Class 8

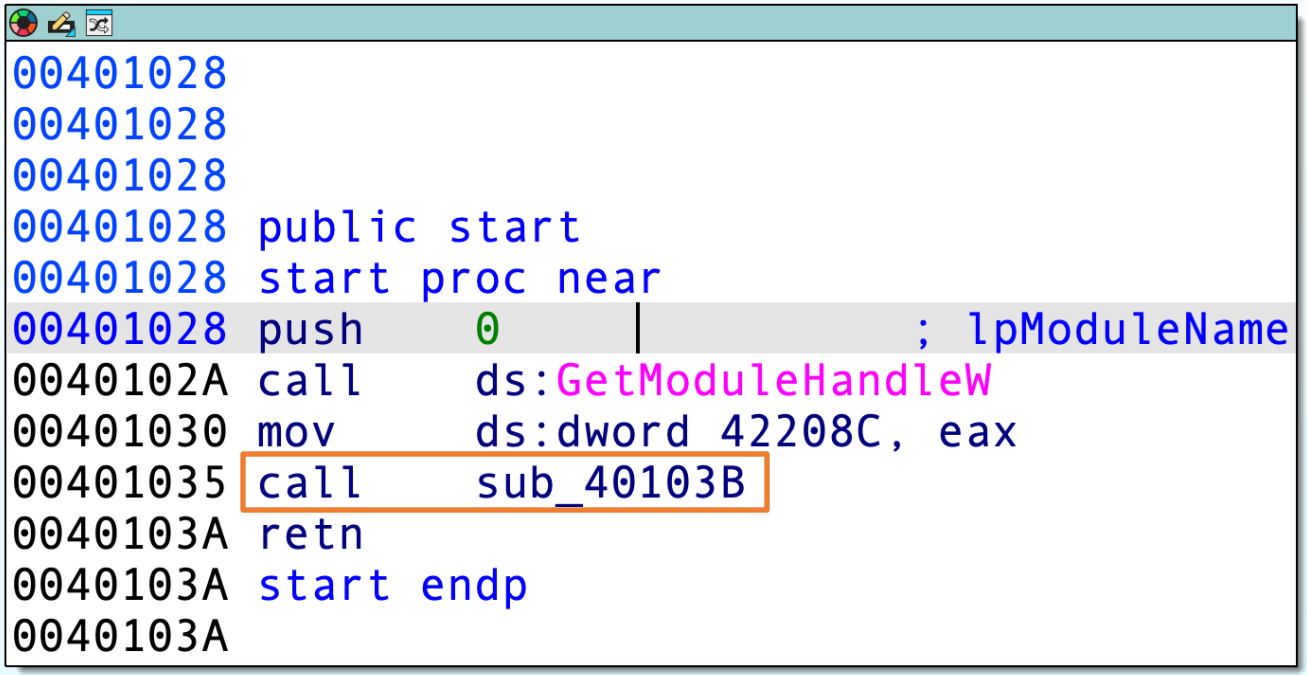
CSC 471 Modern Malware Analysis
Static Analysis & Dynamic Analysis (2) : (De)Obfuscation

Si Chen (schen@wcupa.edu)

4298F9DDA63C3C1B17FEF433C082107A

(Trojan.Win32.Agent.b)

Load 4298F9DDA63C3C1B17FEF433C082107A into IDA



```
00401028
00401028
00401028
00401028 public start
00401028 start proc near
00401028 push    0                      ; lpModuleName
0040102A call    ds:GetModuleHandleW
00401030 mov     ds:dword_42208C, eax
00401035 call    sub_40103B
0040103A retn
0040103A start endp
0040103A
```

Here, we can observe that after obtaining its own module handle, the program assigns the return value (stored in eax) to an address, followed by a call. Let's follow this call to see what it does

West Chester University

4298F9DDA63C3C1B17FEF433C082107A

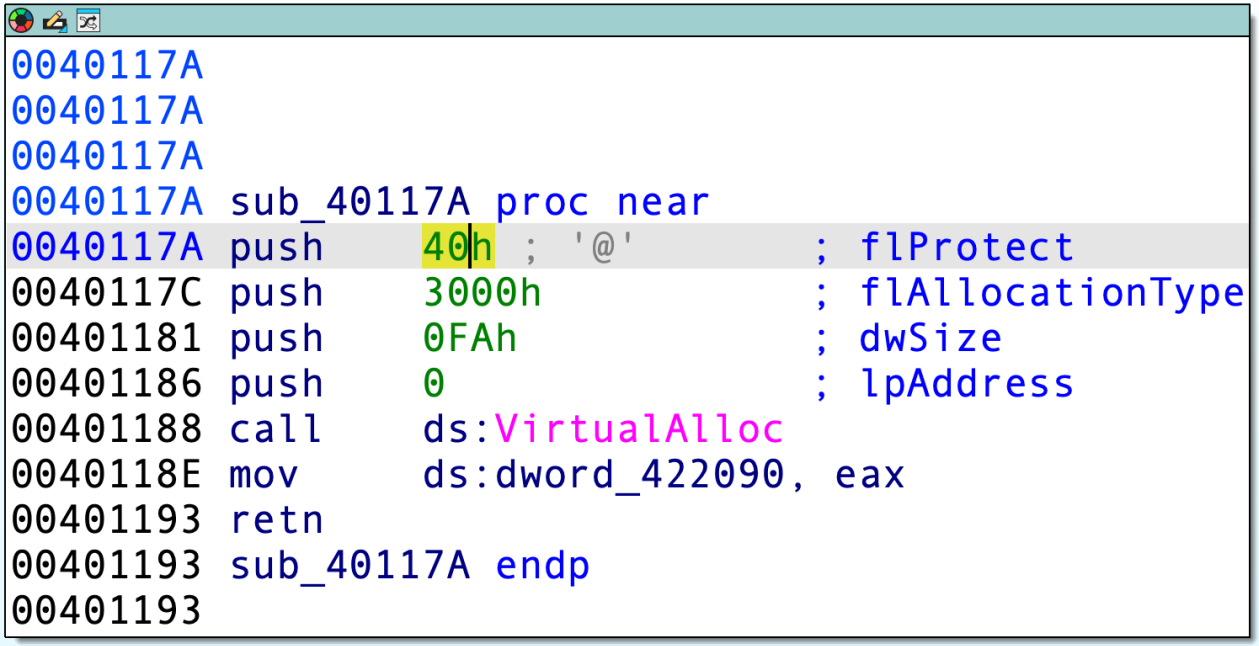
First, let's jump into the first call to examine its content

```
0040103B
0040103B
0040103B sub_40103B proc near
0040103B pminsw  xmm0, xmm1
0040103F pminsw  xmm3, xmm4
00401043 call    sub_40117A
00401048 call    sub_4010E6
0040104D jmp     short loc_40106C
```

```
0040106C
0040106C loc_40106C:
0040106C push    offset LibFileName ; "user32.dll"
00401071 call    ds:LoadLibraryA
00401077 push    offset ProcName ; "user_api_function"
0040107C push    eax             ; hModule
0040107D call    ds:GetProcAddress
00401083 call    ds:GetLastError
00401089 add     eax, offset unk_422011
```


West Chester University

4298F9DDA63C3C1B17FEF433C082107A



```
0040117A
0040117A
0040117A
0040117A sub_40117A proc near
0040117A push    40h ; '@'          ; flProtect
0040117C push    3000h              ; flAllocationType
00401181 push    0FAh               ; dwSize
00401186 push    0                  ; lpAddress
00401188 call    ds:VirtualAlloc
0040118E mov     ds:dword_422090, eax
00401193 retn
00401193 sub_40117A endp
00401193
```

As we can see, the purpose of this call is to allocate memory space using the VirtualAlloc function. It is reasonable to believe that the decrypted code will likely be stored here.

West Chester University

4298F9DDA63C3C1B17FEF433C082107A

```
0040103B
0040103B
0040103B sub_40103B proc near
0040103B pminsw   xmm0, xmm1
0040103F pminsw   xmm3, xmm4
00401043 call     sub_40117A
00401048 call     sub_4010E6
0040104D jmp      short loc_40106C
```

```
0040106C
0040106C loc_40106C:
0040106C push     offset LibFileName ; "user32.dll"
00401071 call     ds:LoadLibraryA
00401077 push     offset ProcName ; "user_api_function"
0040107C push     eax             ; hModule
0040107D call     ds:GetProcAddress
00401083 call     ds:GetLastError
00401089 add      eax, offset unk_422011
```

Returning to the previous level, let's examine the content of the second call

4298F9DDA63C3C1B17FEF433C082107A

```
004010E6
004010E6
004010E6
004010E6 sub_4010E6 proc near
004010E6 mov     ebx, ds:dword_422090
004010EC or      dx, 0CB26h
004010F1 and     eax, ebx
004010F3 mov     edi, ebx
004010F5 not     ax
004010F8 not     dx
004010FB mov     esi, offset unk_445359
00401100 xor     edx, 0CB55h
00401106 and     edx, edx
00401108 mov     ecx, 0FAh
0040110D xor     ax, di
00401110 or      dx, 1Dh
00401114 rep movsb
00401116 retn
00401116 sub_4010E6 endp
00401116
```

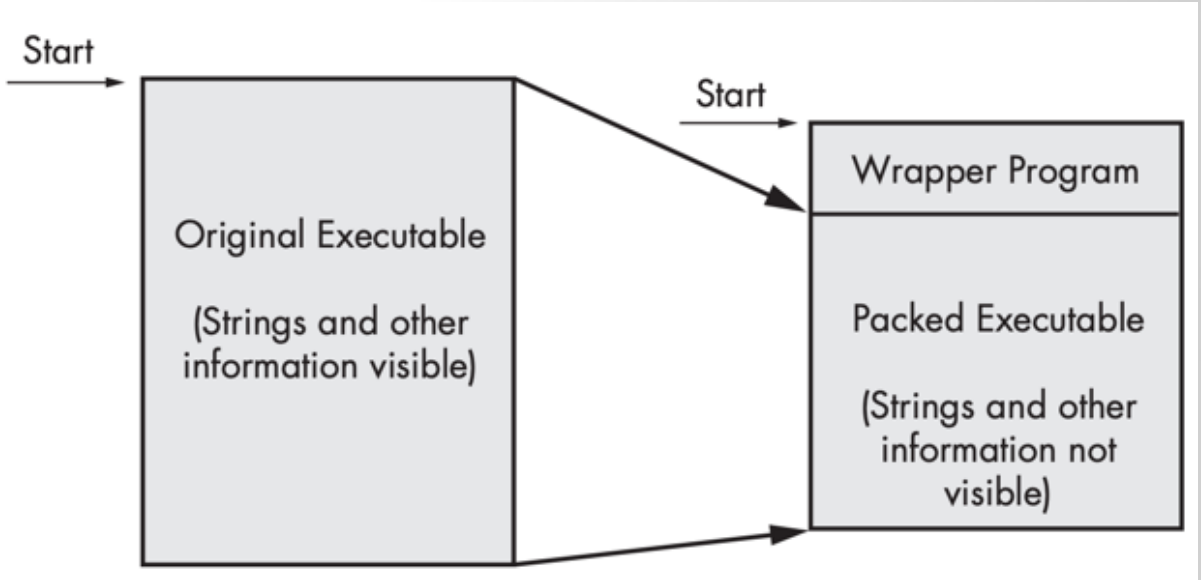
In fact, this is a self-protection mechanism used by viruses, known as obfuscation, or it can also be understood as a "shell" written by the virus author for their malicious program.

Here, we can see that operations such as and, not, and xor are used for decryption. This is something that should not appear in a normal program, so we can directly flag it as malicious: **Trojan.Win32.Agent.c**.

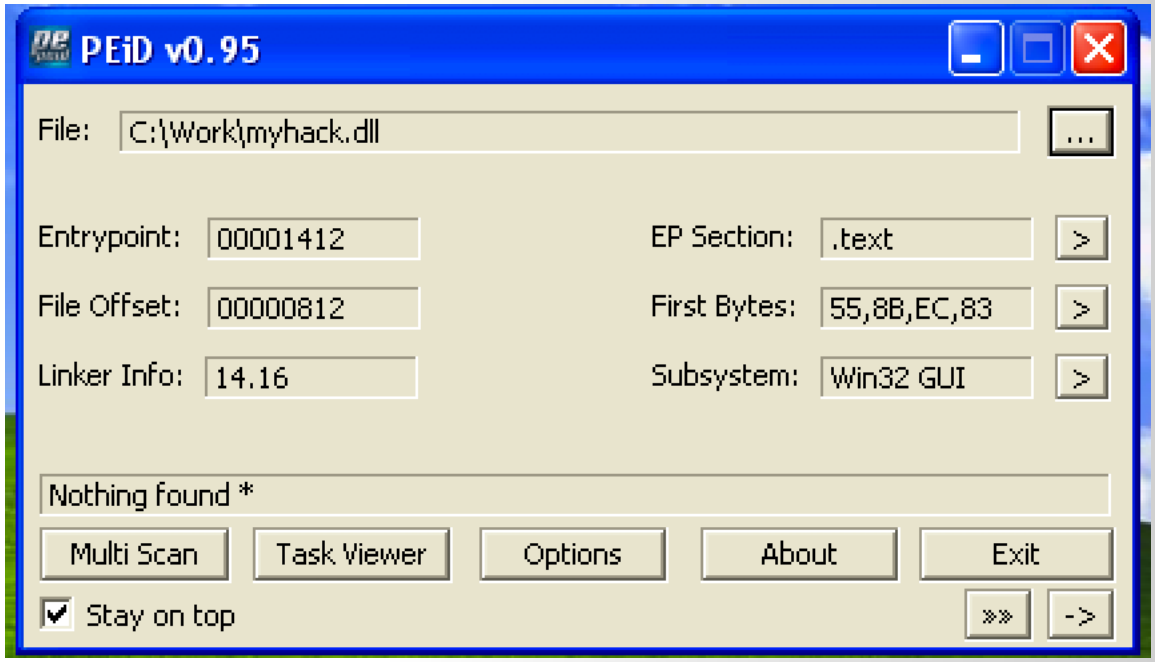
West Chester University

Packed and Obfuscated Malware

- Malware writers often use **packing or obfuscation** to make their files more difficult to detect or analyze.
- **Obfuscated** programs are ones whose execution the malware author has attempted to hide.
- **Packed** programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed.
- Both techniques will severely limit your attempts to statically analyze the malware.



