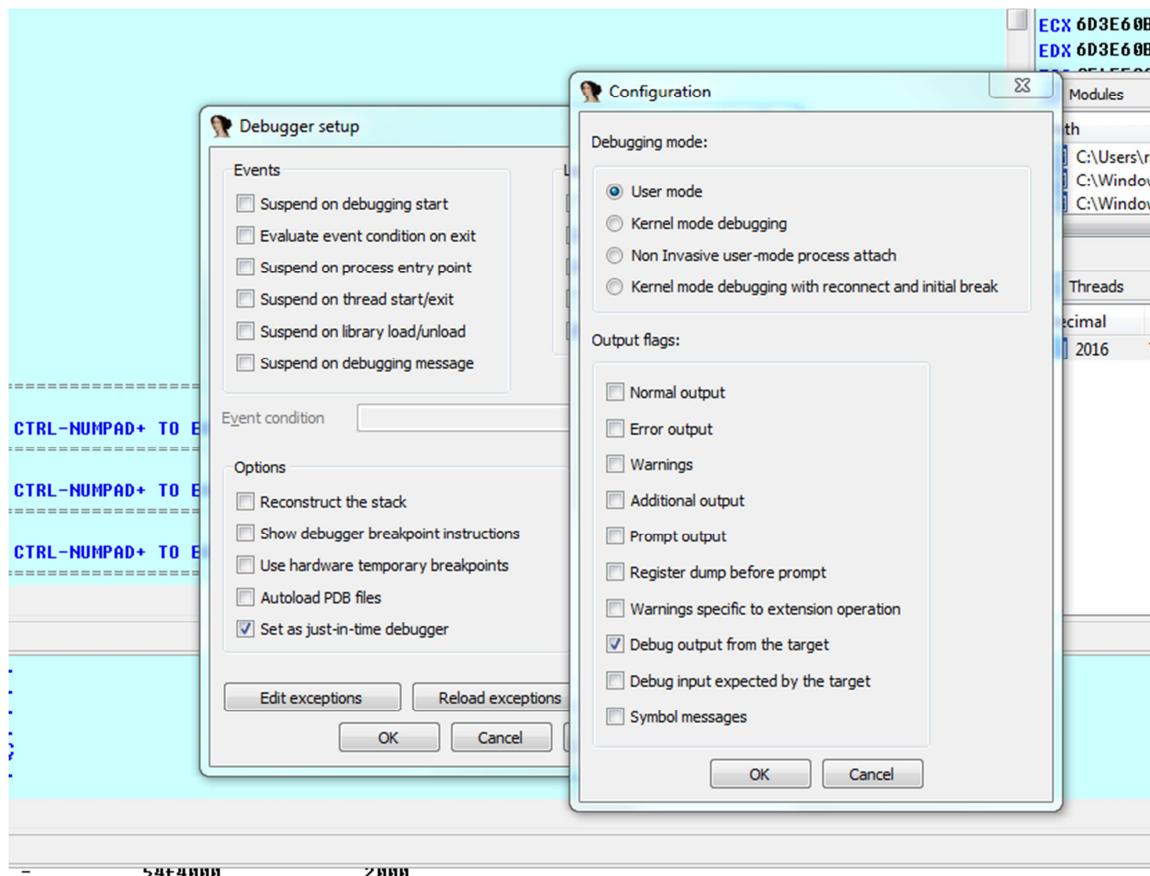


## REVERSING WITH IDA PRO FROM SCRATCH

### PART 44

We have the exercise PRACTICA\_44 to solve it, but we will do it in the next parts. Now, we will see some more info that we can get using Windbg inside IDA. We'll use the previous case of PRACTICA 41 b that we know it was a heap overflow.

We'll change from IDA debugger to Windbg and check that it is in USER mode in DEBUGGER OPTIONS.



Change the gflags to have the PAGE HEAP enabled in the process in Full Mode.

```
practica41b.exe: page heap disabled
C:\Program Files (x86)\Windows Kits\10\Debuggers\x86>gflags.exe -p /enable PRACTICA41b.exe /full
Warning: pageheap.exe is running inside WOW64.
This scenario can be used to test x86 binaries (running inside WOW64)
but not native (IA64) binaries.
path: SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
practica41b.exe: page heap enabled
C:\Program Files (x86)\Windows Kits\10\Debuggers\x86>
```

Voy hasta el path donde está el gflags.exe en la misma carpeta que esta el windbg.exe y cambio a que este habilitado el PAGE HEAP en modo FULL con.

```
gflags.exe -p /enable PRACTICA41b.exe /full
```

I go to gflags.exe path in the same windbg.exe folder and change it to enable the PAGE HEAP in FULL mode with:

**gflags.exe -p /enable PRACTICA41b.exe /full**

Run script2.

```
script.py x script2.py x
1  from os import *
2  import struct
3
4
5  stdin,stdout = popen4(r'PRACTICA41b.exe -1')
6  print "ATACHEA EL DEBUGGER Y APRETA ENTER\n"
7  raw_input()
8
9  #rop_chain = create_rop_chain()
10
11 fruta="A" * 1000 + "\n"
12
13 print stdin
14
15
16 print "Escribe: " + fruta
17 stdin.write(fruta)
18 print stdout.read(40)
19
20
21 |
```

But, this time, when it stops, I attach IDA with the PRACTICA41b analysis loaded in the LOADER and, of course, in Debugger Mode Windbg user.

