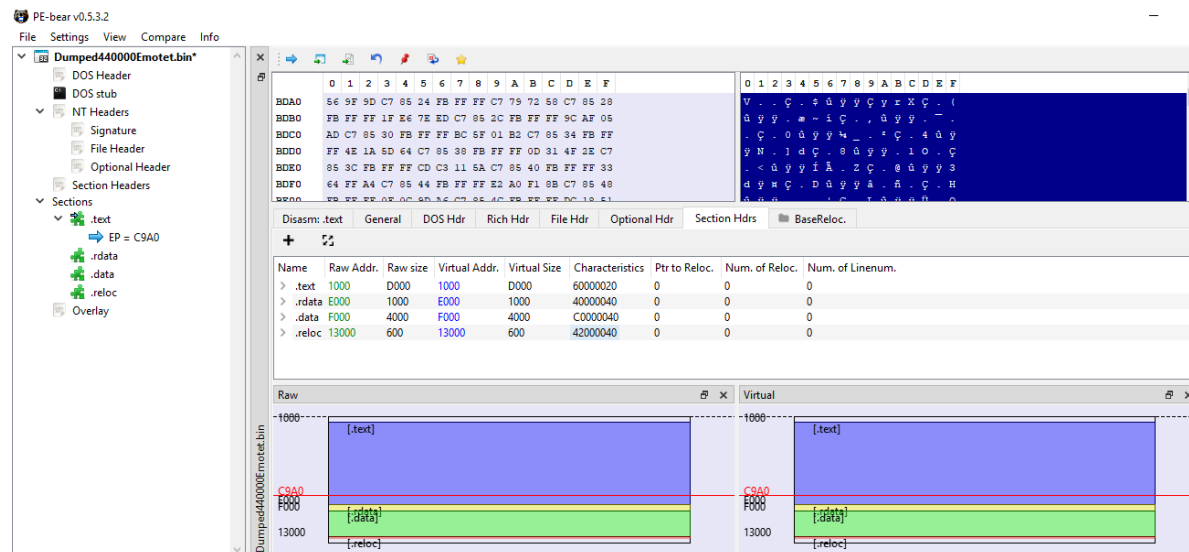
Now we will repair the section header and unmap it using pebear.



Here we need to unmap our section header. To unmap it the raw address and virtual address need to be same .



Here we have fixed the raw address, now we will calculate the raw size. Now make virtual size same as raw size. Also here fix the Image base in Optional Header put the value 440000 and save it.



Here we have finally unpacked our sample we can verify it using Detect it easy