Packed and Obfuscated Malware



West Chester University

Packers and Cryptos

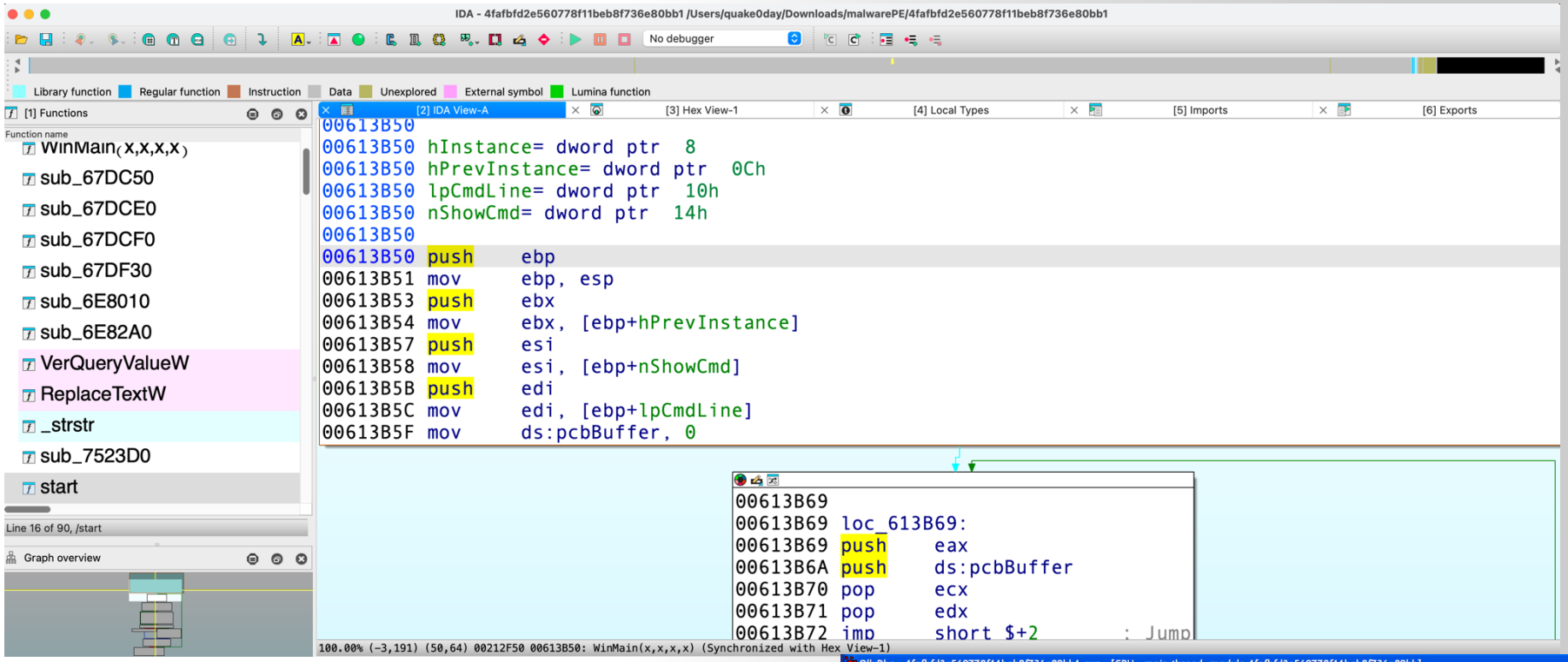
```
➜  ~ upx -o myhack_packed.dll myhack.dll
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95        Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     75264 ->     39424   52.38%    win32/pe     myhack_packed.dll

Packed 1 file.
```

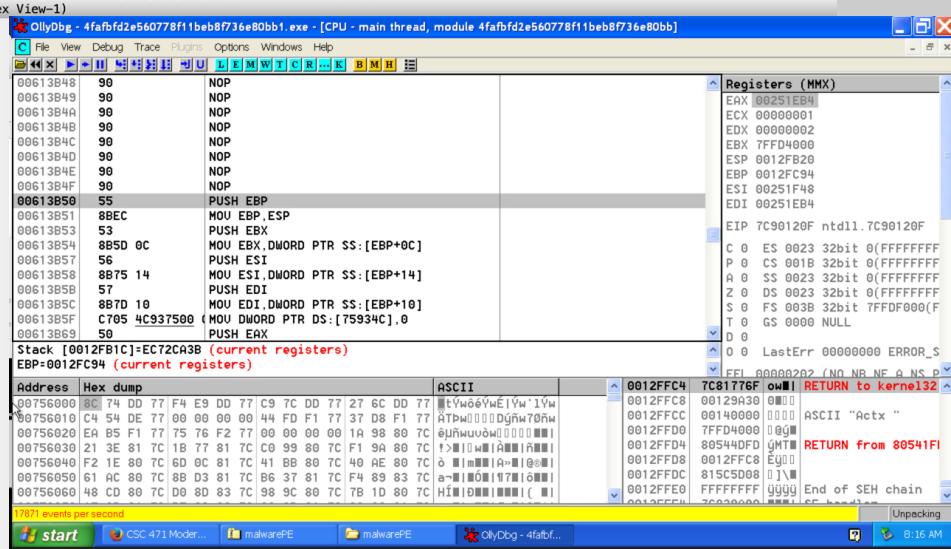

West Chester University

4fafbfd2e560778f11beb8f736e80bb1

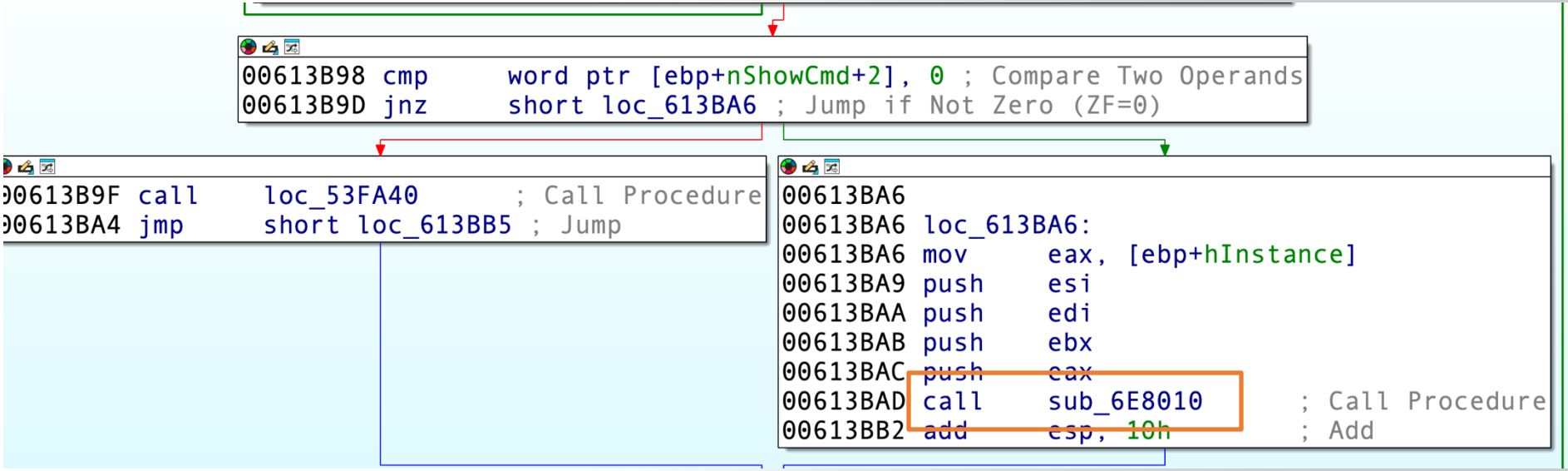
(Trojan.Win32.Agent.b)

4fafbfd2e560778f11beb8f736e80bb1





IDA jump to 0x00613B50, which is the location of the main function. This is where the actual code of the sample is executed and is the focus of our analysis.



In this main function, we primarily analyze the call instructions. For example, let's look at the call located at 0x00613BAD.

```
006E8010 sub     esp, 834h       ; Integer Subtraction
006E8016 push    ebx
006E8017 xor     ebx, ebx        ; Logical Exclusive OR
006E8019 push    esi
006E801A mov     [esp+83Ch+var_1], bl
006E8021 mov     [esp+83Ch+var_B], bl
006E8028 mov     [esp+83Ch+var_2], 6Ch ; 'l'
006E8030 call    ds:GetEnvironmentStrings ; Indirect Call Near Procedure
006E8036 mov     ds:dword_76046C, eax
006E803B call    ds:GetProcessHeap ; Indirect Call Near Procedure
006E8041 mov     ds:dword_7603D0, eax
006E8046 mov     [esp+83Ch+var_C], 72h ; 'r'
006E804E call    sub_46B1C0      ; Call Procedure
006E8053 mov     al, [esp+83Ch+var_2]
006E805A mov     dl, [esp+83Ch+var_C]
006E8061 mov     cl, 73h ; 's'
006E8063 push    offset Buffer   ; lpString2
006E8068 push    offset Buffer   ; lpString1
006E806D mov     [esp+844h+var_11], cl
006E8074 mov     [esp+844h+var_3], al
006E807B mov     [esp+844h+var_4], 64h ; 'd'
006E8083 mov     [esp+844h+var_F], dl
006E808A mov     [esp+844h+var_E], cl
006E8091 mov     [esp+844h+var_5], 2Eh ; '.'
006E8099 mov     [esp+844h+var_6], al
006E80A0 mov     [esp+844h+var_7], al
```

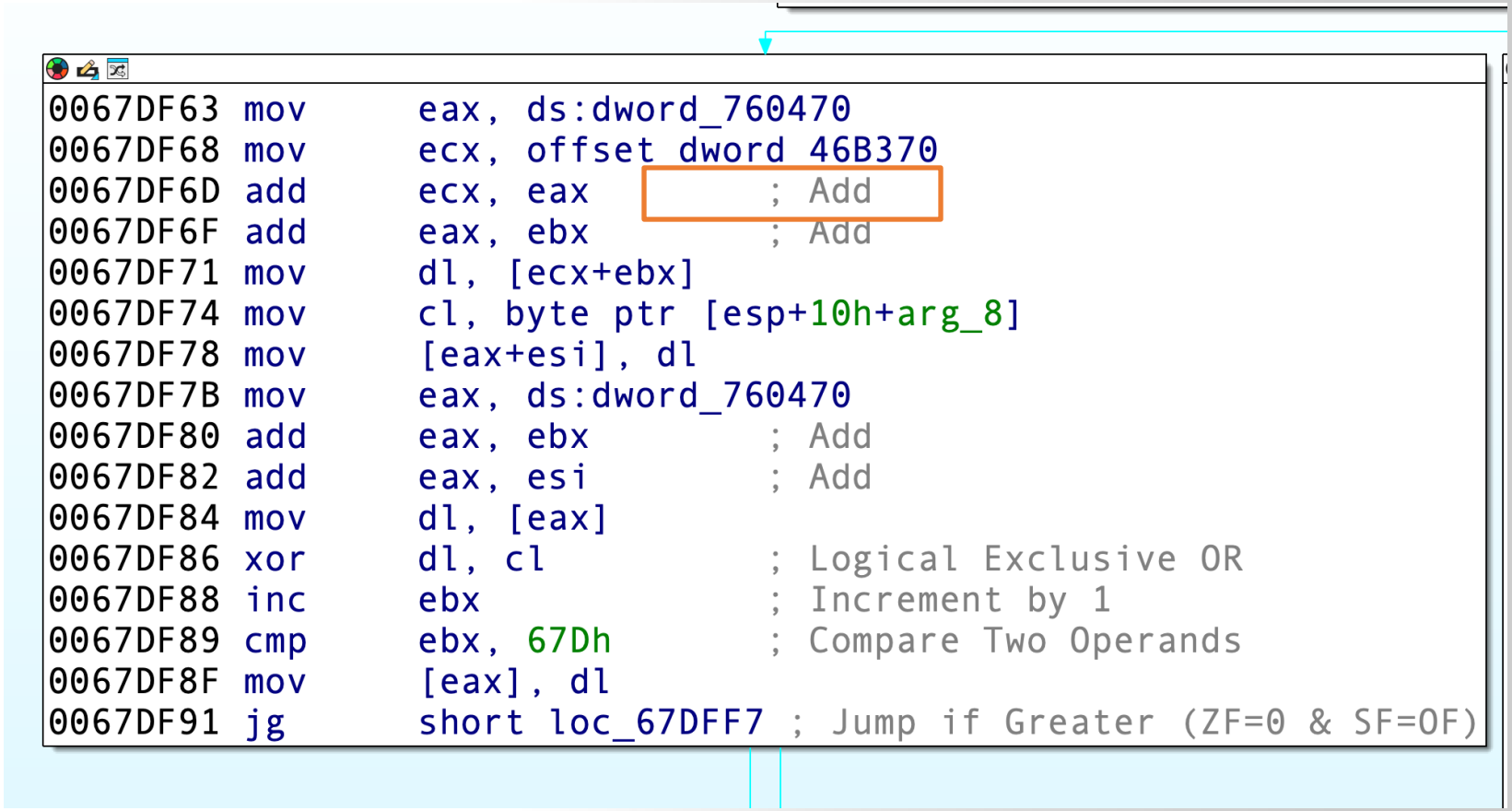
- In this code, we can see many instances of single characters being moved into memory. As mentioned earlier, this is highly suspicious.

```
006E81E0 mov     cl, [esp+83Ch+var_C]
006E81E7 mov     [esp+83Ch+var_834], edx
006E81EB mov     dl, [esp+83Ch+var_11]
006E81F2 mov     [esp+83Ch+var_E], dl
006E81F9 mov     edx, offset dword_53FA20
006E81FE add     edx, 10h          ; Add
006E8201 mov     [esp+83Ch+var_F], cl
006E8208 mov     cl, [esp+83Ch+var_2]
006E820F push    edx               ; int
006E8210 push    eax               ; int
006E8211 push    ebx               ; nIndex
006E8212 mov     [esp+848h+var_82C], offset loc_4010B0
006E821A mov     [esp+848h+var_808], offset unk_756180
006E8222 mov     ds:pcbBuffer, ebx
006E8228 mov     ds:dword_759350, ebx
006E822E mov     ds:dword_7609B8, eax
006E8233 mov     [esp+848h+var_4], 64h ; 'd'
006E823B mov     [esp+848h+var_5], 2Eh ; '.'
006E8243 mov     [esp+848h+var_6], cl
006E824A call    sub_67DF30        ; Call Procedure
006E824F mov     al, [esp+848h+var_2]
006E8256 mov     cl, [esp+848h+var_4]
006E825D lea     edx, [esp+848h+var_834] ; Load Effective Address
006E8261 mov     [esp+848h+var_7], al
006E8268 push    edx
006E8269 mov     [esp+84Ch+var_8], cl
```


