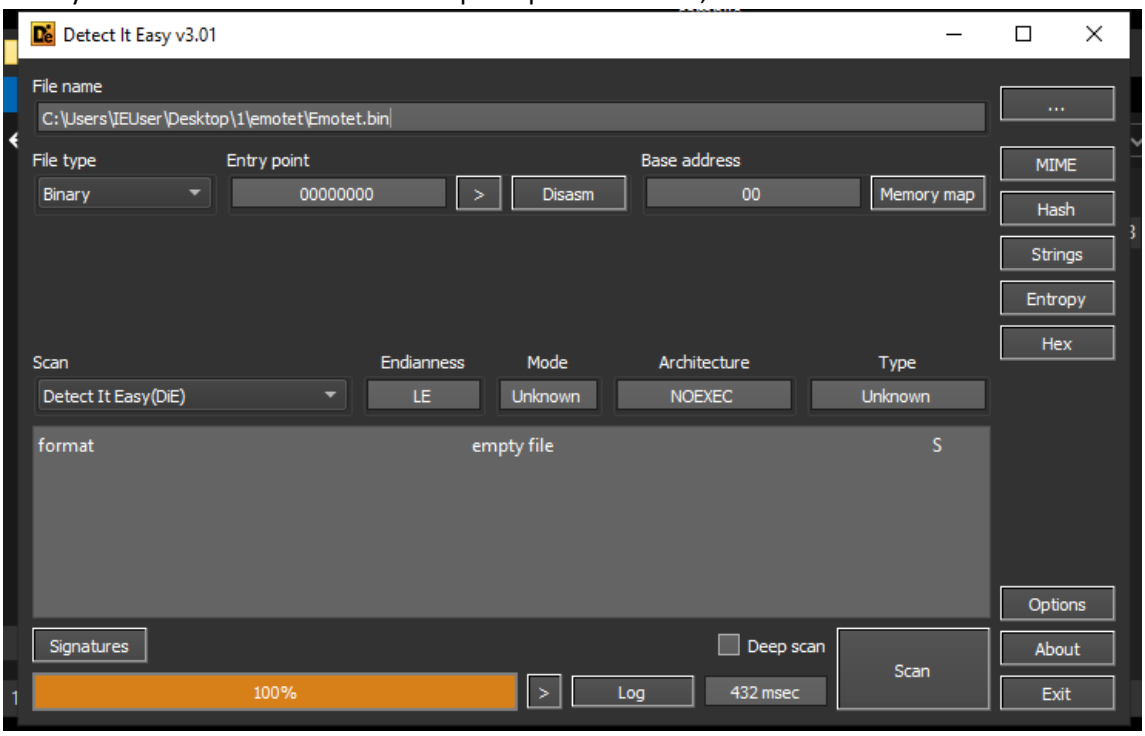
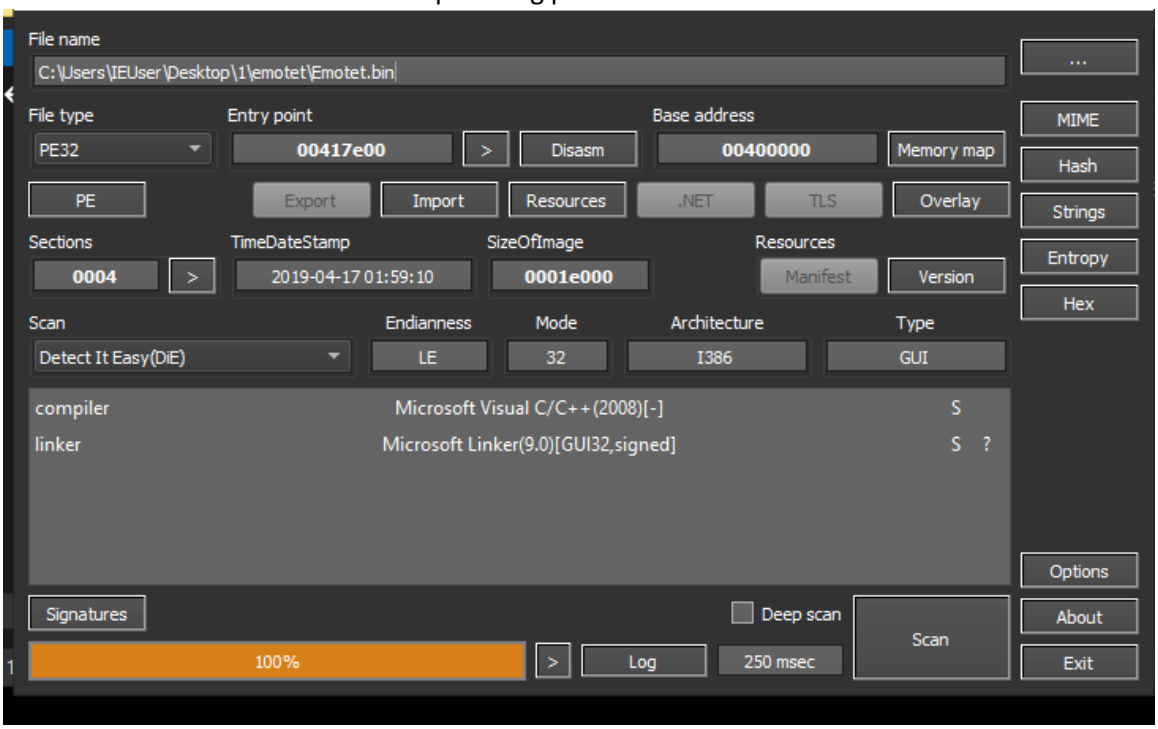
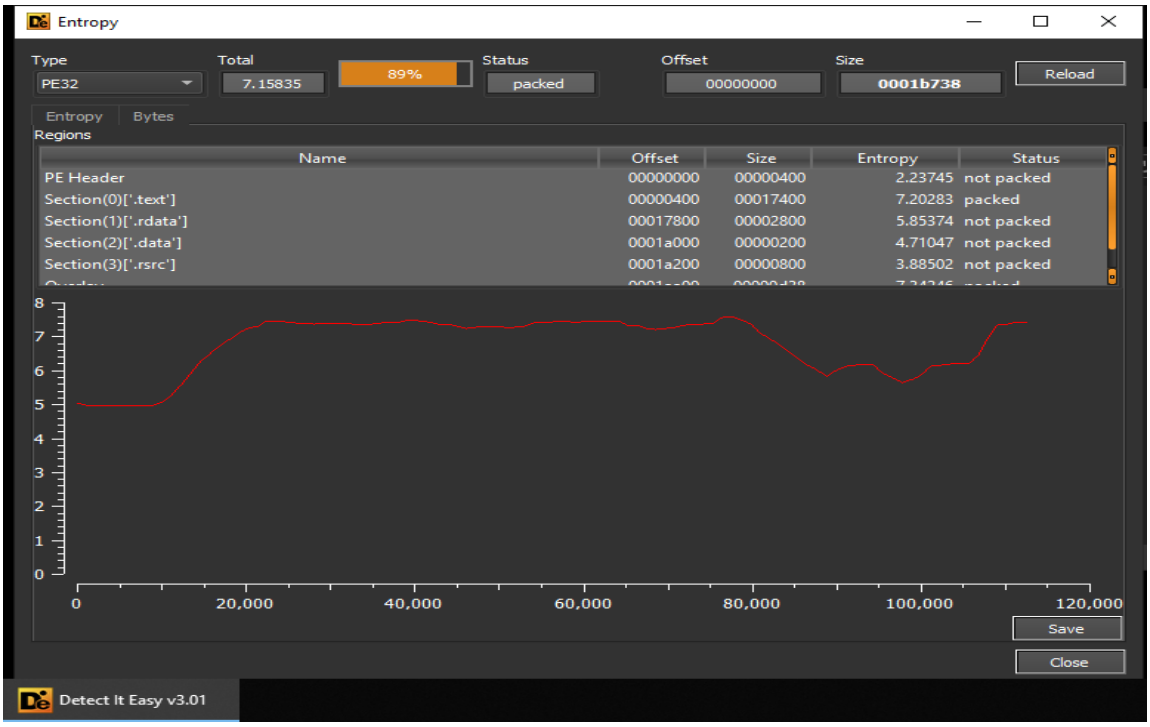
Intially we will check whether the sample is packed or not , for that we can use tool like Detect It Easy.