Logically, it will crash like before when it tries to write outside the allocated block and it overflows.

```

IDA View-EP
[...]
ucrtbase!003999356 cmp    eax, 0Bh
ucrtbase!00399935F jc     short loc_00399935C
ucrtbase!003999241 cmp    eax, 0FFFFFFFFFFh
ucrtbase!003999344 jz     short loc_00399935C
[...]
ucrtbase!00399934D mov    [esi], al
ucrtbase!00399934E inc    esi
ucrtbase!00399934F mov    [esp-40h], esi
ucrtbase!00399924C push   offset ucrtbase__iob
ucrtbase!003999351 call   near ptr ucrtbase__fgetc_nolock
ucrtbase!003999356 pop    ecx
ucrtbase!003999357 mov    [ebp-20h], eax
ucrtbase!003999358 jmp    short loc_00399933C
ucrtbase!00399935C ;
ucrtbase!00399935D ;
ucrtbase!00399935E loc_00399935C: ; CODE XREF: ucrtbase!ucrtbase_fwrite+EFJ
ucrtbase!00399935F mov    byte ptr [esi], 0
ucrtbase!00399935F jmp    short loc_003999305
ucrtbase!00399935F ;
ucrtbase!003999361 db    89h ; 
ucrtbase!003999362 db    5Dh ; ]
ucrtbase!003999363 db    004h ; E
ucrtbase!003999364 db    0Bh ; 
[...]

```

For this, you have to set Windbg well inside IDA. Anyways, if you had problems to install Windbg and IDA doesn't recognize it, you can attach Windbg outside IDA and type the commands there. You won't have the IDA interface, but you'll have the same info.

Mark Russinovich and David Solomon. (These resources may not be available in some languages and countries.)

#### Remarks

This extension command can be used to perform a variety of tasks.

The standard **Iheap** command is used to display heap information for the current process. (This should be used only for user-mode processes. The **Ipool** extension command should be used for system processes.)

The **Iheap -b** and **Iheap -B** commands are used to create and delete conditional breakpoints in the heap manager.

The **Iheap -l** command detects leaked heap blocks. It uses a garbage collector algorithm to detect all busy blocks from the heaps that are not referenced anywhere in the process address space. For huge applications, it can take a few minutes to complete. This command is only available in Windows XP and later versions of Windows.

The **Iheap -x** command searches for a heap block containing a given address. If the **-v** option is used, this command will additionally search the entire virtual memory space of the current process for pointers to this heap block. This command is only available in Windows XP and later versions of Windows.

The **Iheap -p** command displays various forms of page heap information. Before using **Iheap -p**, you must enable the page heap for the target process. This is done through the Global Flags (**gflags.exe**) utility. To do this, start the utility, fill in the name of the target application in the **Image File Name** text box, select **Image File Options** and **Enable page heap**, and click **Apply**. Alternatively, you can start the Global Flags utility from a Command Prompt window by typing **gflags /I xxx.exe +hpa**, where **xxx.exe** is the name of the target application.

The **Iheap -p -t[c|s]** commands are not supported beyond Windows XP. Use the **UMDH** tool provided with the debugger package to obtain similar results.

The **Iheap -srch** command displays those heap entries that contain a certain specified pattern.

The **Iheap -fit** command limits the display to only heap allocations of a specified size.

The **Iheap -stat** command displays heap usage statistics.

Here is an example of the standard **Iheap** command:

Windbg has a lot of useful heap commands. I think that to work with heaps it is the most complete.

```

WINDBG>heap -p -a esi
address 054F5000 found in
_DPH_HEAP_ROOT @ 3ce1000
in busy allocation ( DPH_HEAP_BLOCK:      UserAddr
54d16b4:          54f4f90
UserSize - 6c -
VirtAddr 54f4000
VirtSize) 2000
59e58e89 verifier!AVrfDebugPageHeapAllocate+0x00000229
77681d4e ntdll!RtlDebugAllocateHeap+0x00000030
775eb586 ntdll!RtlpAllocateHeap+0x000000c4
77593541 ntdll!RtlAllocateHeap+0x0000023a
6d345e7b ucrtbase!malloc+0x00000029
002411ac PRACTICA41b+0x000011ac
0024109d PRACTICA41b+0x0000109d
0024136a PRACTICA41b+0x0000136a
75a2338a kernel32!BaseThreadInitThunk+0x0000000e
77599a02 ntdll!_RtlUserThreadStart+0x00000070
775999d5 ntdll!_RtlUserThreadStart+0x0000001b

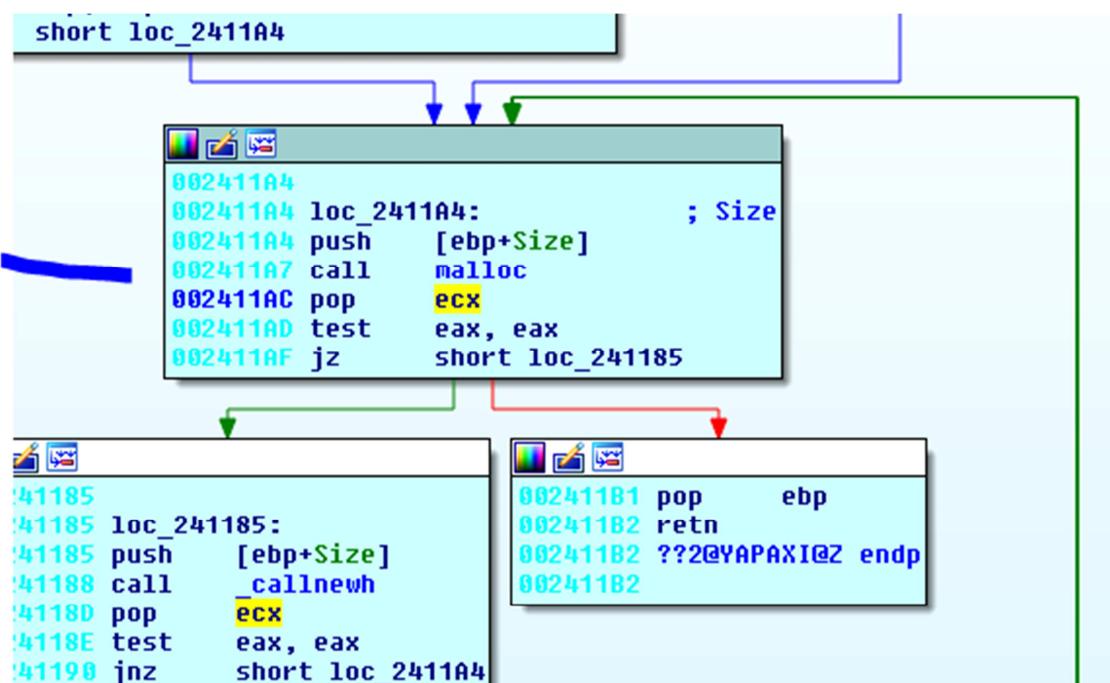
```

That command is very useful. It just works completely with the page heap full enabled. We see it tells me the allocated block size that it is used or busy (not free) and tells me the place history where it passed when that block was allocated.

If we had run the same command in a block that was freed or **free**, it would tell us the history of those places where it was freed.

The allocation comes from:

002411ac PRACTICA41b+0x000011ac



And above in the history list, we see how it calls malloc. Then, it internally calls RtlAllocateHeap, etc.

Downwards in the history list, we have:

0024109d PRACTICA41b+0x0000109d

The screenshot shows the assembly view of the debugger. A blue arrow points from the assembly code at address 0024109d down to the memory dump windows below. The assembly code includes instructions for pushing arguments onto the stack, calling the operator new, and then comparing the result with zero. The memory dump windows show the stack contents at addresses 002410A9 and 002410C1, where the value 0 is stored at [ebp+var\_10].

```

0024109d
0024109d var_10= dword ptr -10h
0024109d Size= dword ptr -8Ch
0024109d Dst= dword ptr -8
0024109d Buf= dword ptr -4
0024109d argc= dword ptr 8
0024109d argv= dword ptr 0Ch
0024109d envp= dword ptr 10h
0024109d
0024109d push    ebp
0024109d mov     ebp, esp
0024109d sub     esp, 10h
0024109d push    6Ch           ; Size
0024109d call    ??2@YAPAXI@Z      ; operator new(uint)
0024109d add     esp, 4
002410A0 mov     [ebp+Dst], eax
002410A3 cmp     [ebp+Dst], 0
002410A7 jz     short loc_2410C1

```

|  |  |
|--|--|
| 002410A9 push 6Ch ; Size<br>002410AB push 0 ; Val<br>002410AD mov eax, [ebp+Dst] | 002410C1 loc_2410C1:<br>002410C1 mov [ebp+var_10], 0 |
|--|--|

We see it marks the calls return address where it entered to allocate.

The screenshot shows the WinDbg debugger's output window displaying a heap dump. It lists memory allocations and their details. A blue arrow points from the output window to the specific allocation entry for address 054f5000, which is highlighted in yellow. The output shows the allocation was made via the RtlDebugAllocateHeap function.

```

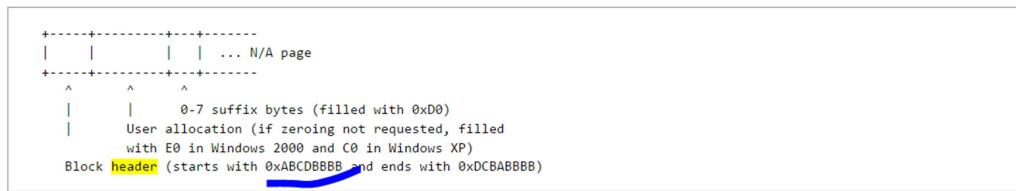
Output window
0024136a PRACTICA41b+0x0000136a
75a2338a kernel32!BaseThreadInitThunk+0x0000000e
77599a02 ntdll!_RtlUserThreadStart+0x00000070
775999d5 ntdll!_RtlUserThreadStart+0x0000001b

WINDBG>heap -p -a esi
address 054f5000 found in
_DPH_HEAP_ROOT @ 3ce1000
In busy allocation ( _DPM_HEAP_BLOCK:
54d16b4: UserAddr 54f4f90 UserSize - 6c -
VirtAddr 54f4000 VirtSize) 2000
59e58e89 verifier!AUvFDebugPageHeapAlloc+0x00000000
77631d4e ntdll!RtlDebugAllocateHeap+0x00000030
775eb586 ntdll!RtlpAllocateHeap+0x000000c4
77599541 ntdll!RtlAllocateHeap+0x00000023a
6d345e7b ucrtbase!malloc+0x0000002b

```

It shows the user address that is the block start for the user where it can be written. The block header is before.