Continuing further, there is another call at 0x006E824A

West Chester University

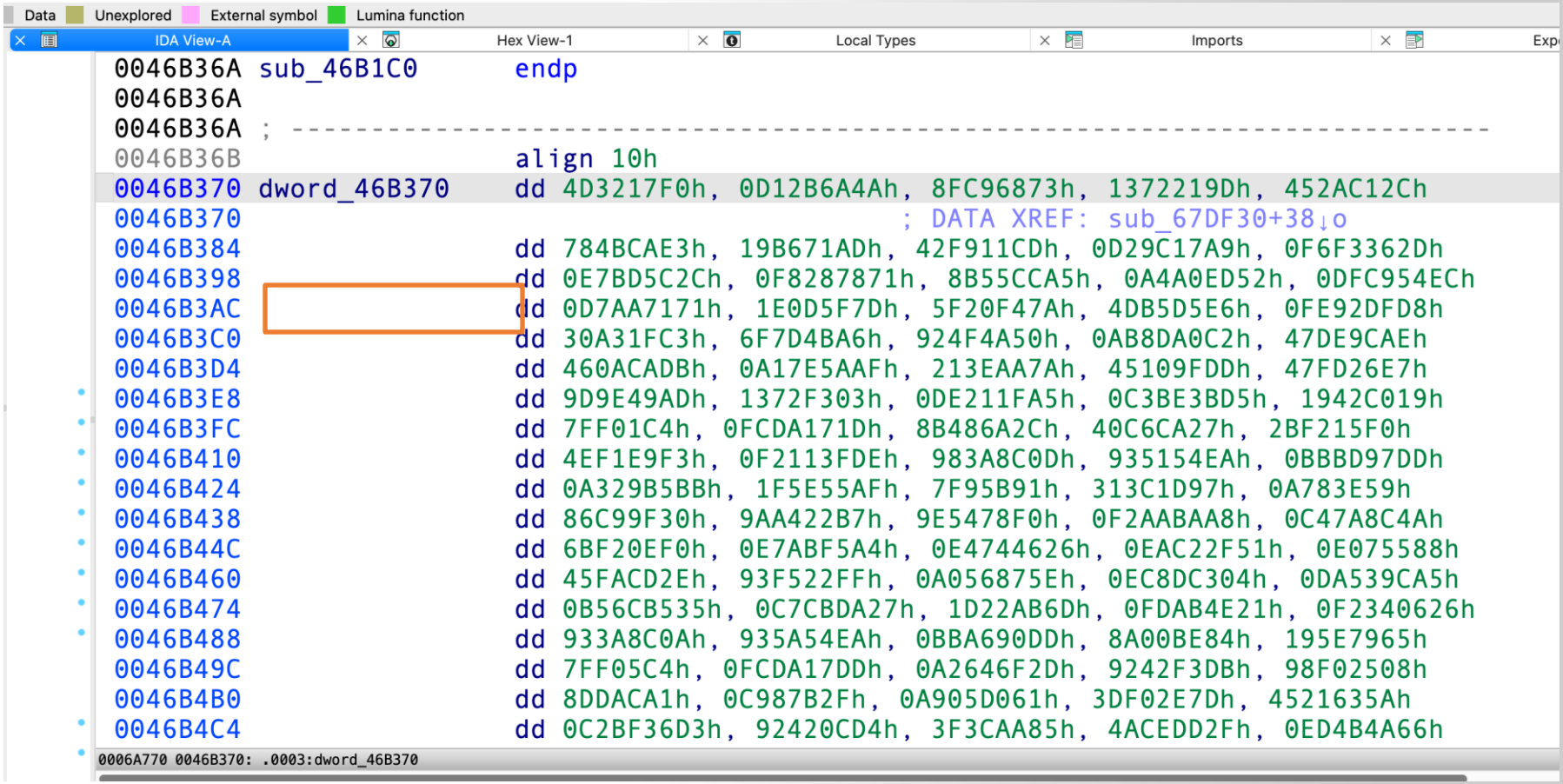
4fafbfd2e560778f11beb8f736e80bb1



```
0067DF63 mov     eax, ds:dword_760470
0067DF68 mov     ecx, offset dword_46B370
0067DF6D add     ecx, eax              ; Add
0067DF6F add     eax, ebx              ; Add
0067DF71 mov     dl, [ecx+ebx]
0067DF74 mov     cl, byte ptr [esp+10h+arg_8]
0067DF78 mov     [eax+esi], dl
0067DF7B mov     eax, ds:dword_760470
0067DF80 add     eax, ebx              ; Add
0067DF82 add     eax, esi              ; Add
0067DF84 mov     dl, [eax]
0067DF86 xor     dl, cl                ; Logical Exclusive OR
0067DF88 inc     ebx                   ; Increment by 1
0067DF89 cmp     ebx, 67Dh             ; Compare Two Operands
0067DF8F mov     [eax], dl
0067DF91 jg      short loc_67DFF7 ; Jump if Greater (ZF=0 & SF=OF)
```

Starting from 0x0067DF63, this is actually a decryption process. Why do we say that? First, at 0x0067DF68, there is a mov assignment statement, which assigns the content at address 0x0046B370 to ecx. Let's take a look at the content at this address

4fafbfd2e560778f11beb8f736e80bb1



As you can see, this is a bunch of garbled data, likely encrypted.

4fafbfd2e560778f11beb8f736e80bb1

```
0067DF63 mov     eax, ds:dword_760470
0067DF68 mov     ecx, offset dword_46B370
0067DF6D add     ecx, eax          ; Add
0067DF6F add     eax, ebx          ; Add
0067DF71 mov     dl, [ecx+ebx]
0067DF74 mov     cl, byte ptr [esp+10h+arg_8]
0067DF78 mov     [eax+esi], dl
0067DF7B mov     eax, ds:dword_760470
0067DF80 add     eax, ebx          ; Add
0067DF82 add     eax, esi          ; Add
0067DF84 mov     dl, [eax]
0067DF86 xor     dl, cl            ; Logical Exclusive OR
0067DF88 inc     ebx               ; Increment by 1
0067DF89 cmp     ebx, 67Dh         ; Compare Two Operands
0067DF8F mov     [eax], dl
0067DF91 jg      short loc_67DFF7 ; Jump if Greater (ZF=0 & SF=OF)
```

Following this, there is a series of operations, including add (addition) and xor (exclusive OR). The xor operation, in particular, is a common decryption technique often used by malicious programs. From the final inc (increment) and cmp (compare) operations, we can deduce that ebx holds the number of binary codes to be decrypted, which is 0x67D in this case.

West Chester University

Dynamic Analysis

Dynamic Analysis

- Dynamic analysis is the process of executing malware in a monitored environment to observe its behaviors.

4fafbfd2e560778f11beb8f736e80bb1 (revisited)

(Trojan.Win32.Agent.b)

Packed and Obfuscated Malware

•**Definition:**

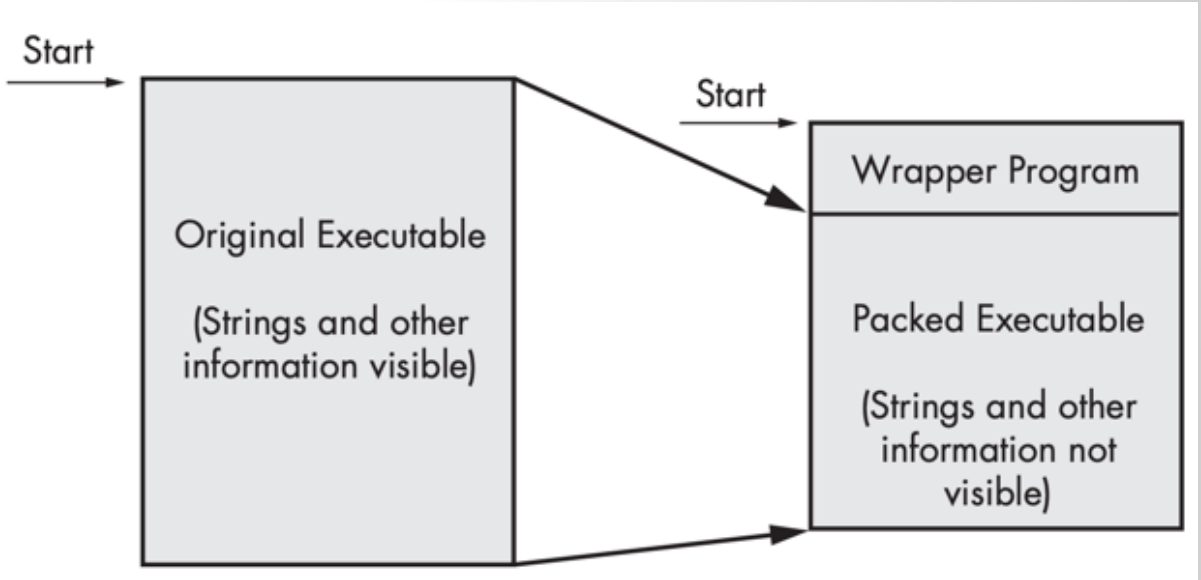
Obfuscation is a technique used by malware authors to hide malicious code.

•**How it works:**

- The PE (Portable Executable) file is encrypted and embedded within the program.
- During execution, the program decrypts and runs the hidden PE file.

•**Goal:**

To conceal the malicious payload and evade detection



How Obfuscation Works

1.Encryption:

The malicious PE file is encrypted and embedded in the program.

2.Execution:

1. The program allocates memory using functions like **VirtualAlloc**.
2. Decrypts the PE file into the allocated memory.
3. Executes the decrypted payload.

3.Result:

The malicious code runs hidden from detection tools.

West Chester University

Removing Obfuscation: Key Steps

1.Set Breakpoints:

Use a debugger (e.g., OllyDbg) to set breakpoints on memory allocation functions like VirtualAlloc.

2.Monitor Memory Allocation:

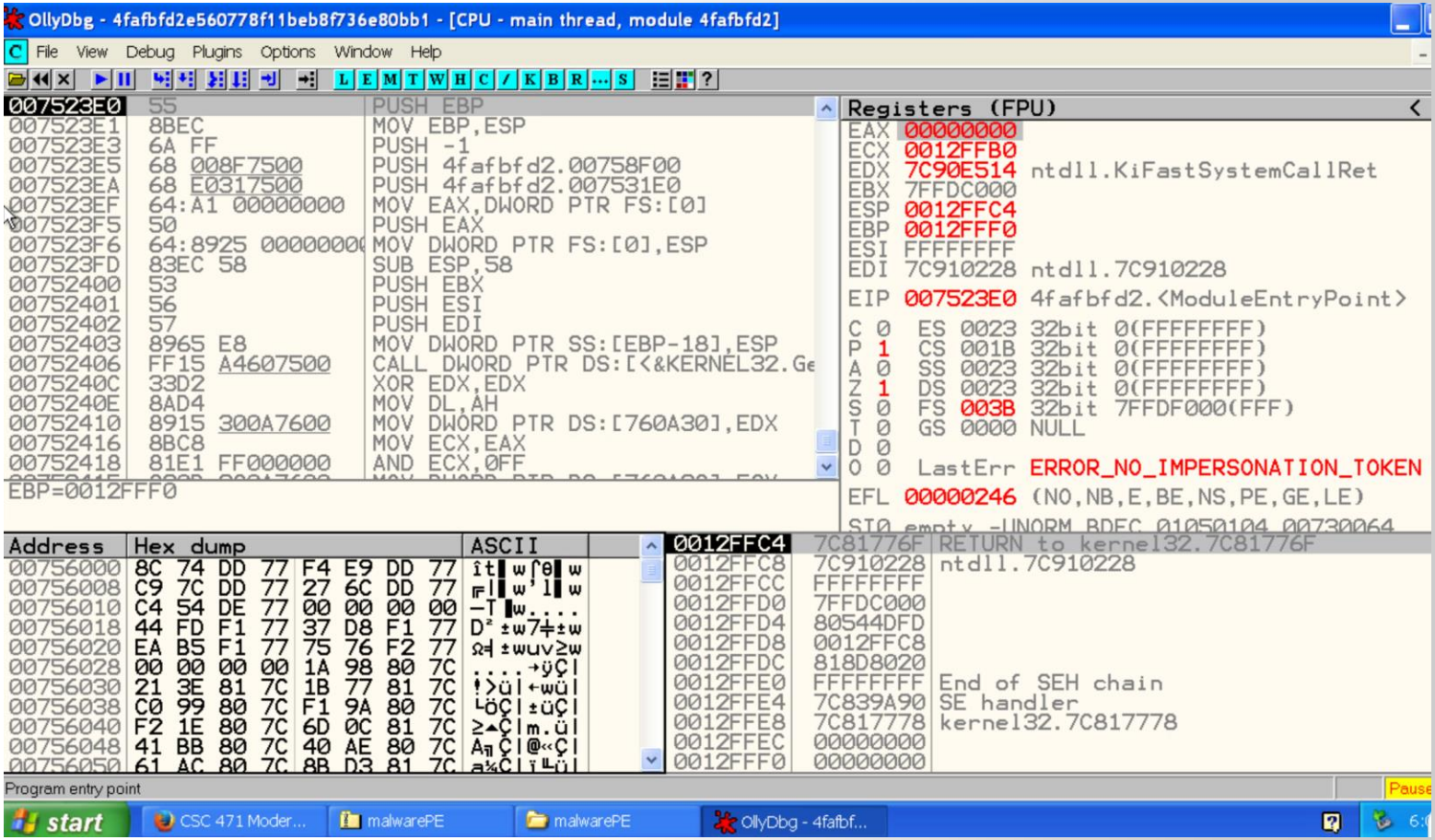
1. Track the starting address of allocated memory.
2. Set hardware breakpoints to detect writes to this memory.