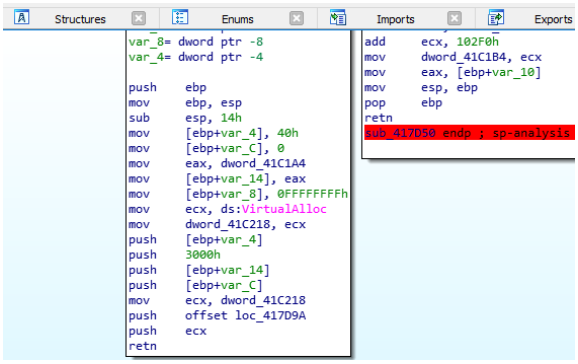
Here there is no mention of the sample being packed.



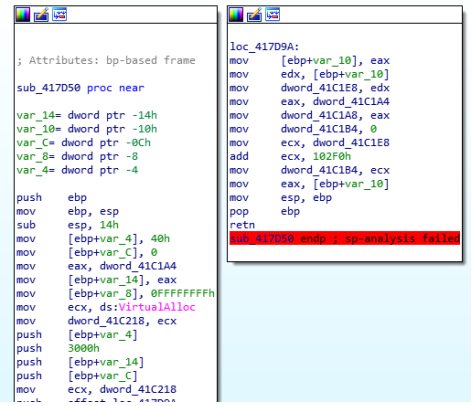
Now here we checked the entropy of the sample was 7.15 which is high , and also here it is showing the sample status as being packed.



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.



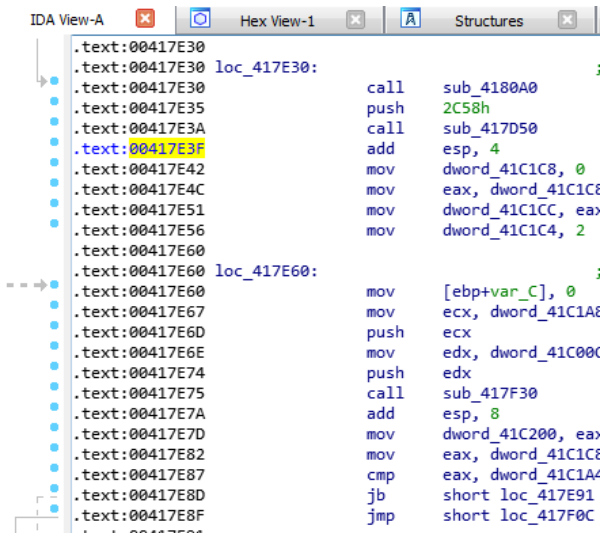
Normally ret execute the code present in top of the stack, so here it is calling VirtualAlloc ,now when VirtualAlloc returns it will return to the loc_41709A and inside it is the final return.



```
; 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], 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
```

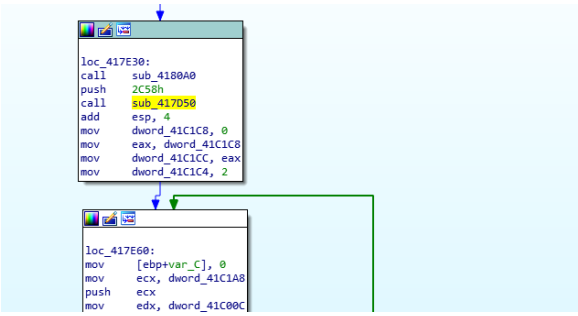
```
loc_417D9A:
mov     [ebp+var_10], eax
mov     edx, [ebp+var_10]
mov     dword_41C1E8, edx
mov     eax, dword_41C1A4
mov     dword_41C1A8, eax
mov     dword_41C1B4, 0
mov     ecx, dword_41C1E8
add     ecx, 102F0h
mov     dword_41C1B4, ecx
mov     eax, [ebp+var_10]
mov     esp, ebp
pop     ebp
retn
sub_417D50 endp ; sp-analysis failed
```

This is where the final return control go.



```
IDA View-A | Hex View-1 | Structures

.text:00417E30
.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
.text:00417E60 loc_417E60:
.text:00417E60                 mov     [ebp+var_C], 0
.text:00417E67                 mov     ecx, dword_41C1A8
.text:00417E6D                 push    ecx
.text:00417E6E                 mov     edx, dword_41C00C
.text:00417E74                 push    edx
.text:00417E75                 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                 jb      short loc_417E91
.text:00417E8F                 jmp     short loc_417F0C
```

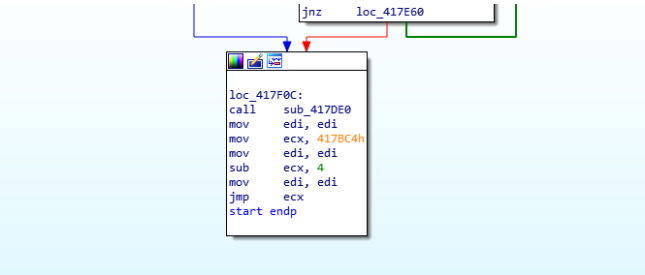
Here we can put breakpoint and this can be unpacking code location and once the code is unpacked we will be looking for that , like any abnormal jump.