Full page heap block -- allocated:



Full page heap block -- freed:



To see the stack trace of the allocation or the freeing of a heap block or full page heap block, use `dt DPH_BLOCK_INFORMATION` with the `header` address, followed by `g` to start the trace.

In Microsoft WEB, we see the header info, in this case, for the heap in Full Page Mode. It starts with ABCDBBBB and ends with DCABBBBB, Let's check if we see it just before the start where we wrote.

A screenshot of the WinDbg debugger. The assembly dump shows a sequence of memory writes starting from address 054F4F70. The writes include:

- 054F4F70 dd 0ABCDBBBB (highlighted)
- 054F4F71 db 0
- 054F4F75 db 10h
- 054F4F76 db 0CEh ; +
- 054F4F77 db 3
- 054F4F78 db 6Ch ; l
- 054F4F79 db 0
- 054F4F7A db 0
- 054F4F7B db 0
- 054F4F7C db 0
- 054F4F7D db 10h
- 054F4F7E db 0
- 054F4F7F db 0
- 054F4F80 db 0
- 054F4F81 db 0
- 054F4F82 db 0
- 054F4F83 db 0
- 054F4F84 db 0
- 054F4F85 db 0
- 054F4F86 db 0
- 054F4F87 db 0
- 054F4F88 dd offset unk\_3F2518C (highlighted)
- 054F4F8C dd 0DCBABBBB (highlighted)
- 054F4F90 db 41h ; A
- 054F4F91 db 41h ; A

The stack trace shows the command `dt UNKNOWN 054F4F70: debug975:054F4F70`.

With the Windbg `dt` command followed by `_DPH_BLOCK_INFORMATION` it will give info about the header fields.

Con el comando `dt` del Windbg seguido de `_DPH_BLOCK_INFORMATION` nos dará la información de los campos del header.

If we go to the address pointed by the stack trace...

```
UNKnown 054F4F70: debug975:054F4F70 (Synchronized With EIF)

WINDBG>dt _DPH_BLOCK_INFORMATION 054F4F70
verifier!_DPH_BLOCK_INFORMATION
+0x000 StartStamp : 0xabcdbbbb
+0x004 Heap : 0x03ce1000 Void
+0x008 RequestedSize : 0x6c
+0x00c ActualSize : 0x1000
+0x010 Internal : _DPH_BLOCK_INTERNAL_INFORMATION
+0x018 StackTrace : 0x03f2518c Void
+0x01c EndStamp : 0xdccbabb

WINDBG
```

```
|BG>#heap -p -a esi  
address 054f5000 found in  
_DPH_HEAP_ROOT @ 3ce1000  
in busy allocation ( DPH_HEAP_BLOCK: UserAddr  
54d16b4: 54f4f90  
59e58e89 verifier!UfRDebugPageHeapAllocate+0x000000229  
77631d4e ntdl!RtlDebugAllocateHeap+0x00000030  
775eb586 ntdl!RtlpAllocateHeap+0x000000c4  
77593541 ntdl!RtlAllocateHeap+0x0000023a  
6d34e7b ucrtbase!malloc+0x0000002b  
002411ac PRACTIC41b+0x000011ac  
0024109d PRACTIC41b+0x0000109d  
0024136a PRACTIC41b+0x0000136a  
75a2338a kernel32!BaseThreadInitThunk+0x0000000e  
77599a02 ntdll!RtlUserThreadStart+0x00000070
```

We see that a little down, it saves the allocation history. It matches with what the command gave us.

```
|heap -p -a xxx
```

Now, we'll try what info it gives us when we use it with a normal heap. Obviously, it won't be that specific nor have the history of each block.

I disable the Page Guard Full.

```
C:\Program Files <x86>\Windows Kits\10\Debuggers\x86>gflags.exe -p /disable PRACTICA41b.exe
Warning: pageheap.exe is running inside WOW64.
This scenario can be used to test x86 binaries (running inside WOW64)
but not native (IA64) binaries.

path: SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
      practica41b.exe: page heap disabled

C:\Program Files <x86>\Windows Kits\10\Debuggers\x86>
```

I run the script again and when it stops, I attach IDA again with the analysis and the local Windbg debugger as before.

Logically, we don't have the same info and the program crashed when it jumped to execute. Let's see the heap.

```
[Output window]
22 potential unreachable blocks were detected.

WINDBG>!heap -s

***** NT HEAP STATS BELOW *****
LFH Key : 0x4383e84f
Termination on corruption : ENABLED
Heap Flags Reserv Commit Virt Free List UCR Virt Lock Fast
(k) (k) (k) (k) (k) length blocks cont. heap
-----
00670000 00000002 1024 312 1024 9 8 1 0 0 LFH
00030000 00001002 1088 256 1088 5 2 2 0 0 LFH
-----
```

The heap stats.