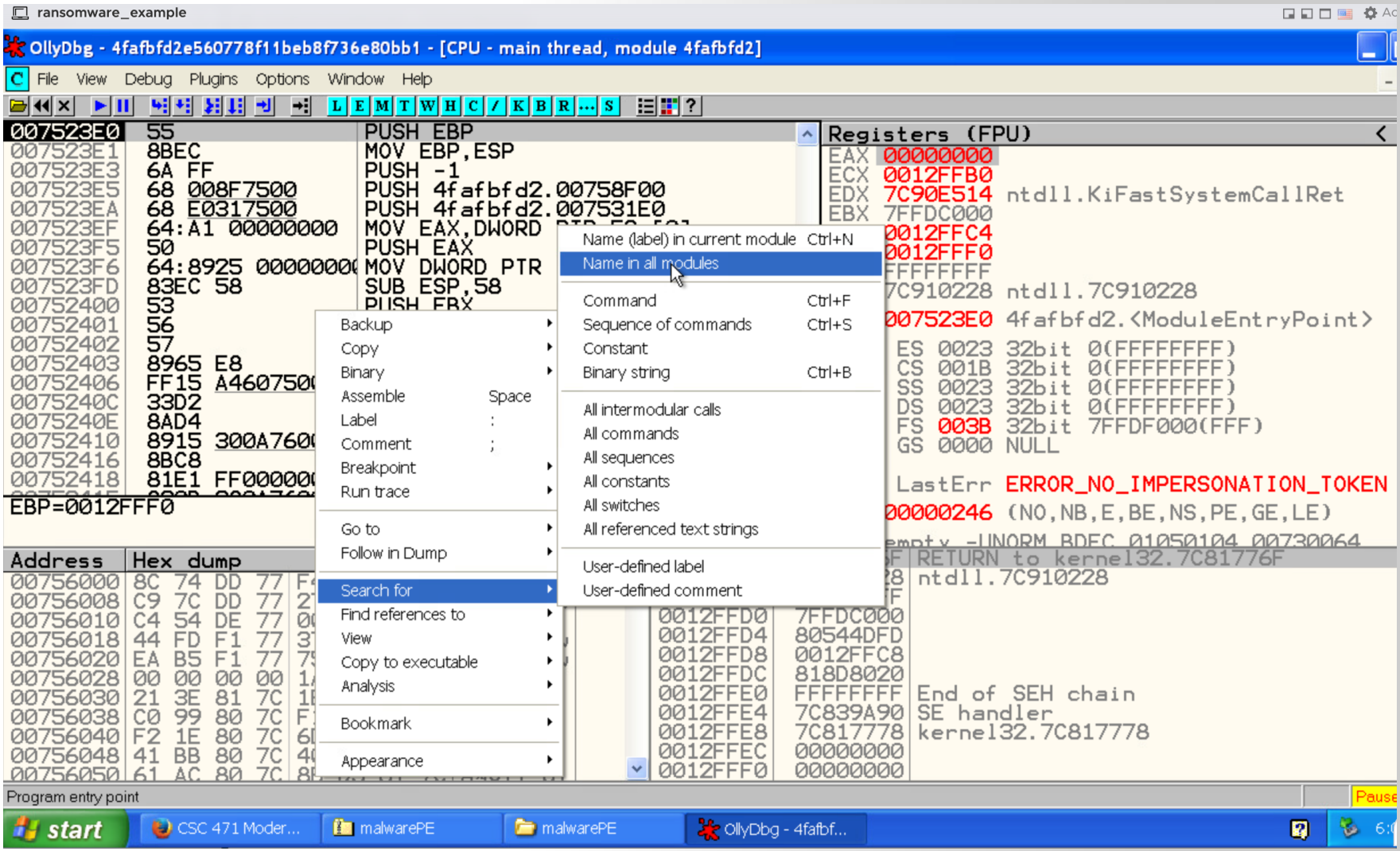
3.Analyze Decryption:

1. Observe the decryption process in memory.
2. Dump the decrypted PE file for further analysis.

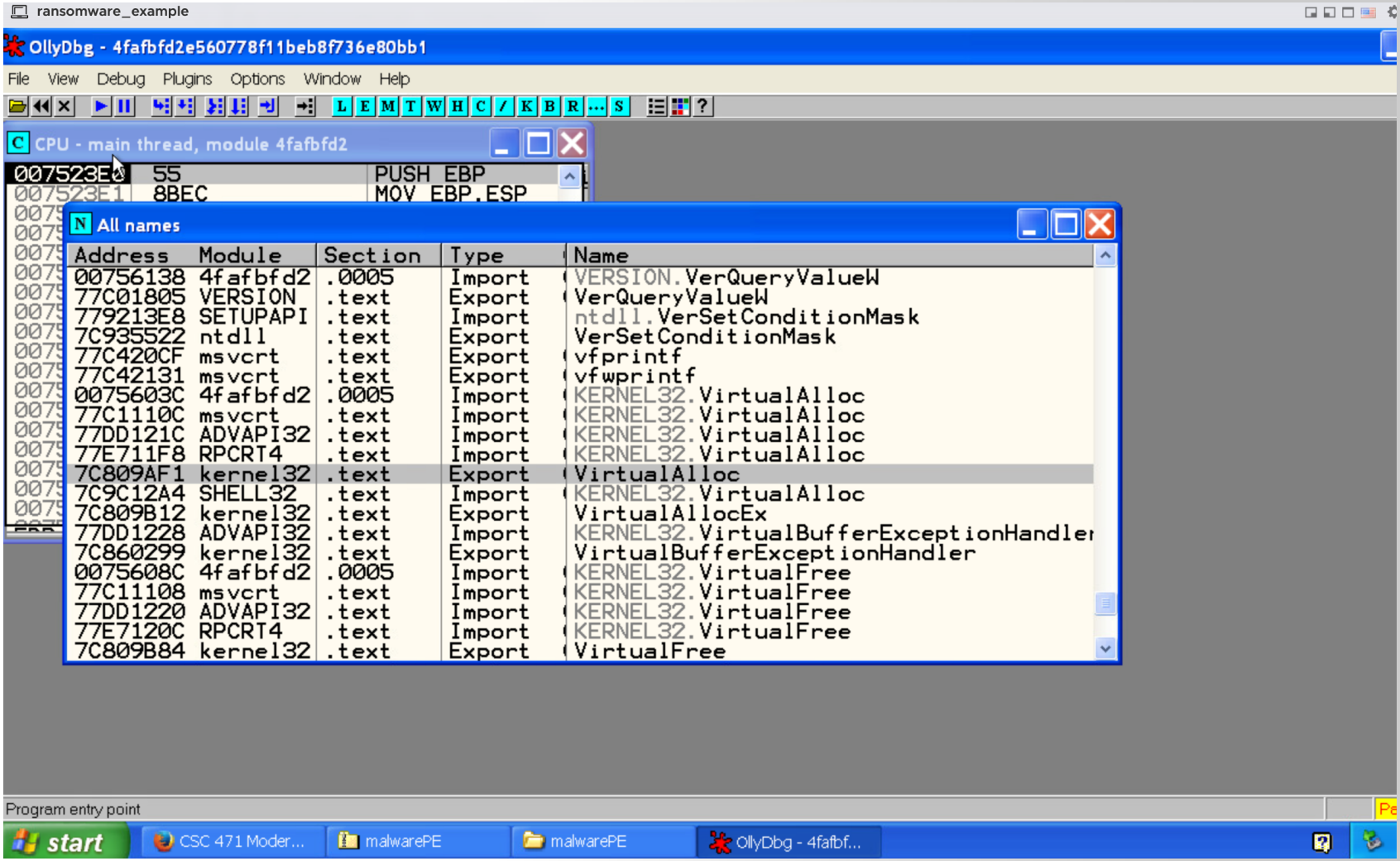
West Chester University

Load target file into OllyDBG

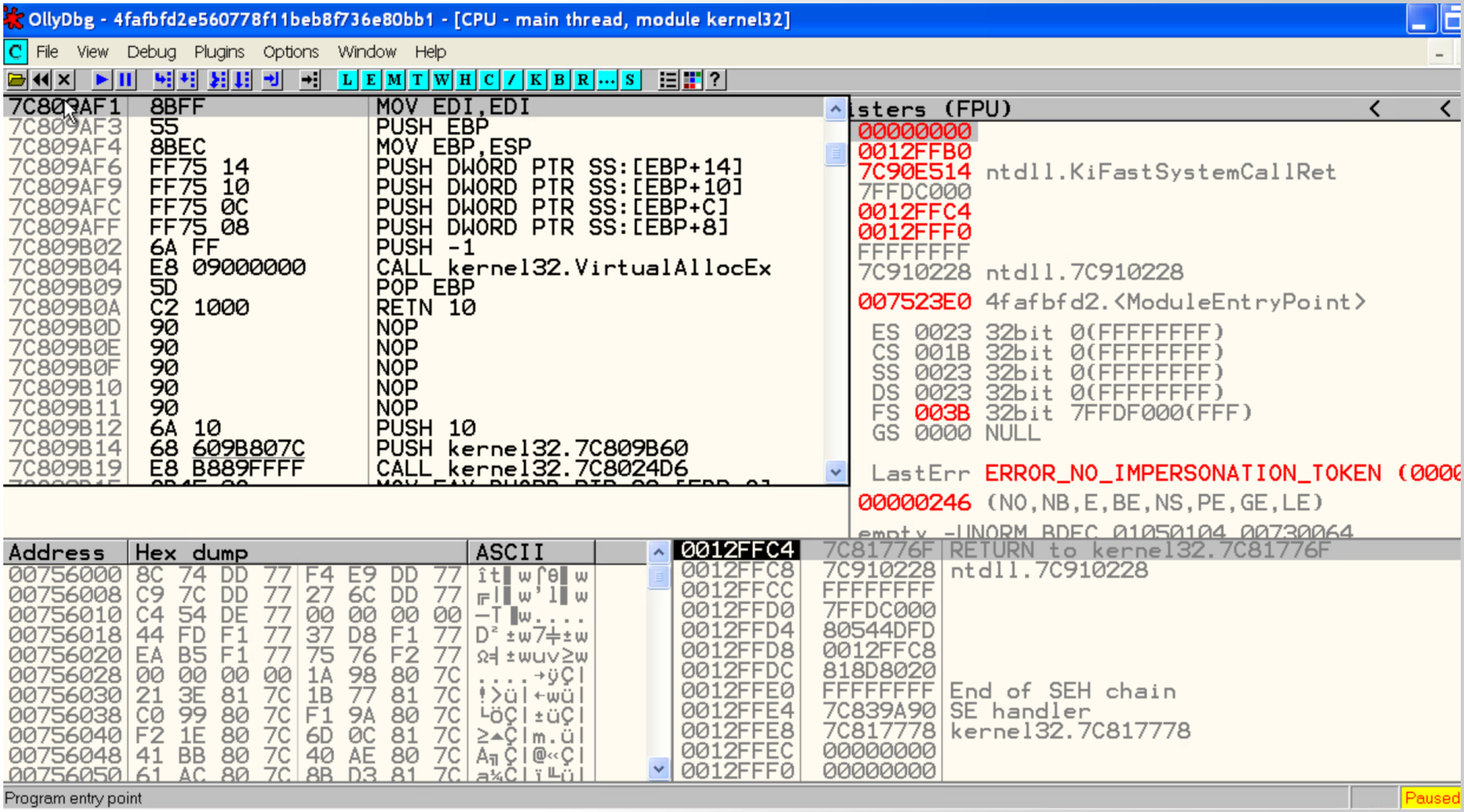




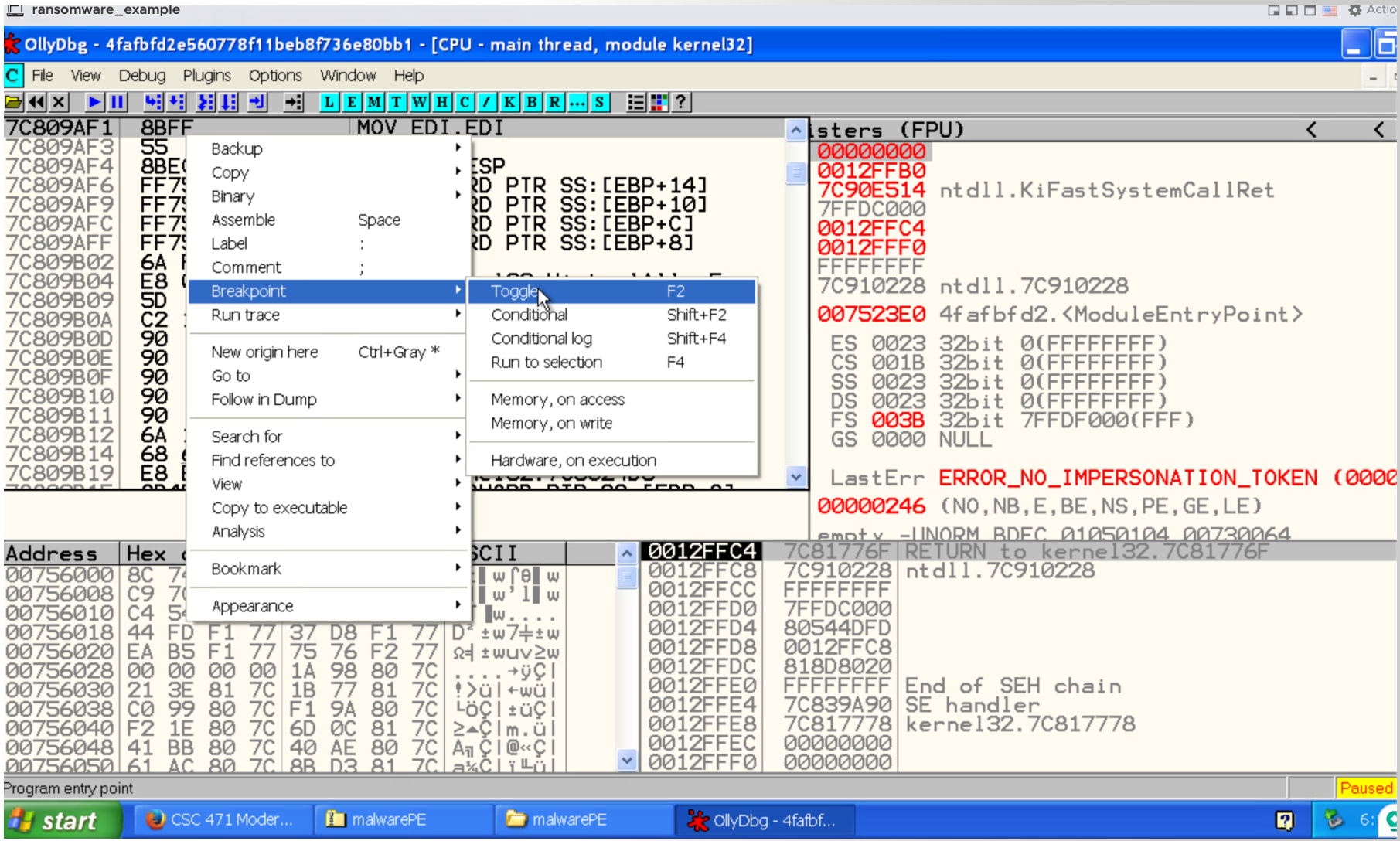
Right click → “Search for” → Name in all modules