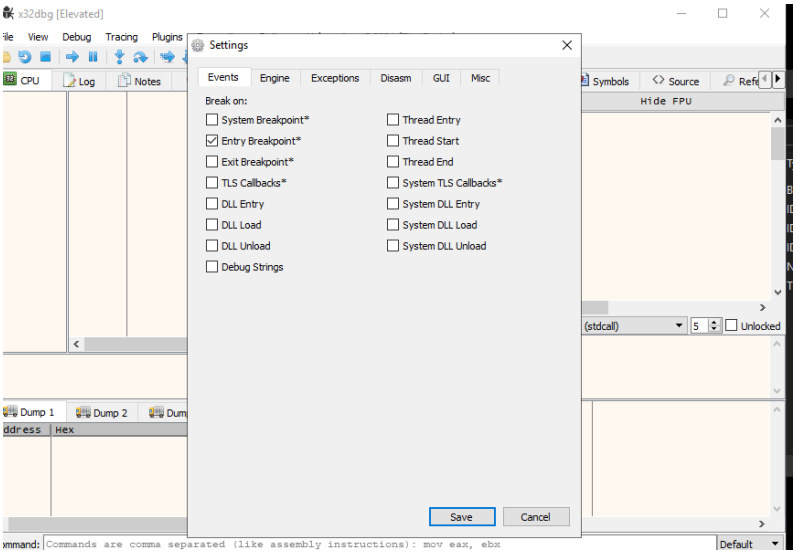
```
loc_417E30:
call    sub_4180A0
push    2C58h
call    sub_417D50
add     esp, 4
mov     dword_41C1C8, 0
mov     eax, dword_41C1C8
mov     dword_41C1CC, eax
mov     dword_41C1C4, 2
```

```
loc_417E60:
mov     [ebp+var_C], 0
mov     ecx, dword_41C1A8
push    ecx
mov     edx, dword_41C00C
```

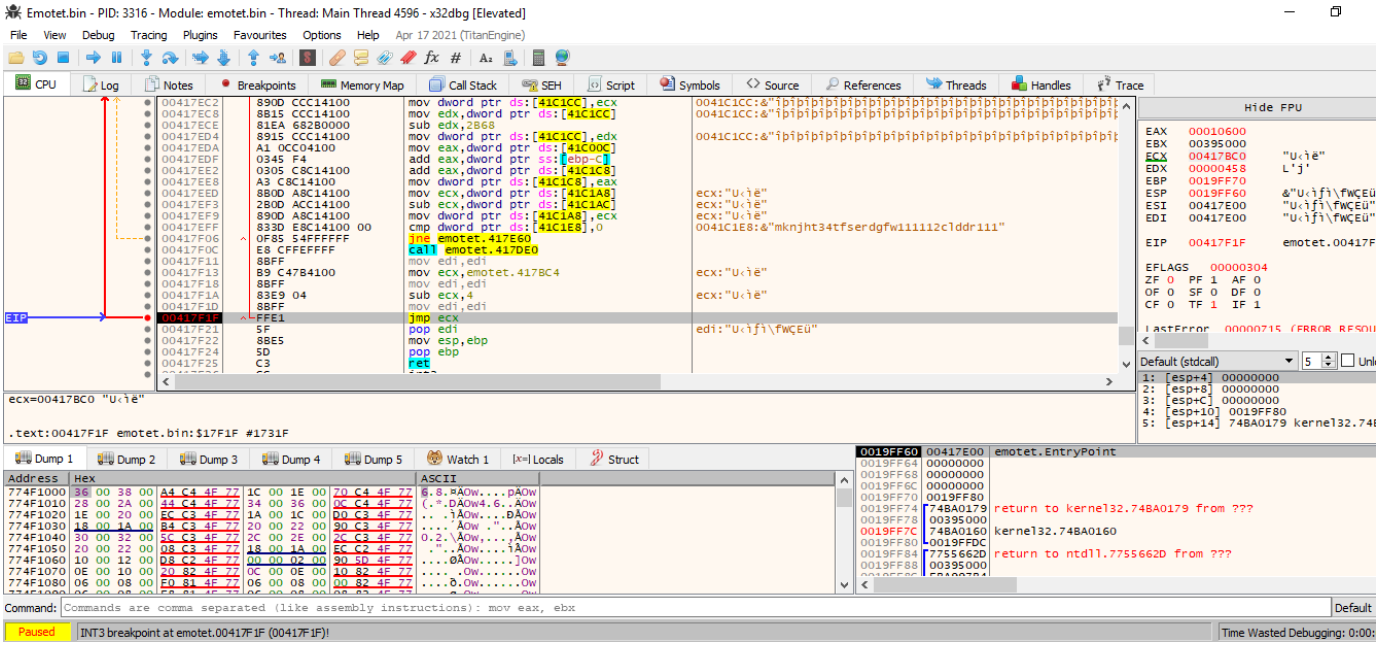
Here we find an abnormal jump.



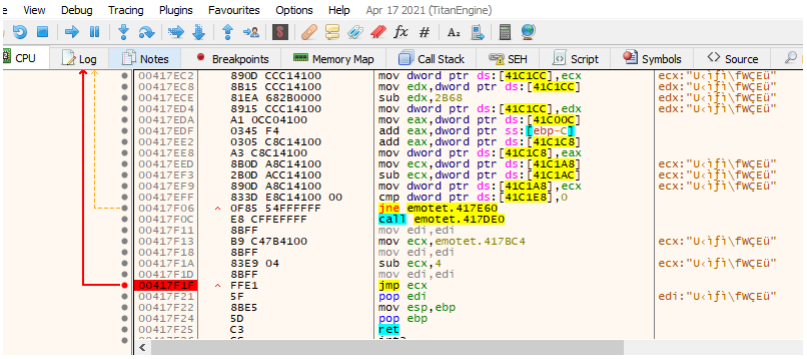
Now we will load the sample in x32dbg. Here in Options->Preferences->Uncheck the System Breakpoint and TLS Callbacks.



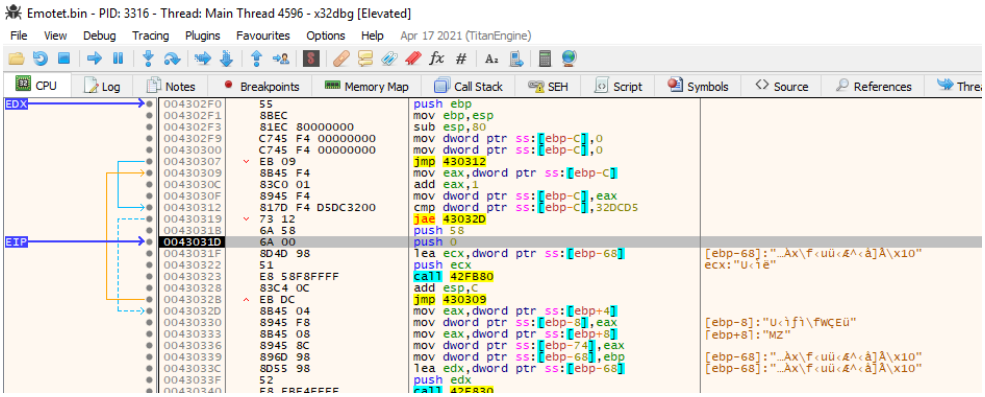
Now here we have put the breakpoint on abnormal jump , just before the unpacked code gets run.