If we go to the zone where it jumped, looking at the stack, we know it comes from there.

```

text:0040121000 mov    eax, [ebp+Size]
text:0040121004 mov    eax, [ebp+Buf]
text:0040121008 push   eax, [ebp+Size]
text:004012100C push   eax, [ebp+Buf]
text:0040121010 call   ds:_gets_s
text:0040121014 add    esp, 8
text:0040121018 push   1
text:004012101C push   edx, [ebp+Buf]
text:0040121020 mov    eax, [edx+64h]
text:0040121024 add    esp, 4
text:0040121028 mov    ecx, [ebp+Buf]
text:004012102C push   ecx
text:0040121030 push   offset aHolaS
text:0040121034 call   sub_1121140
text:0040121038 add    esp, 8
text:004012103C pop    ebp
text:0040121040 retn
text:0040121043 sub_1121010 endp
text:0040121047 ; -----
text:0040121048 align 10h
text:0040121050

```

And we see that EBP didn't change and as we know the program, we'll cheat looking at the buffer value to which we passed a `gets_s` that is the allocated block start. Now, it copies there. It is passed as an argument (obviously, we can do this because it is a simple program and to learn, if not, we have to enable the **Page Guard** and do what we saw before)

```

.text:0040121000 mov    eax, [ebp+Size]
text:0040121004 mov    eax, [ebp+Buf]
text:0040121008 push   eax, [ebp+Size]
text:004012100C push   eax, [ebp+Buf]
text:0040121010 call   ds:_gets_s
text:0040121014 add    esp, 8
text:0040121018 push   1
text:004012101C push   edx, [ebp+Buf]
text:0040121020 mov    eax, [edx+64h]
text:0040121024 add    esp, 4
text:0040121028 mov    ecx, [ebp+Buf]
text:004012102C push   ecx
text:0040121030 push   offset aHolaS
text:0040121034 call   sub_1121140
text:0040121038 add    esp, 8
text:004012103C pop    ebp
text:0040121040 retn
text:0040121043 sub_1121010 endp

```

The Buf variable continues pointing to the heap block start. So, we can go there.

```

debug031:006B8E3D db 0
debug031:006B8E3E db 72h ; r
debug031:006B8E3F db 0
debug031:006B8E40 db 40h ; @
debug031:006B8E41 db 0E8h ; p
debug031:006B8E42 db 92h ; E
debug031:006B8E43 db 40h ; M
debug031:006B8E44 db 1Fh
debug031:006B8E45 db 30h ; 0
debug031:006B8E46 db 0
debug031:006B8E47 db 0Ch
debug031:006B8E48 unk_6B8E48 db 41h ; A ; DATA XREF: debug020:002AF084To
debug031:006B8E49 db 41h ; A
debug031:006B8E4A db 41h ; A
debug031:006B8E4B db 41h ; A
debug031:006B8E4C db 41h ; A
debug031:006B8E4D db 41h ; A
debug031:006B8E4E db 41h ; A
debug031:006B8E4F db 41h ; A
debug031:006B8E50 db 41h ; A
debug031:006B8E51 db 41h ; A
debug031:006B8E52 db 41h ; A
debug031:006B8E53 db 41h ; A
debug031:006B8E54 db 41h ; A
debug031:006B8E55 db 41h ; A
debug031:006B8E56 db 41h ; A
debug031:006B8E57 db 41h ; A

```

```

Content source: I:\target\, length: 100
WINDBG>U!heap -x 0x6b8e48
^ Syntax error in 'U!heap -x 0x6b8e48'
WINDBG>?heap -x 0x6b8e48
List corrupted: (Flink->Blink = 41414141) != (Block = 006b2c60)
HEAP 00670000 (Seg 00670000) At 006b2c58 Error: block list entry corrupted