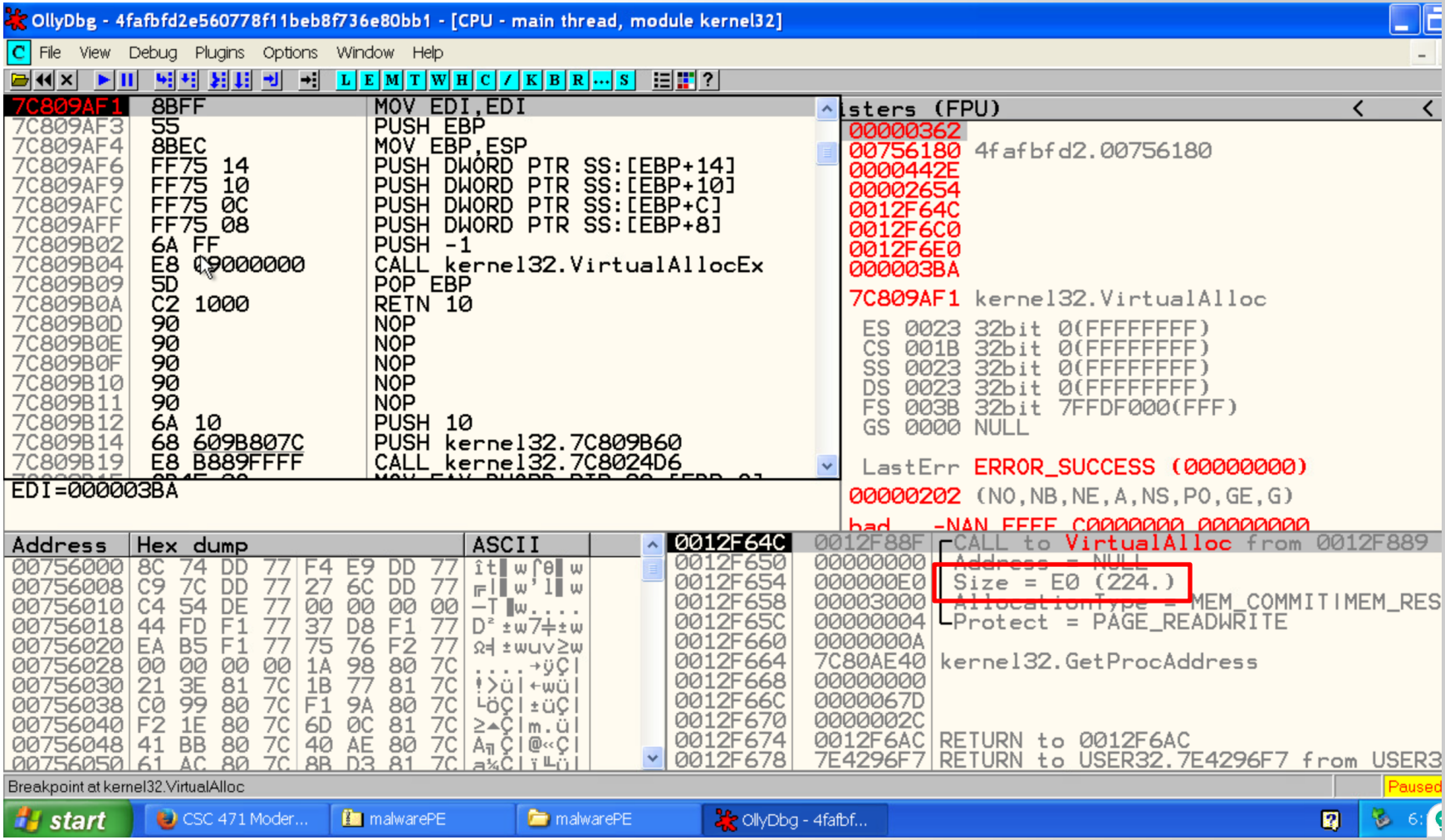
Looking for “VirtualAlloc” func → double click



Here we go into the VirtualAlloc function.

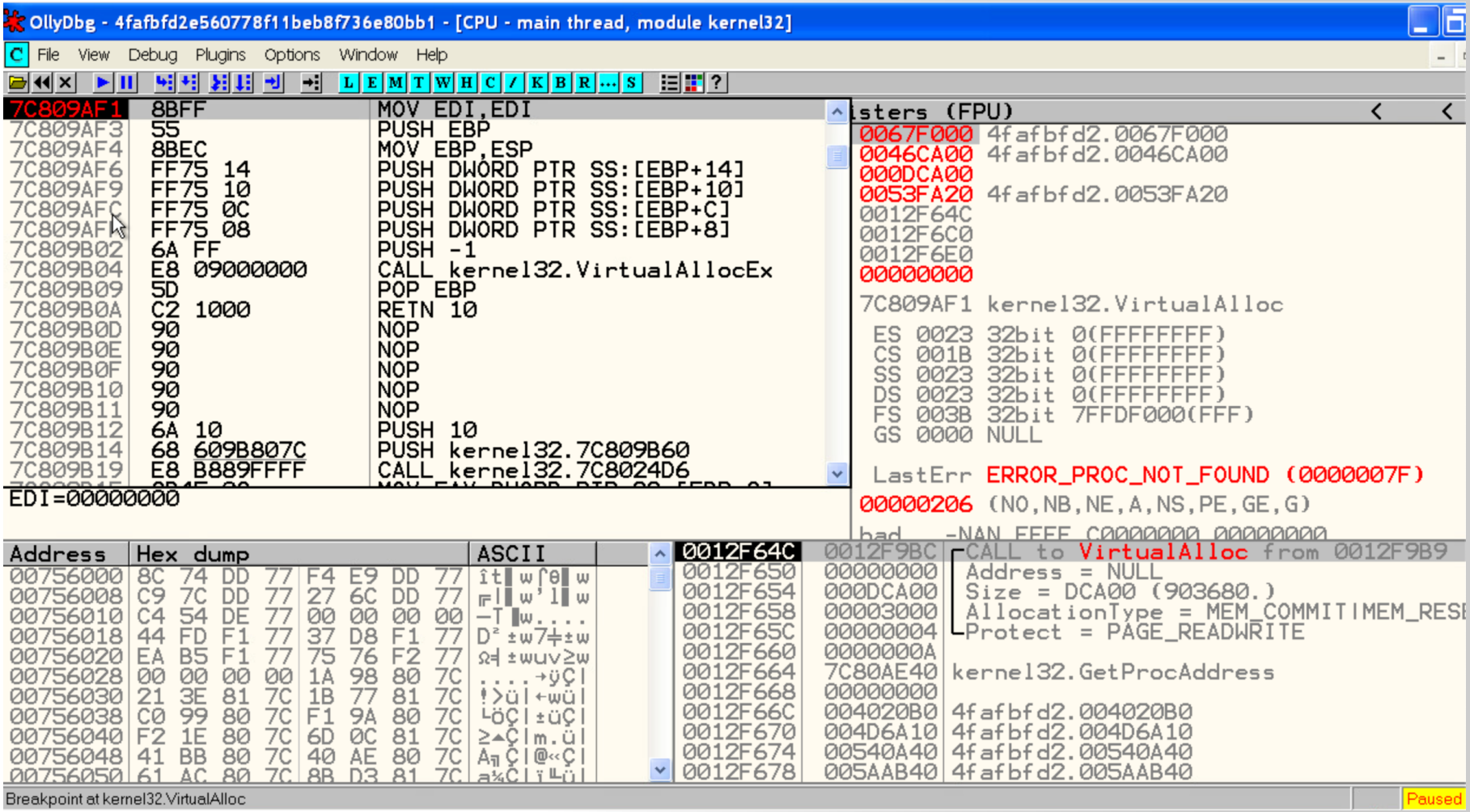


Right click the first line (MOV EDI, EDI) → Breakpoint → Toggle



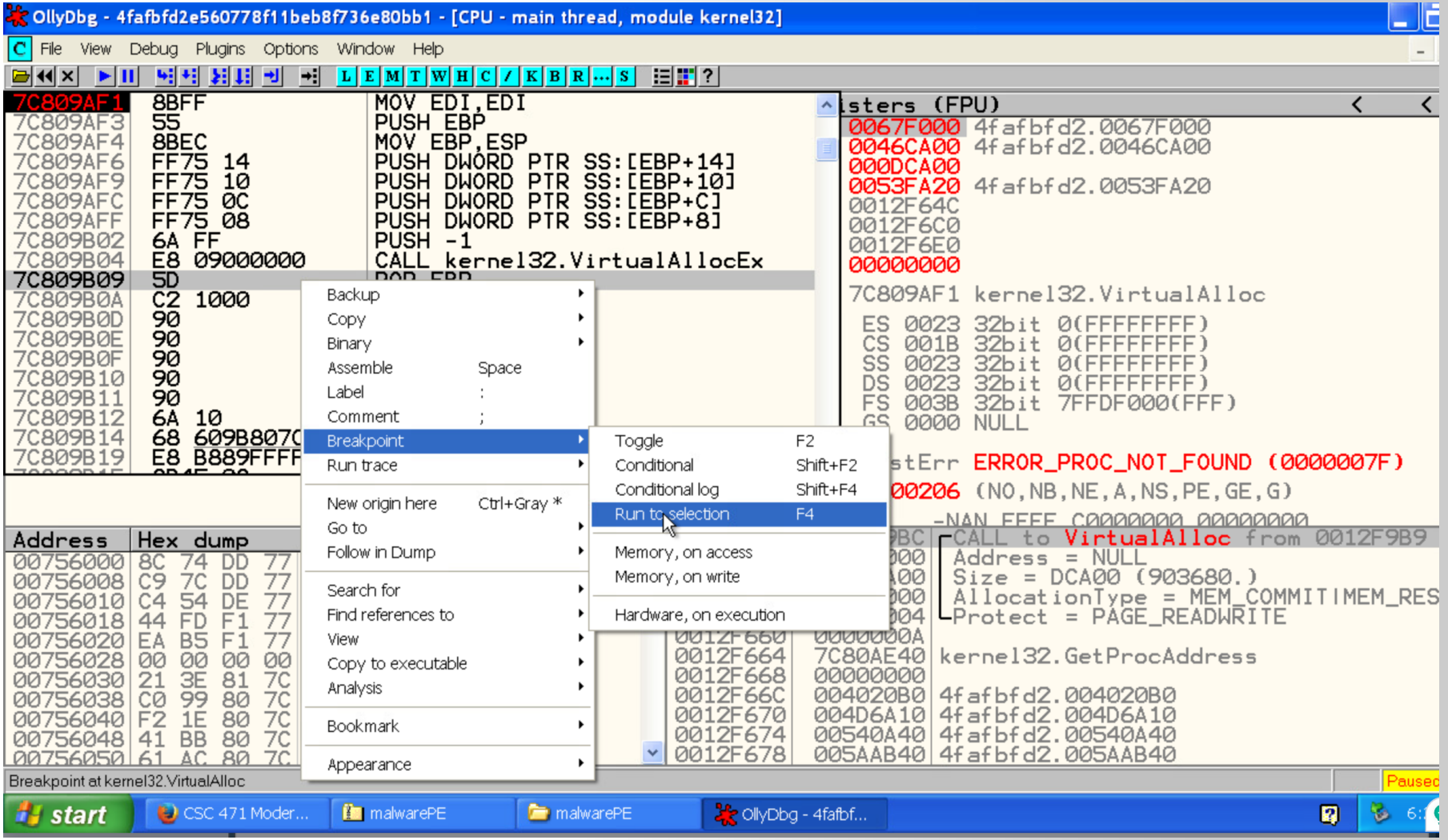
Click the “play” button (F9) and the program will run and hit the break point

Check out the allocation size → 224 Bytes → too small

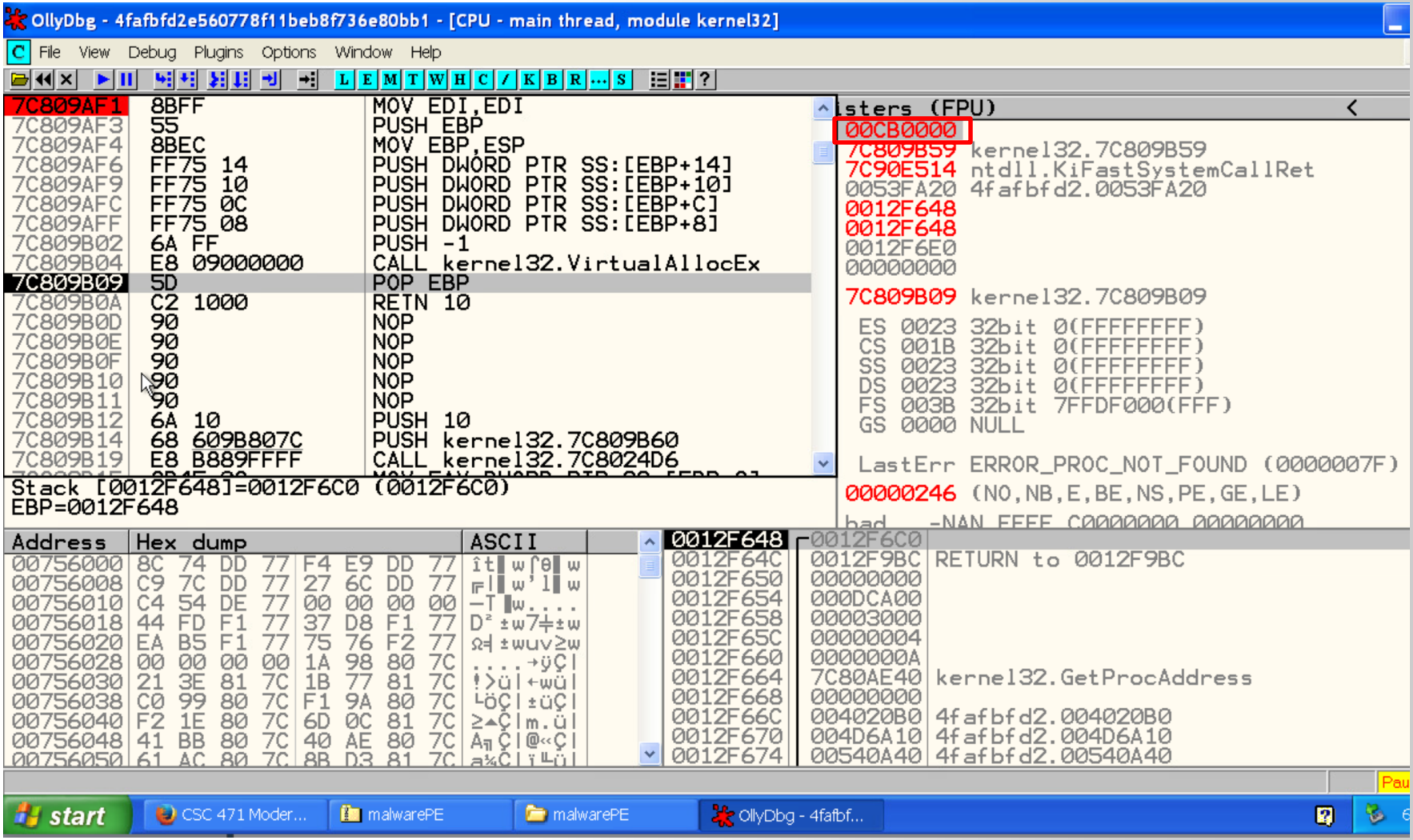


Click the “play” button (F9) again.