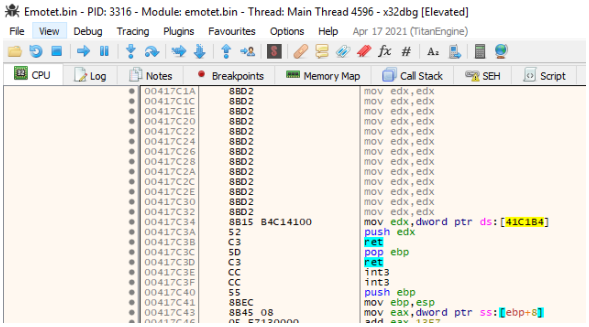
We have here reached the breakpoints , just step over the ecx.



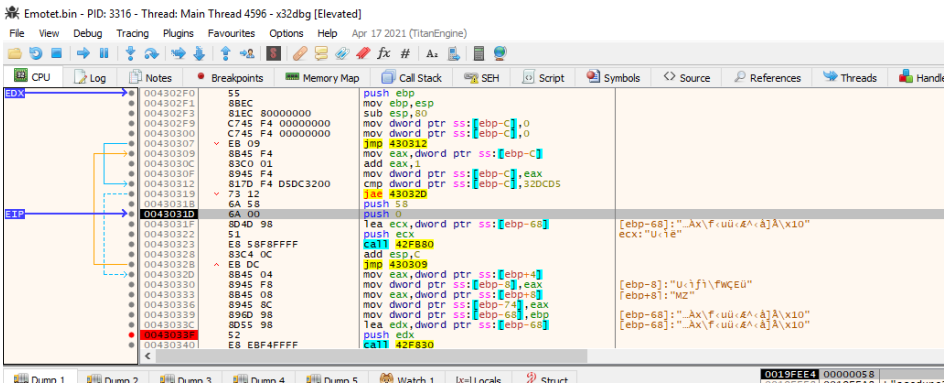
Now here we have taken the jump to ecx. This is the location where the part of unpacked code resides.



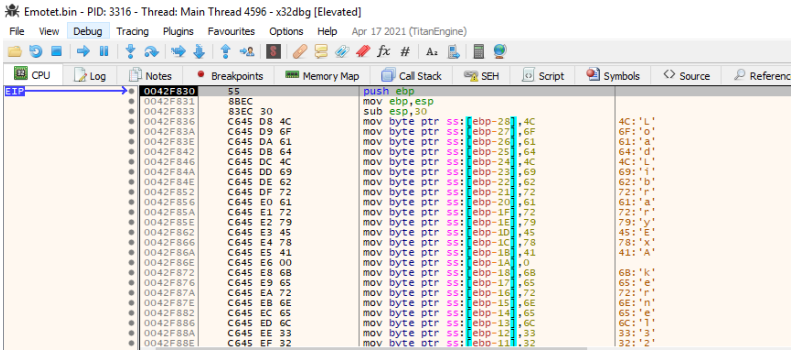
Here in this unpacked code there is an abnormal epilogue i.e. push edx before ret. In edx there can be another unpacked code,put a breakpoint here.



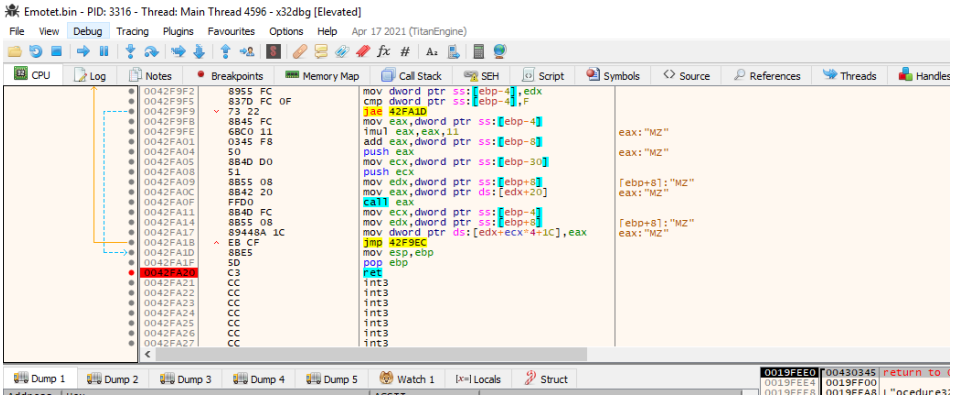
This is the another newly unpacked section.



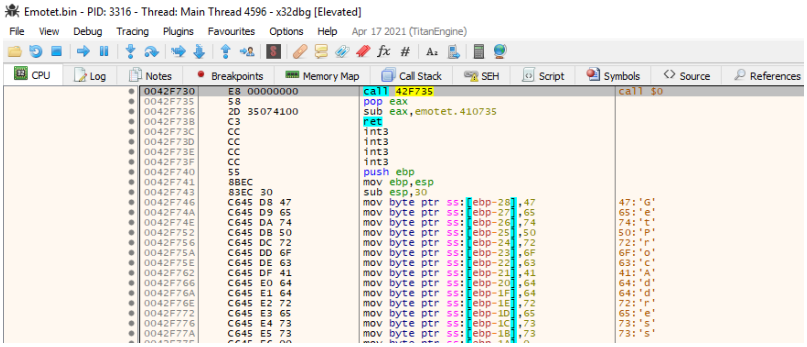
Here we can see it is moving the strings in stack and then using them as parameters for the call .