```

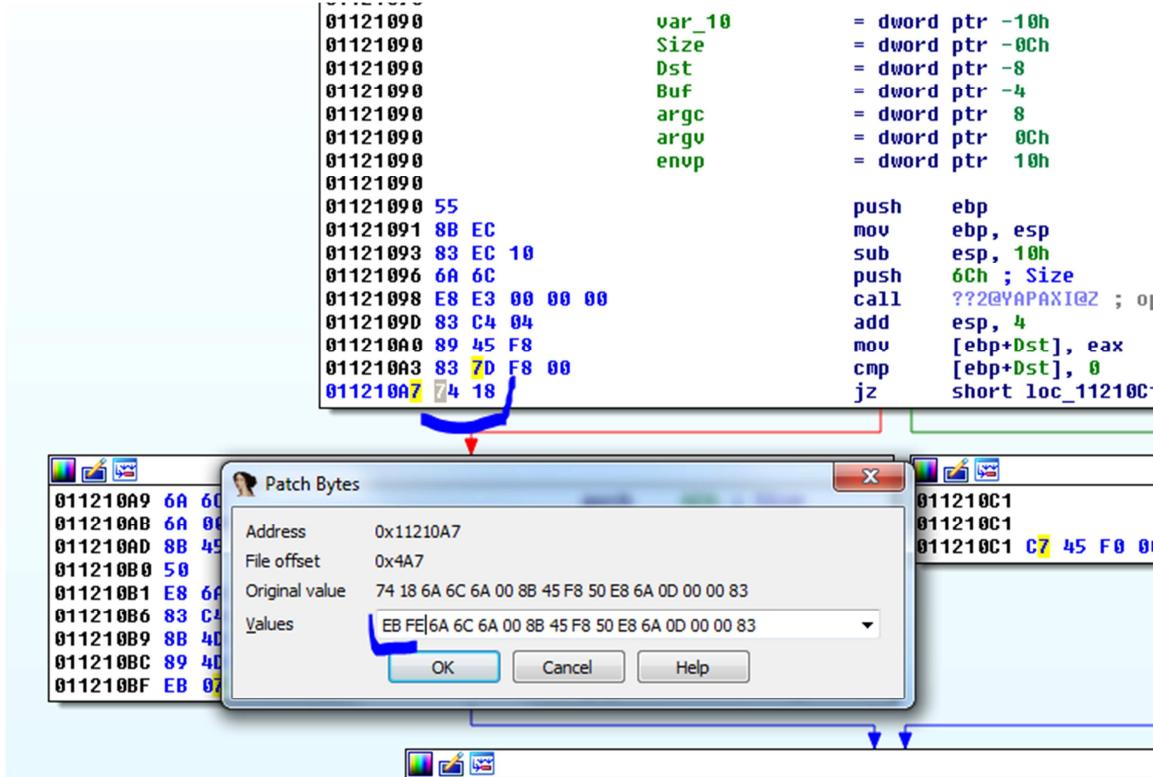
**ERROR: Block 006b8eb8 previous size 71d7 does not match previous block size f**

HEAP 00670000 (Seg 00670000) At 006b8eb8 Error: invalid block Previous

| Entry    | User     | Heap     | Segment  | Size | PreSize | Unused | Flags |
|----------|----------|----------|----------|------|---------|--------|-------|
| 006b8e40 | 006b8e48 | 00670000 | 00670000 | 78   | 448     | c      | busy  |

It is corrupted. We know that.

Obviously, as all is broken, we'll try to attach it before it breaks the heap to see a block info. I can't run it directly in IDA because it runs that in debug mode to the heap. So, I we assemble an EB FE at the start to leave it looping and when I run it, I will attach it.



I'll change the **74 18** of the conditional jump to **EB FE**. Then, **EDIT-PATCH PROGRAM- APPLY PATCH TO INPUT FILE**.

Now, I run the script. When it is looping, I stop there. As it already passed through the malloc, I can see the heap.

The screenshot shows the **IDA View-EIP** window. The assembly code is:

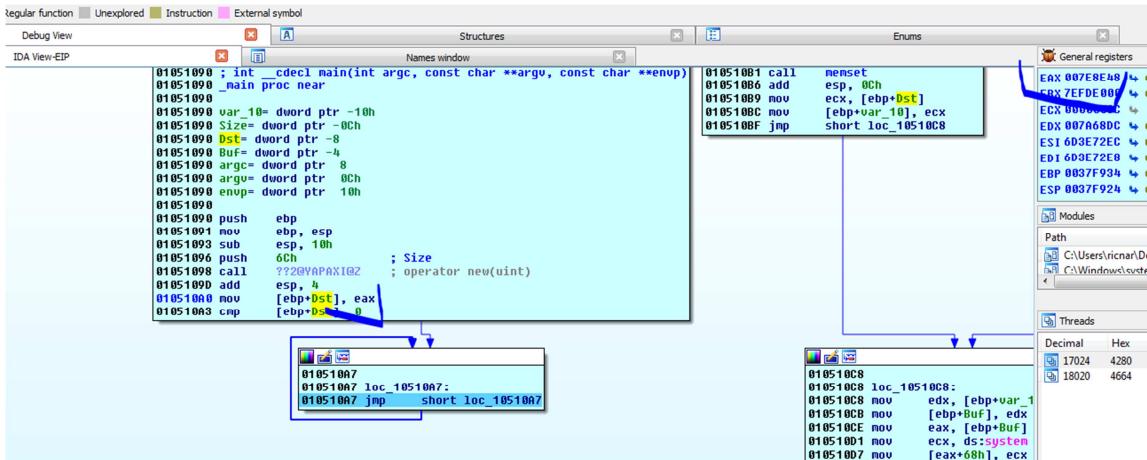
```

.text:01051090 push    ebp
.text:01051091 mov     ebp, esp
.text:01051093 sub    esp, 10h
.text:01051096 push    6Ch ; Size
.text:01051098 call   ??2@YAPAXI@Z ; operator new(uint)
.text:0105109D add    esp, 4
.text:010510A0 mov    [ebp+Dst], eax
.text:010510A3 cmp    [ebp+Dst], 0
.text:010510A7
.text:010510A7 loc_10510A7:           ; CODE XREF: _main:loc_10510A7↓j
.jmp   short loc_10510A7
.text:010510A9 ;
.text:010510A9 push    6Ch ; Size
.text:010510AB push    0 ; Val
.text:010510AD mov    eax, [ebp+Dst]

```

A blue arrow points from the **jmp** instruction at address **010510A7** to the assembly code above it.