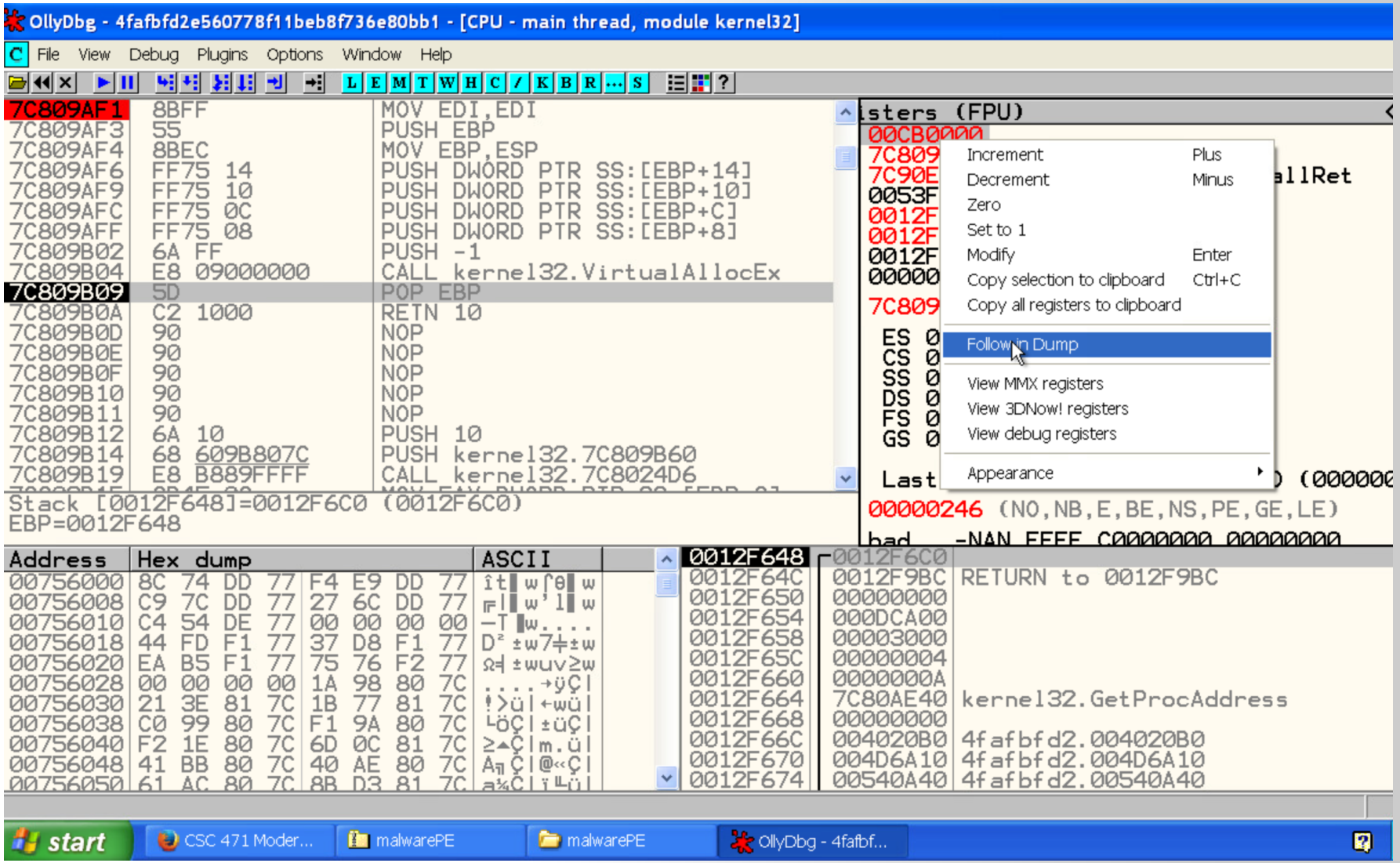
Check out the allocation size → 903680 Bytes →Good!



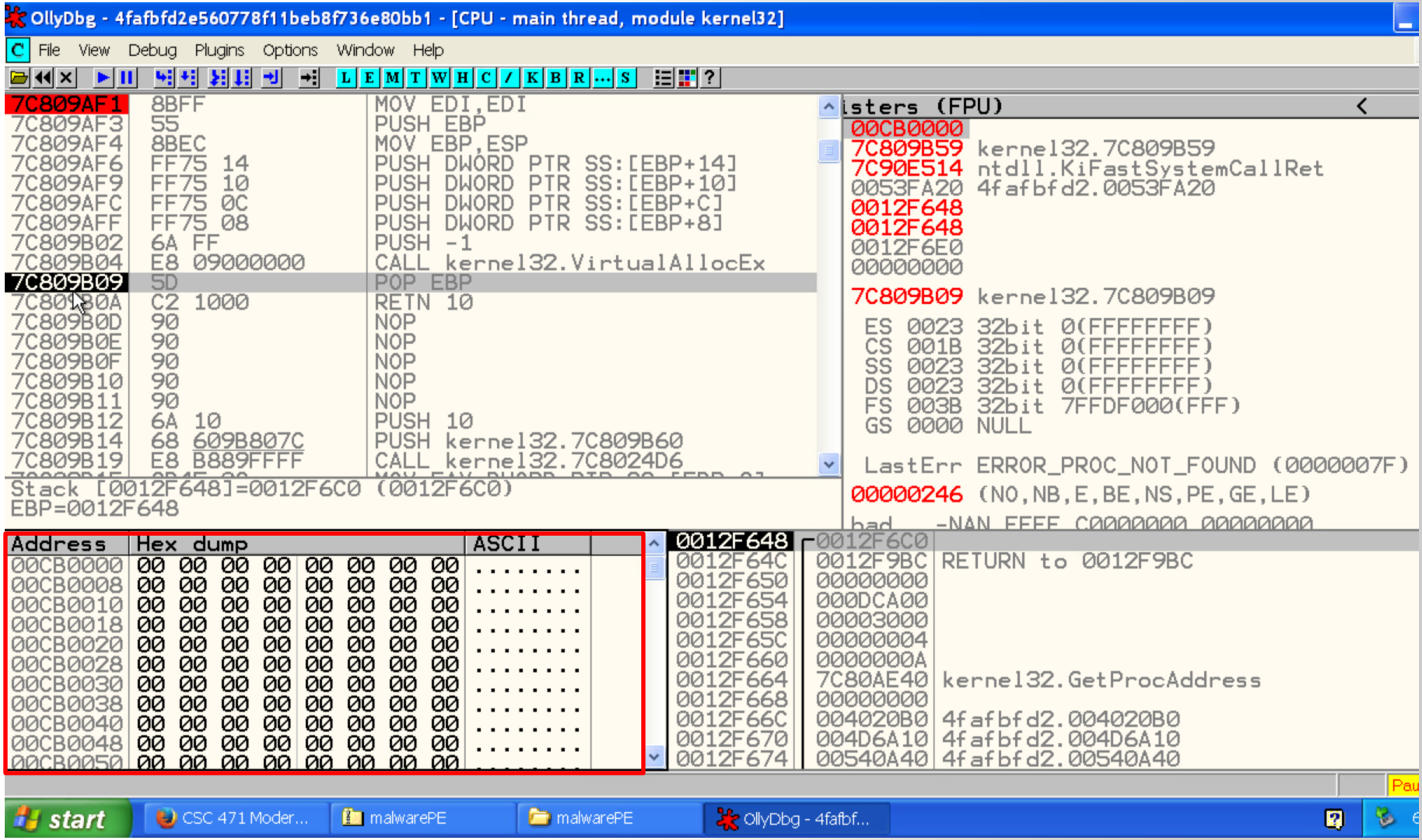
Right click ”POP EBP” link (below CALL VirtualAllocEx)
→ ”Breakpoint”→ Run to selection (F4)



Check EAX → 00CB0000

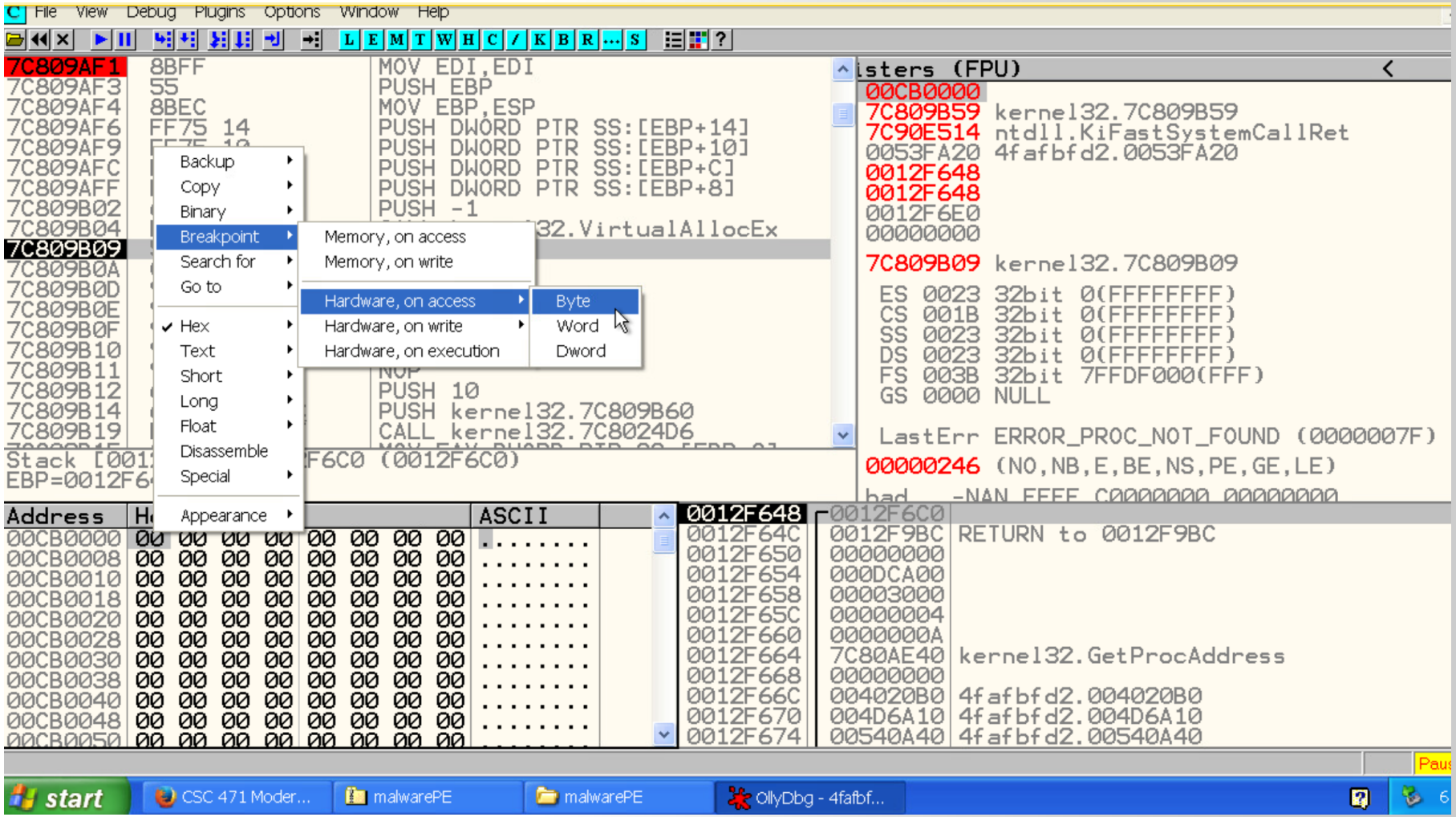


Right click EAX → “follow in Dump”