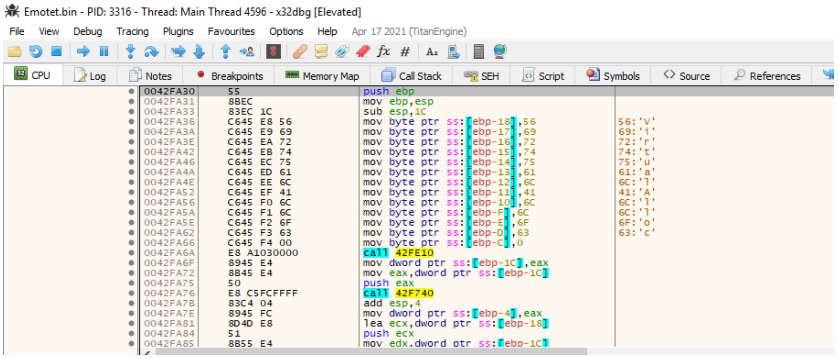


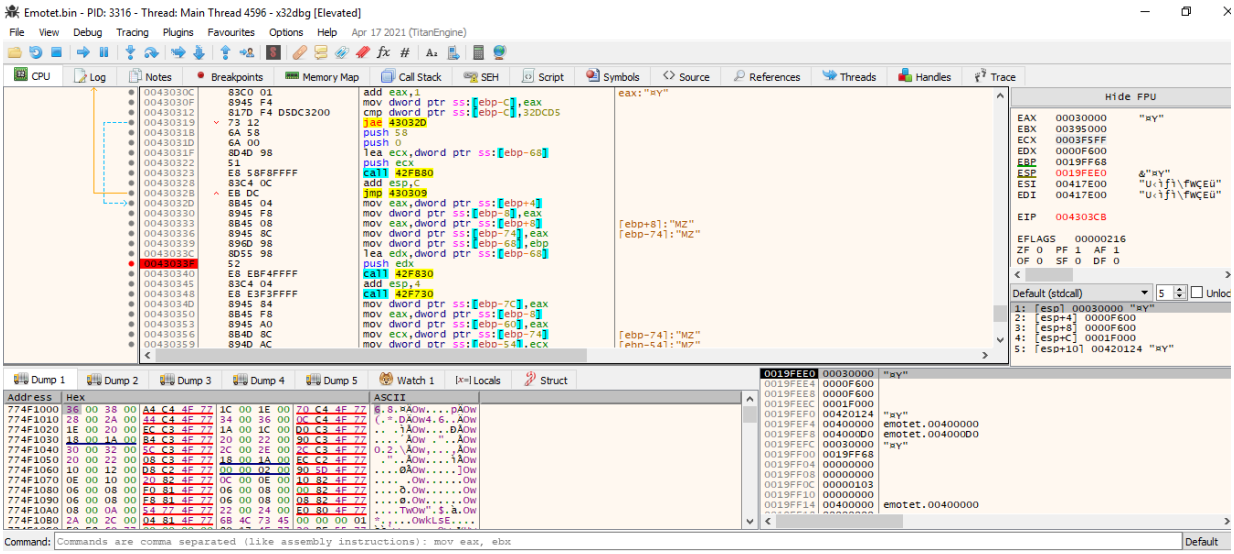
Here now it will return to the caller function.



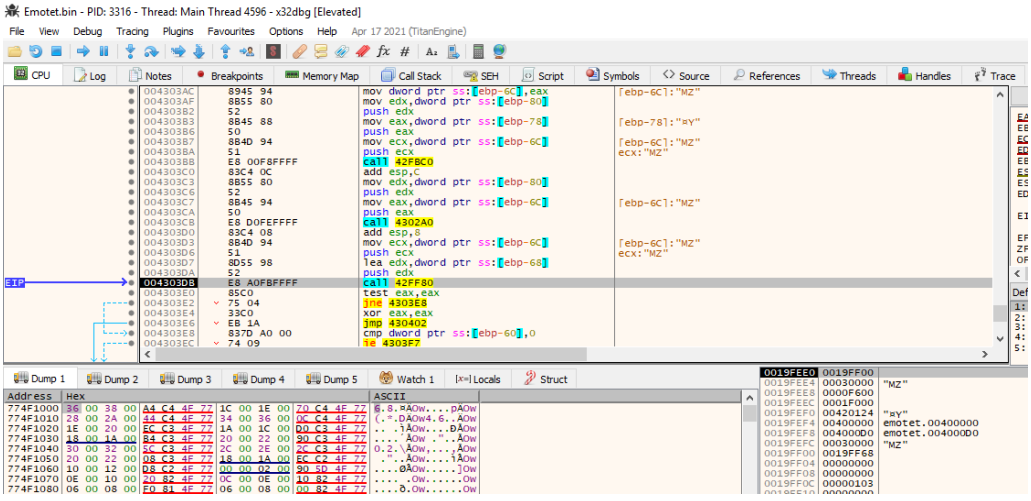
Here the another similar function doing the same approach.



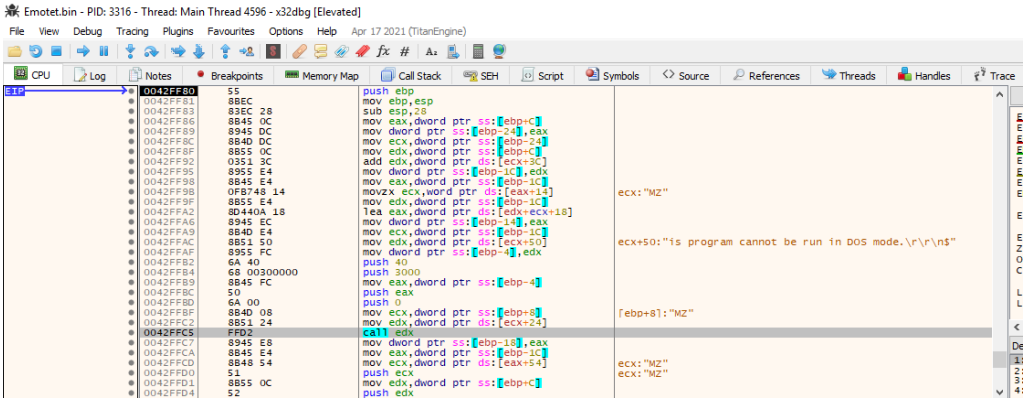


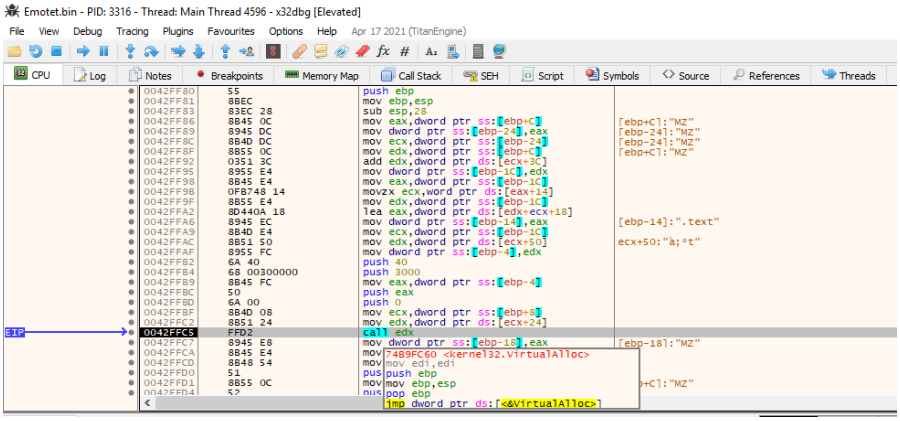


In the below image we have reached the last function call, we will step into it



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.