After allocating EAX, it remains with the block address.



Let's see what it says.

| Expected data back. |          |          |          |      |          |        |       |
|---------------------|----------|----------|----------|------|----------|--------|-------|
| Entry               | User     | Heap     | Segment  | Size | PrevSize | Unused | Flags |
| 007e8e40            | 007e8e48 | 007a0000 | 007a0000 | 78   | 448      | c      | busy  |

As User is always the writable part and Entry is the header start, let's see.

```
Scanning ...  
Scanning references from 252 busy blocks (0 MBytes) ...No potential unreachable blocks were detected.  
WINDBG> !heap -s
```

---

```
***** NT HEAP STATS BELOW *****  
LFH Key : 0x6F05ba4c  
Termination on corruption : ENABLED  
Heap Flags Reserv Commit Virt Free List UCR Virt Lock Fast  
(K) (K) (K) (K) length blocks cont. heap  
007a0000 00000002 1024 300 1024 6 6 1 0 0 LFH
```

It shows just one heap and there, it is. Let's see its content.

```
WINDBG>!heap -a 007a0000  
Index Address Name Debugging options enabled  
1: 007a0000  
Segment at 007a0000 to 008a0000 (0004b000 bytes committed)  
Flags: 00000002  
ForceFlags: 00000000  
Granularity: 8 bytes  
Segment Reserve: 00100000  
Segment Commit: 00002000  
DeCommit Block Thres: 00000800  
DeCommit Total Thres: 00002000  
Total Free Size: 0000031b
```

Max. Allocation Size: 7ffdffff  
Lock Variable at: 007a0138  
Next TagIndex: 0000  
Maximum TagIndex: 0000  
Tag Entries: 00000000  
PsuedoTag Entries: 00000000  
Virtual Alloc List: 007a00a0  
Uncommitted ranges: 007a0090  
007eb000: 000b5000 (741376 bytes)  
FreeList[ 00 ] at 007a00c4: 007e8ec0 . 007e4e90  
007e4e88: 00028 . 00010 [100] - free  
007a6750: 00028 . 00010 [100] - free  
007a6158: 00050 . 00010 [100] - free  
007e2d30: 00028 . 00018 [100] - free  
007e2c58: 00210 . 00018 [100] - free  
007e8eb8: 00078 . 01878 [100] - free

Segment00 at 007a0000:

Flags: 00000000  
Base: 007a0000  
First Entry: 007a0588  
Last Entry: 008a0000  
Total Pages: 00000100  
Total UnCommit: 000000b5  
Largest UnCommit: 00000000  
UnCommitted Ranges: (1)

Heap entries for Segment00 in Heap 007a0000  
address: psize . size flags state (requested size)  
007a0000: 00000 . 00588 [101] - busy (587)  
007a0588: 00588 . 00240 [101] - busy (23f)  
007a07c8: 00240 . 00020 [101] - busy (18)  
007a07e8: 00020 . 01dd8 [101] - busy (1dce)  
007a25c0: 01dd8 . 02d00 [101] - busy (2cf8)  
007a52c0: 02d00 . 00048 [101] - busy (3c)  
007a5308: 00048 . 00038 [101] - busy (30)  
007a5340: 00038 . 00080 [101] - busy (78)  
007a53c0: 00080 . 00080 [101] - busy (78)  
007a5440: 00080 . 00048 [101] - busy (3c)  
007a5488: 00048 . 00228 [101] - busy (220)  
007a56b0: 00228 . 00050 [101] - busy (42)  
007a5700: 00050 . 00080 [101] - busy (78)