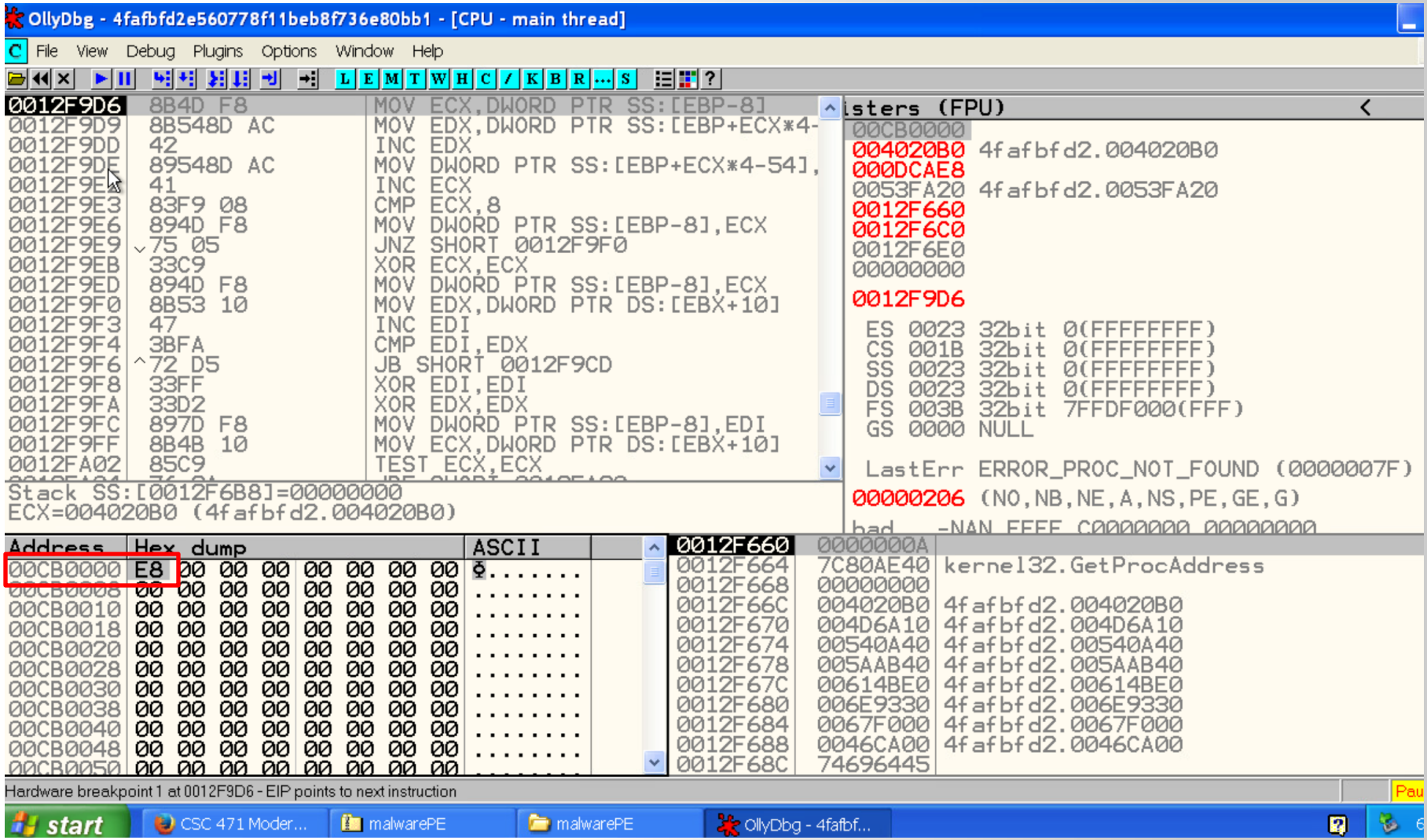


We follow this address in the data window

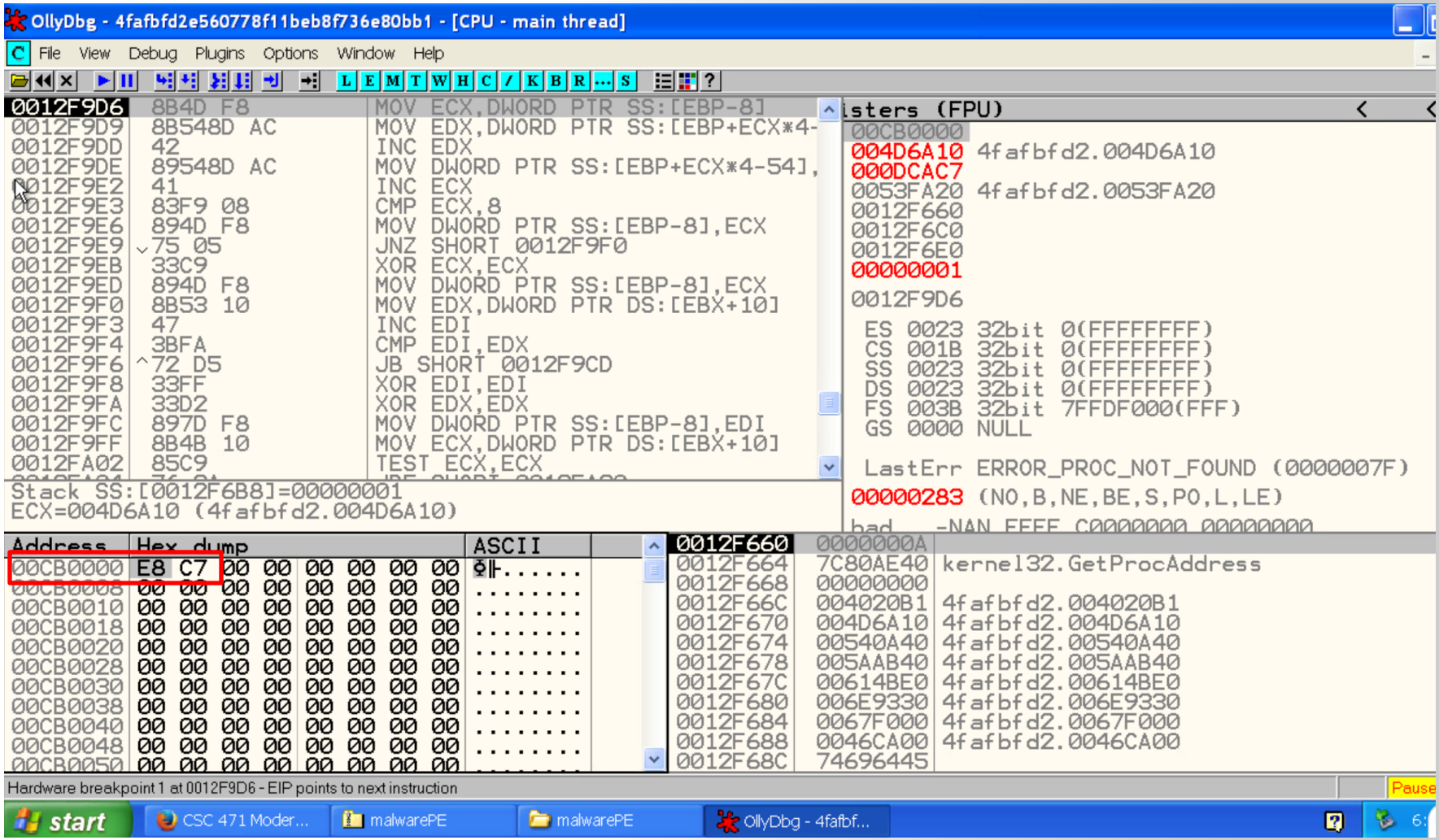
Set a hardware access breakpoint at 0x00CB0000.



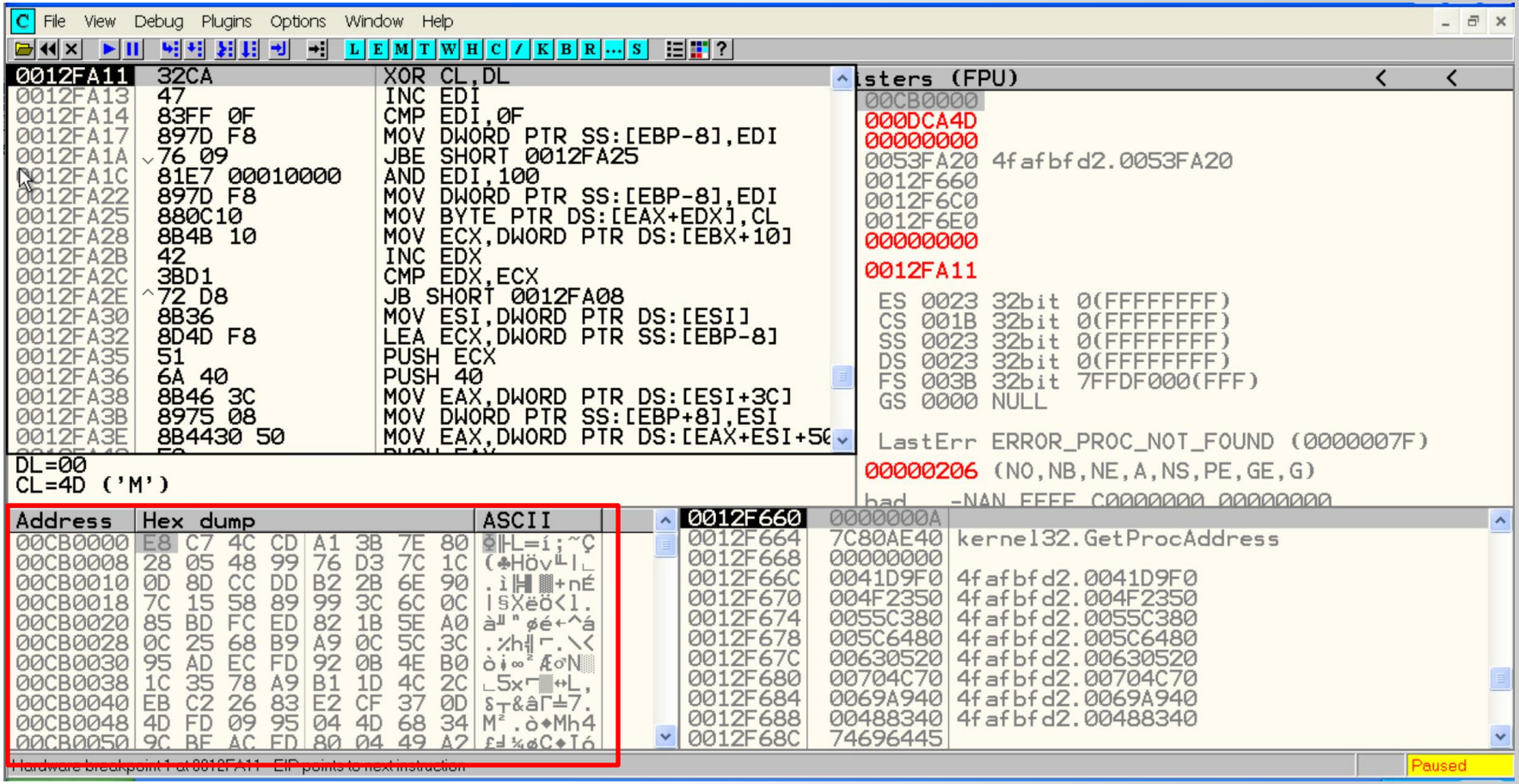
Right click the first byte at 00CB0000 → “Breakpoint” → “Hardware on access” → “Word”