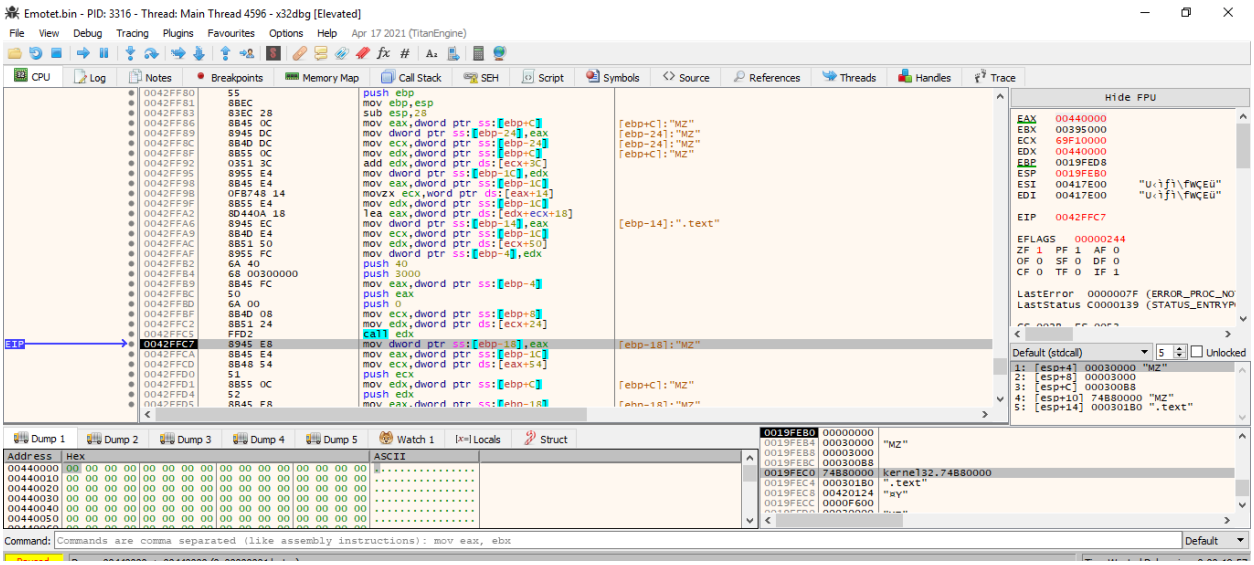

```
LPVOID VirtualAlloc(
  [in, optional] LPVOID lpAddress,
  [in]           SIZE_T dwSize,
  [in]           DWORD  flAllocationType,
  [in]           DWORD  flProtect
);
```

PAGE_EXECUTE_READWRITE 0x40	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.

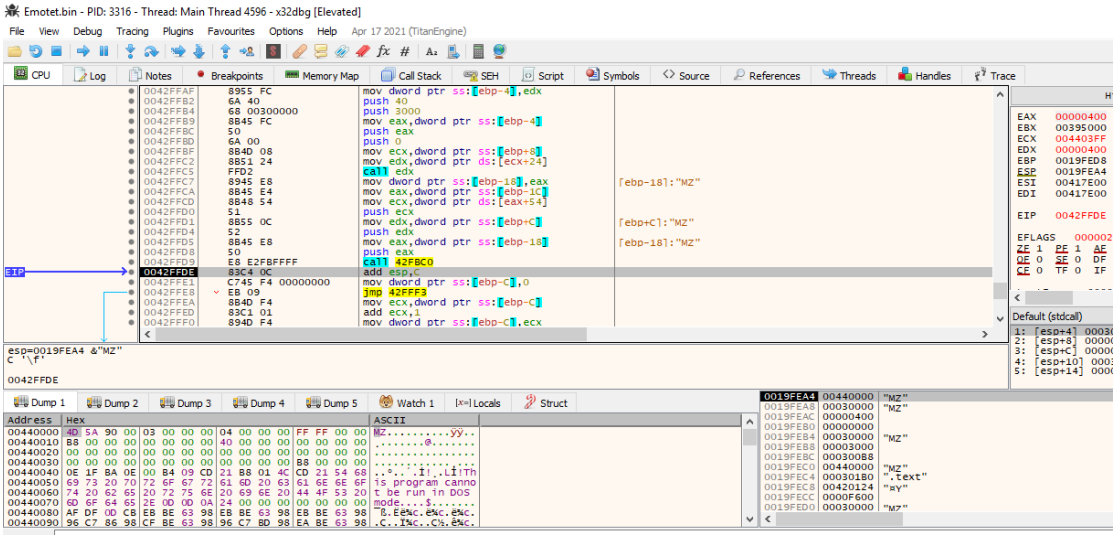
VirtualAlloc Reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process. Memory allocated by this function is automatically initialized to zero.

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

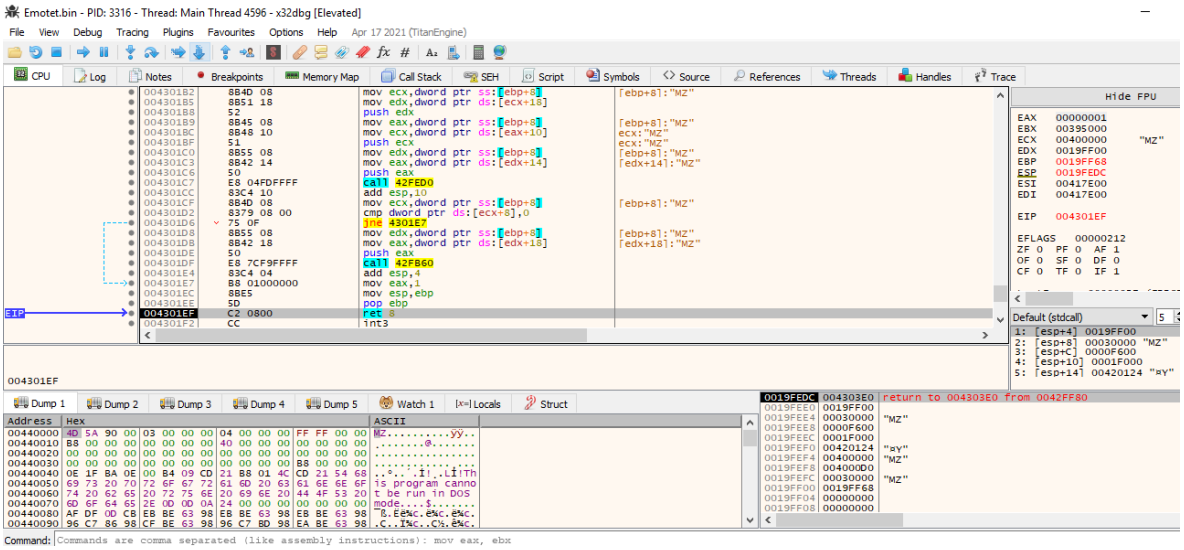
In the image below after stepping over the call edx now we can see the value return base address of the new allocated memory region in the eax i.e 440000 , now we will follow the allocated region in dump.



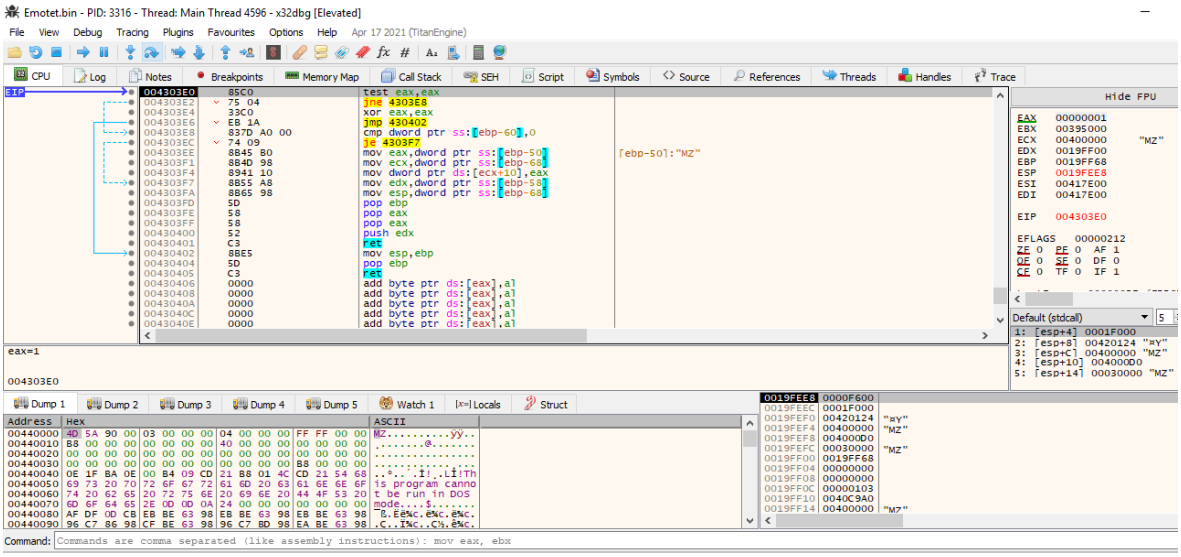
After this call we can see that it has dumped an PE file in the allocated region.



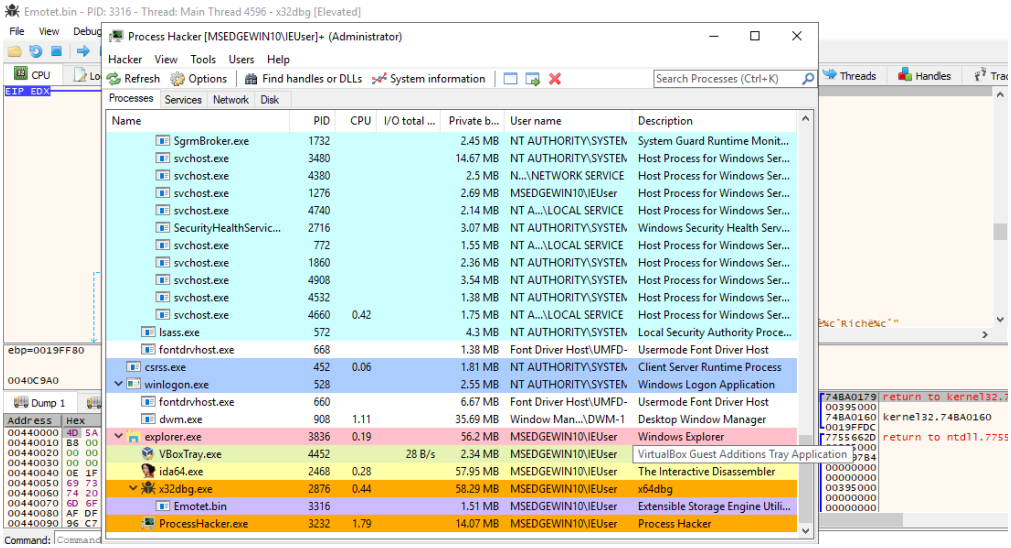
Till now it has allocated different sections of pe in that location.



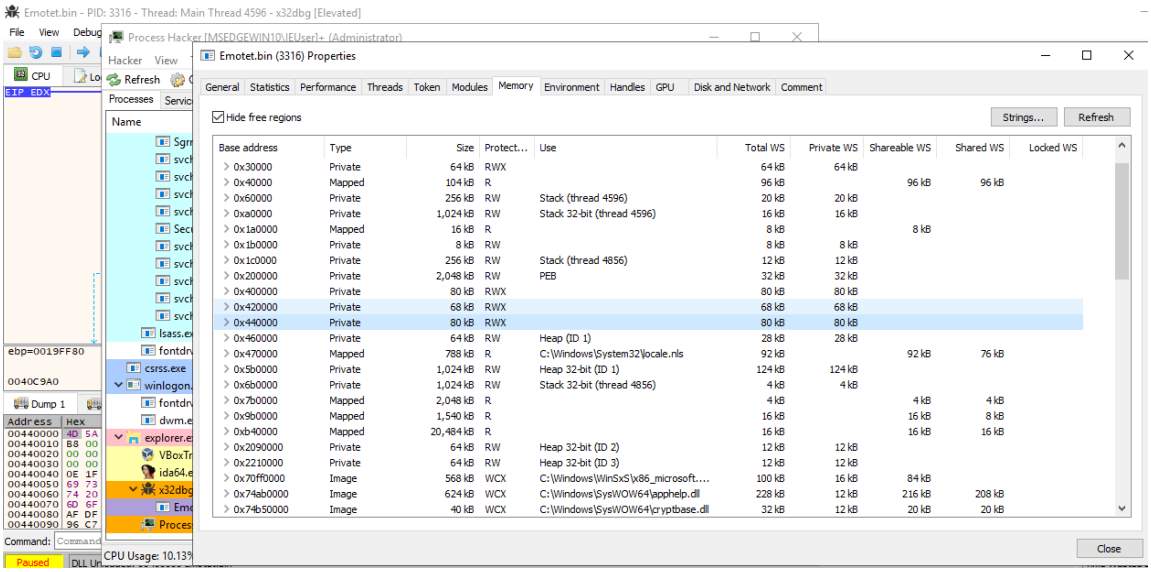
now it has finished the unpacking , now we can dump that into the file.



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

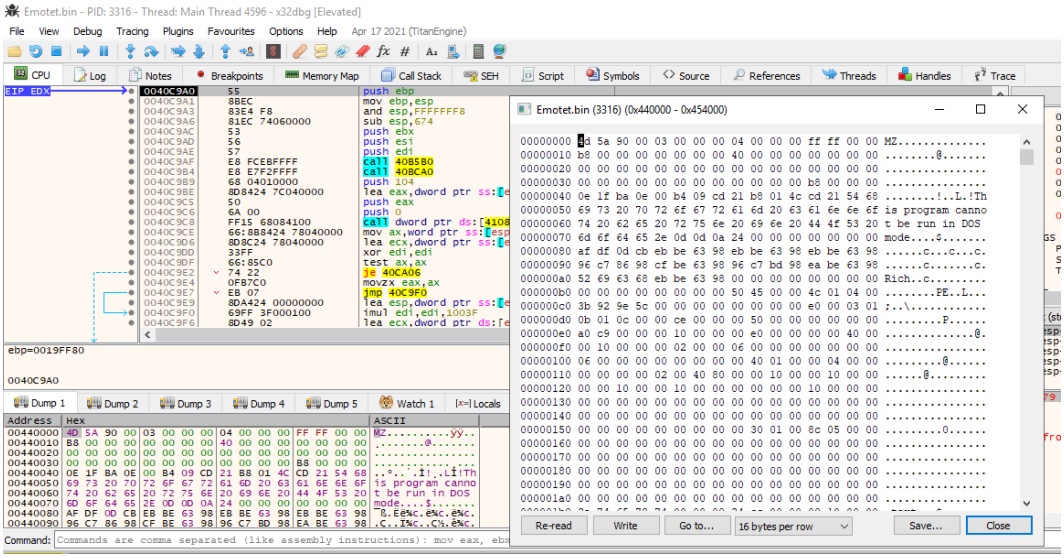


Now we will look the region return earlier by VirtualAlloc i.e. 440000 here we can see the permissiion i.e. rwx and then double click on it.