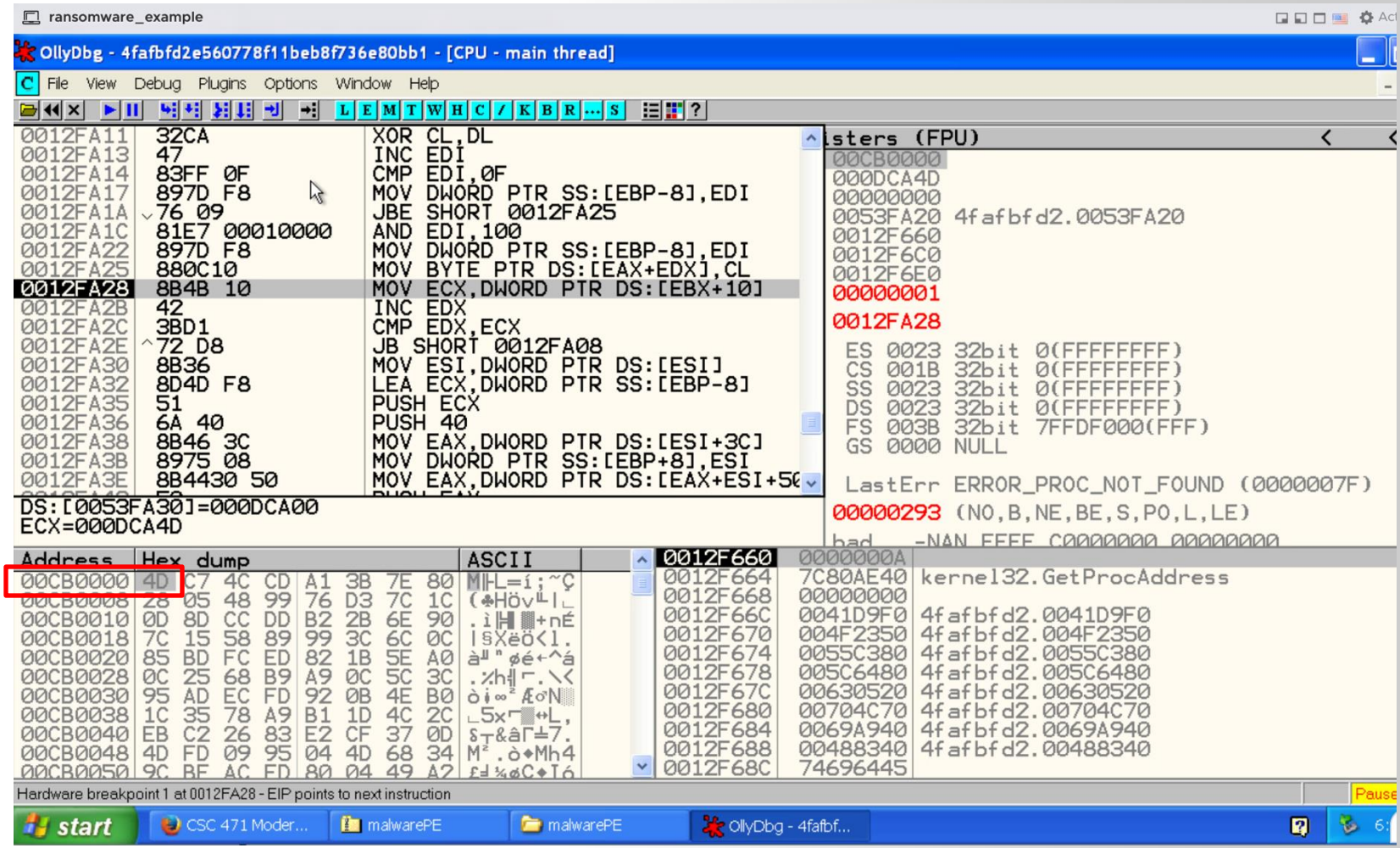
Click Play button (F9) → Check 00CB0000



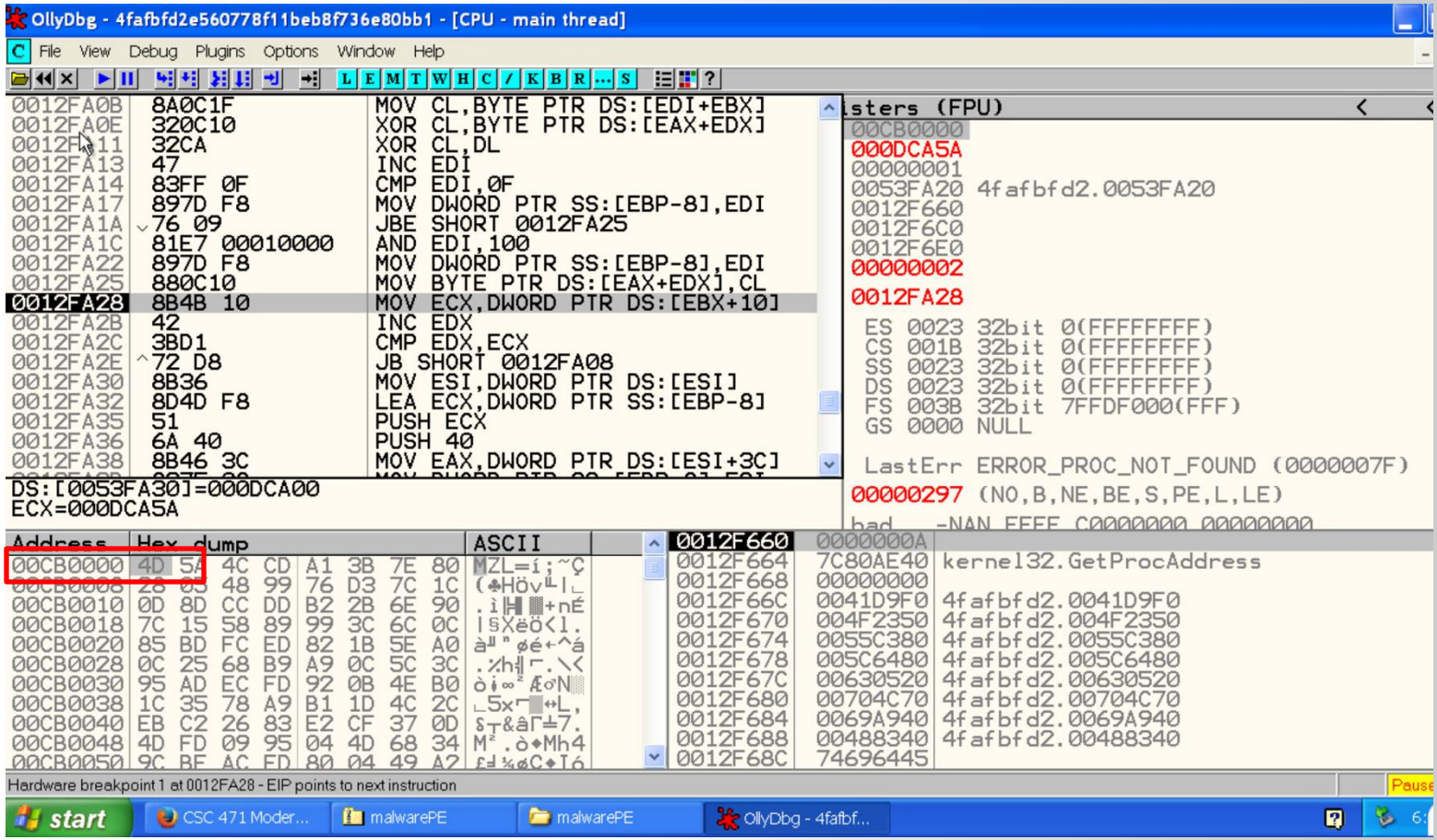
Click Play button (F9) again → Check the data window



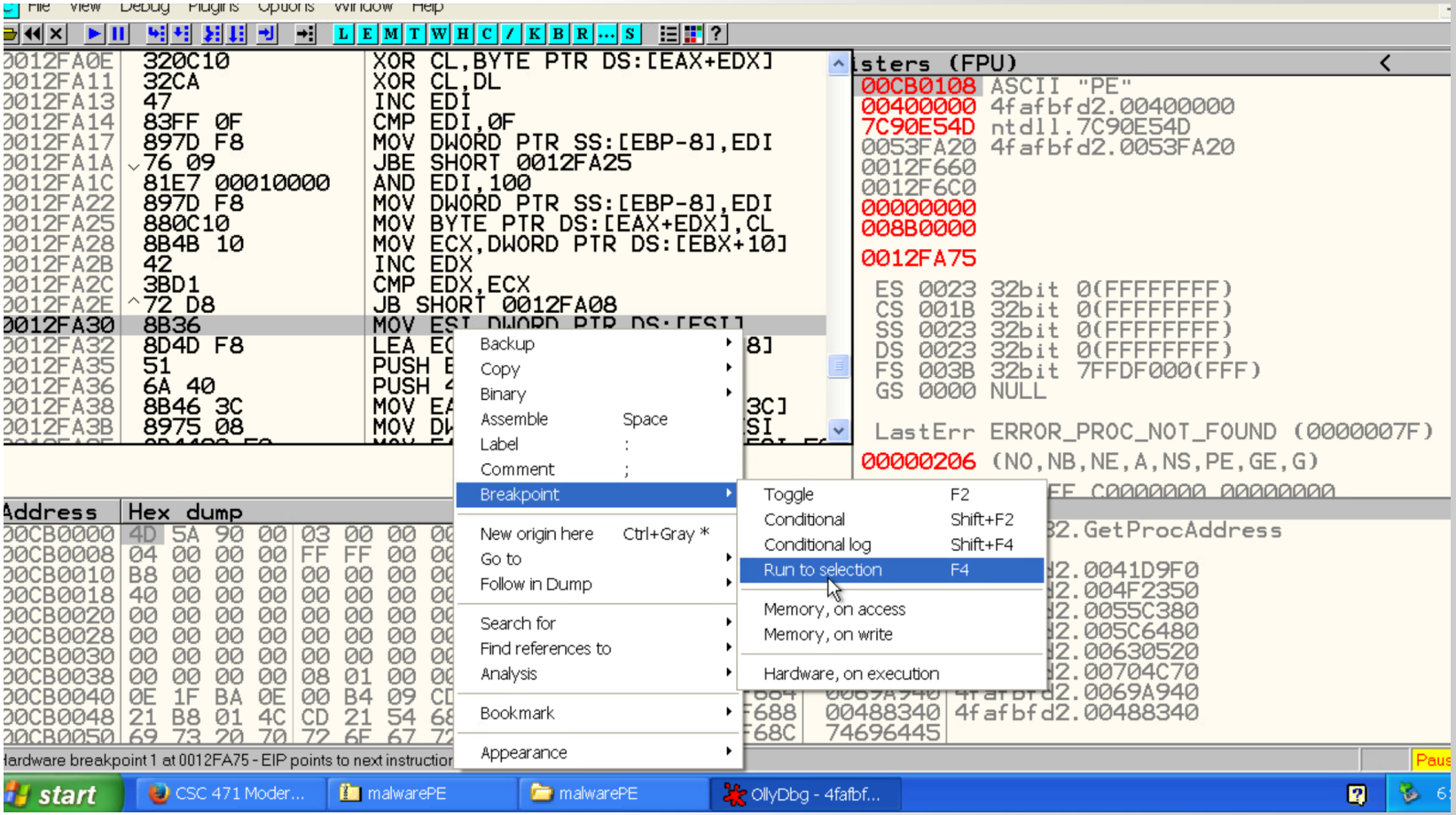
Click Play button (F9) again → Check the data window

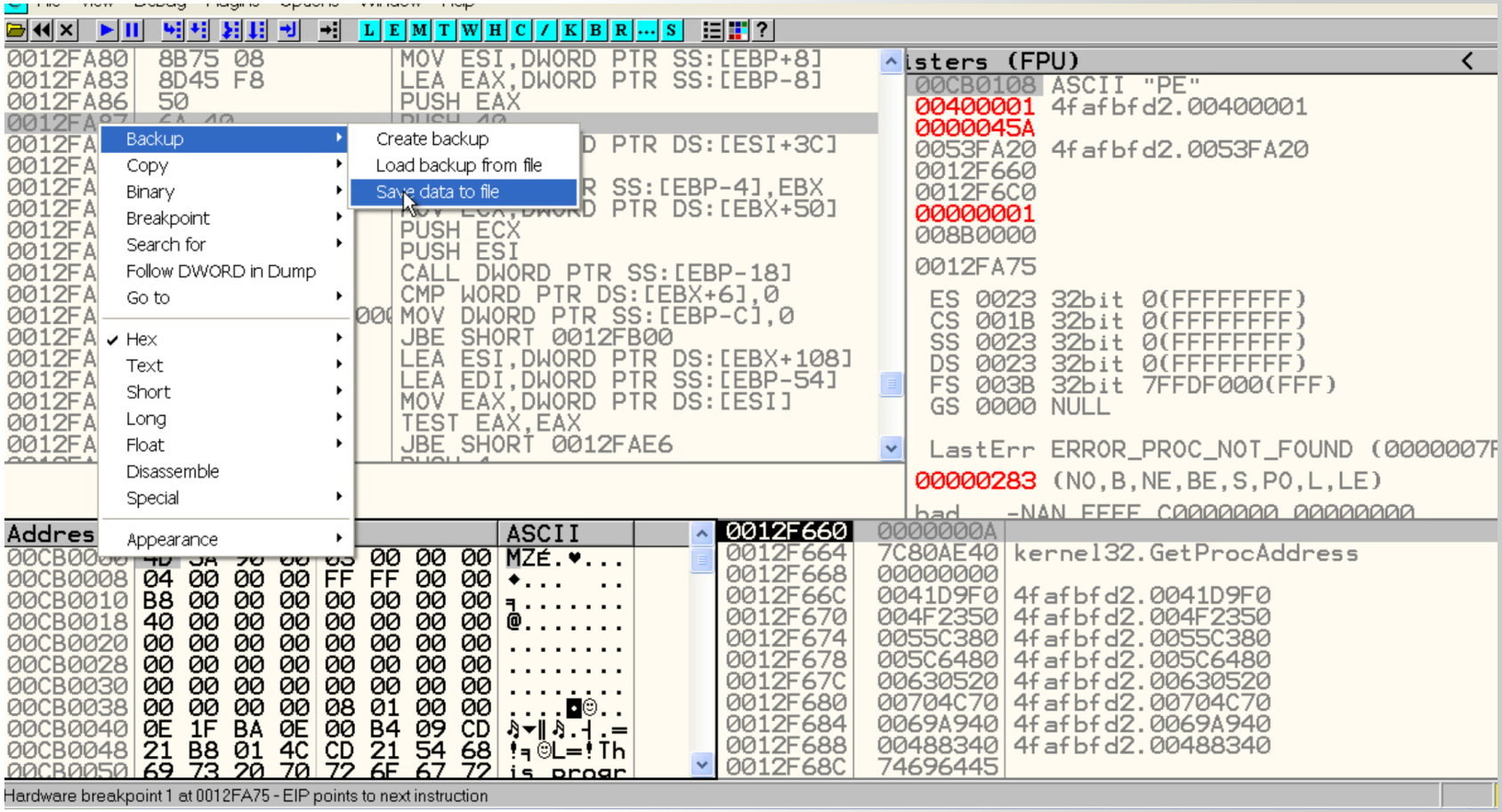


Click Play button (F9) again → Check 00CB0000



Click Play button (F9) again → Check the data window


0012FA0E 320C10 XOR CL,BYTE PTR DS:[EAX+EDX]
0012FA11 32CA XOR CL,DL
0012FA13 47 INC EDI
0012FA14 83FF 0F CMP EDI,0F
0012FA17 897D F8 MOV DWORD PTR SS:[EBP-8],EDI
0012FA1A 76 09 JBE SHORT 0012FA25
0012FA1C 81E7 00010000 AND EDI,100
0012FA22 897D F8 MOV DWORD PTR SS:[EBP-8],EDI
0012FA25 880C10 MOV BYTE PTR DS:[EAX+EDX],CL
0012FA28 8B4B 10 MOV ECX,DWORD PTR DS:[EBX+10]
0012FA2B 42 INC EDX
0012FA2C 3BD1 CMP EDX,ECX
0012FA2E 72 D8 JB SHORT 0012FA08
0012FA30 8B36 MOV ESI,DWORD PTR DS:[ESI]
0012FA32 8D4D F8 LEA EC
0012FA35 51 PUSH E
0012FA36 6A 40 PUSH 4
0012FA38 8B46 3C MOV EA
0012FA3B 8975 08 MOV DW
isters (FPU)
00CB0108 ASCII "PE"
00400000 4fafbfd2.00400000
7C90E54D ntdll.7C90E54D
0053FA20 4fafbfd2.0053FA20
0012F660
0012F6C0
00000000
008B0000
0012FA75
ES 0023 32bit 0(FFFFFFFF)
CS 001B 32bit 0(FFFFFFFF)
SS 0023 32bit 0(FFFFFFFF)
DS 0023 32bit 0(FFFFFFFF)
FS 003B 32bit 7FFDF000(FFF)
GS 0000 NULL
LastErr ERROR_PROC_NOT_FOUND (0000007F)
00000206 (NO,NB,NE,A,NS,PE,GE,G)
Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Breakpoint
New origin here Ctrl+Gray *
Go to
Follow in Dump
Search for
Find references to
Analysis
Bookmark
Appearance
Toggle F2
Conditional Shift+F2
Conditional log Shift+F4
Run to selection F4
Memory, on access
Memory, on write
Hardware, on execution
Address Hex dump
00CB0000 4D 5A 90 00 03 00 00 00
00CB0008 04 00 00 00 FF FF 00 00
00CB0010 B8 00 00 00 00 00 00 00
00CB0018 40 00 00 00 00 00 00 00
00CB0020 00 00 00 00 00 00 00 00
00CB0028 00 00 00 00 00 00 00 00
00CB0030 00 00 00 00 00 00 00 00
00CB0038 00 00 00 00 08 01 00 00
00CB0040 0E 1F BA 0E 00 B4 09 CD
00CB0048 21 B8 01 4C CD 21 54 68
00CB0050 69 73 20 70 72 6F 67 72
32.GetProcAddress
2.0041D9F0
2.004F2350
2.0055C380
2.005C6480
2.00630520
2.00704C70
2.0069A940
F688 00488340 4fafbfd2.00488340
F68C 74696445
Hardware breakpoint 1 at 0012FA75 - EIP points to next instruction
start
CSC 471 Moder...
malwarePE
malwarePE
OllyDbg - 4fafbf...


Backup
Copy
Binary
Breakpoint
Search for
Follow DWORD in Dump
Go to
Hex
Text
Short
Long
Float
Disassemble
Special
Appearance
Create backup
Load backup from file
Save data to file
Hardware breakpoint 1 at 0012FA75 - EIP points to next instruction

Using OllyDbg to Remove Obfuscation

1.Set Breakpoint on VirtualAlloc:

1. Run the program and pause at VirtualAlloc.
2. Monitor the return value (memory address) and allocation size.

2.Identify Large Allocations:

1. Focus on large memory allocations (e.g., > 900,000 bytes) typical for PE files.

3.Set Hardware Breakpoint:

1. Set a hardware breakpoint at the allocated memory address.
2. Detect when data is written to this memory.

West Chester University

Q & A