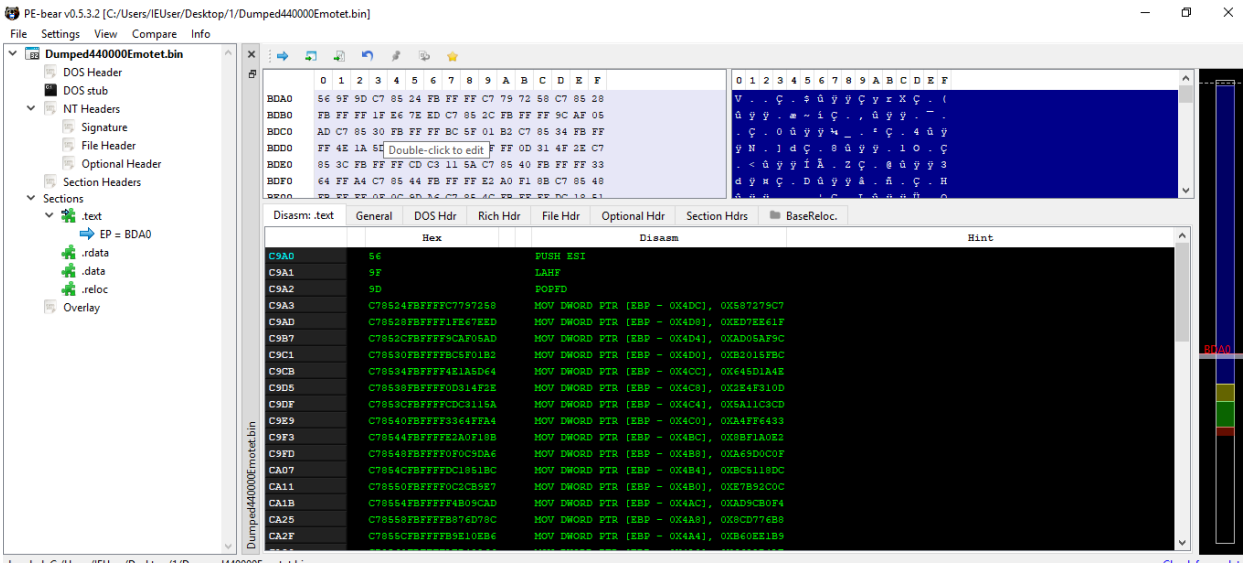


Double click on that memory region.This exe is same as the one we dumped previosly in x32dbg.

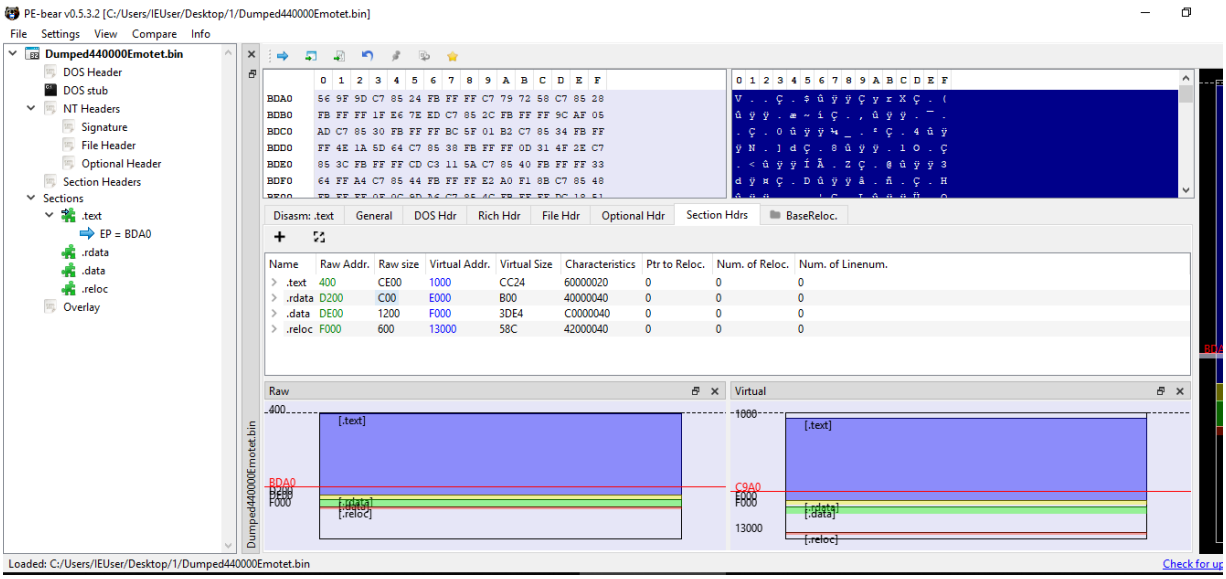
Now right click on it and save it desired location.



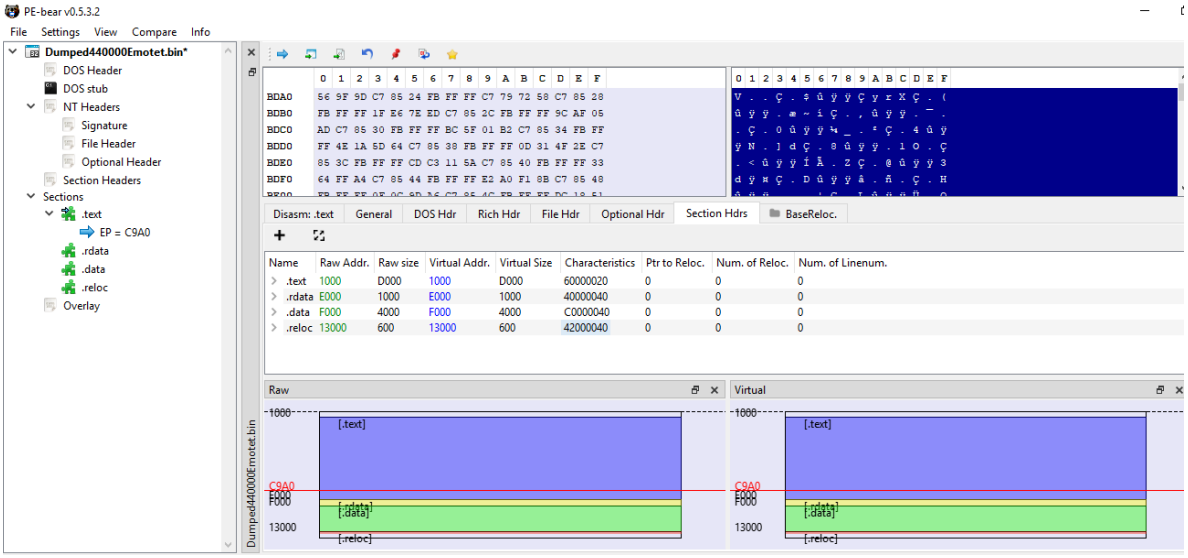
Now we will repair the section header and unmap it using pebear.



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