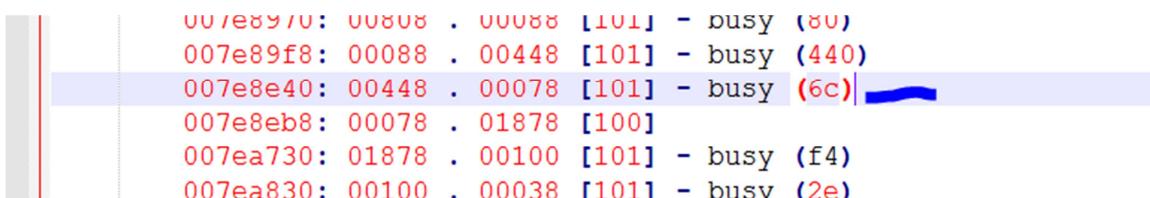
007a5780: 00080 . 00018 [101] - busy (10)  
007a5798: 00018 . 00050 [101] - busy (46)  
007a57e8: 00050 . 00080 [101] - busy (78)  
007a5868: 00080 . 00018 [101] - busy (10)  
007a5880: 00018 . 00018 [101] - busy (10)  
007a5898: 00018 . 00020 [101] - busy (14)  
007a58b8: 00020 . 00070 [101] - busy (64)  
007a5928: 00070 . 00208 [101] - busy (200)  
007a5b30: 00208 . 00208 [101] - busy (200)  
007a5d38: 00208 . 00030 [101] - busy (24)  
007a5d68: 00030 . 00030 [101] - busy (24)  
007a5d98: 00030 . 00038 [101] - busy (30)  
007a5dd0: 00038 . 00028 [101] - busy (20)  
007a5df8: 00028 . 00028 [101] - busy (20)  
007a5e20: 00028 . 00028 [101] - busy (20)  
007a5e48: 00028 . 00028 [101] - busy (20)  
007a5e70: 00028 . 00018 [101] - busy (10)  
007a5e88: 00018 . 00080 [101] - busy (78)  
007a5f08: 00080 . 00080 [101] - busy (78)  
007a5f88: 00080 . 00018 [101] - busy (10)  
007a5fa0: 00018 . 00020 [101] - busy (14)  
007a5fc0: 00020 . 00020 [101] - busy (10)  
007a5fe0: 00020 . 00078 [101] - busy (6c)  
007a6058: 00078 . 00080 [101] - busy (78)  
007a60d8: 00080 . 00018 [101] - busy (10)  
007a60f0: 00018 . 00018 [101] - busy (10)  
007a6108: 00018 . 00050 [101] - busy (42)  
007a6158: 00050 . 00010 [100]  
007a6168: 00010 . 00058 [101] - busy (4a)  
007a61c0: 00058 . 00080 [101] - busy (78)  
007a6240: 00080 . 00020 [101] - busy (10)  
007a6260: 00020 . 00018 [101] - busy (10)  
007a6278: 00018 . 00080 [101] - busy (78)  
007a62f8: 00080 . 00020 [101] - busy (10)  
007a6318: 00020 . 00018 [101] - busy (10)  
007a6330: 00018 . 00018 [101] - busy (10)  
007a6348: 00018 . 00070 [101] - busy (68)  
007a63b8: 00070 . 00080 [101] - busy (78)  
007a6438: 00080 . 00018 [101] - busy (10)  
007a6450: 00018 . 00070 [101] - busy (68)  
007a64c0: 00070 . 00078 [101] - busy (70)  
007a6538: 00078 . 00080 [101] - busy (78)

007a65b8: 00080 . 00020 [101] - busy (10)  
007a65d8: 00020 . 00018 [101] - busy (10)  
007a65f0: 00018 . 00020 [101] - busy (10)  
007a6610: 00020 . 00078 [101] - busy (6a)  
007a6688: 00078 . 00088 [101] - busy (7c)  
007a6710: 00088 . 00018 [101] - busy (10)  
007a6728: 00018 . 00028 [101] - busy (20)  
007a6750: 00028 . 00010 [100]  
007a6760: 00010 . 00080 [101] - busy (78)  
007a67e0: 00080 . 00080 [101] - busy (78)  
007a6860: 00080 . 03d20 [101] - busy (3d1f)  
007aa580: 03d20 . 378b0 [101] - busy (378a8) Internal  
007e1e30: 378b0 . 00080 [101] - busy (78)  
007e1eb0: 00080 . 00020 [101] - busy (17)  
007e1ed0: 00020 . 00400 [101] - busy (3f8) Internal  
007e22d0: 00400 . 00400 [101] - busy (3f8) Internal  
007e26d0: 00400 . 00080 [101] - busy (78)  
007e2750: 00080 . 00080 [101] - busy (78)  
007e27d0: 00080 . 00028 [101] - busy (20)  
007e27f8: 00028 . 00028 [101] - busy (20)  
007e2820: 00028 . 00070 [101] - busy (66)  
007e2890: 00070 . 00080 [101] - busy (78)  
007e2910: 00080 . 00028 [101] - busy (20)  
007e2938: 00028 . 00028 [101] - busy (20)  
007e2960: 00028 . 00070 [101] - busy (68)  
007e29d0: 00070 . 00078 [101] - busy (6a)  
007e2a48: 00078 . 00210 [101] - busy (208)  
007e2c58: 00210 . 00018 [100]  
007e2c70: 00018 . 00070 [101] - busy (66)  
007e2ce0: 00070 . 00028 [101] - busy (20)  
007e2d08: 00028 . 00028 [101] - busy (20)  
007e2d30: 00028 . 00018 [100]  
007e2d48: 00018 . 00078 [101] - busy (6c)  
007e2dc0: 00078 . 02000 [101] - busy (1ff8) Internal  
007e4dc0: 02000 . 00028 [101] - busy (20)  
007e4de8: 00028 . 00028 [101] - busy (20)  
007e4e10: 00028 . 00028 [101] - busy (20)  
007e4e38: 00028 . 00028 [101] - busy (20)  
007e4e60: 00028 . 00028 [101] - busy (20)  
007e4e88: 00028 . 00010 [100]  
007e4e98: 00010 . 00078 [101] - busy (6a)  
007e4f10: 00078 . 00408 [101] - busy (400)

007e5318: 00408 . 00028 [101] - busy (20)  
007e5340: 00028 . 00800 [101] - busy (7f8) Internal  
007e5b40: 00800 . 006d0 [101] - busy (6c8)  
007e6210: 006d0 . 00c08 [101] - busy (c00)  
007e6e18: 00c08 . 00800 [101] - busy (7f8) Internal  
007e7618: 00800 . 00228 [101] - busy (220)  
007e7840: 00228 . 00228 [101] - busy (220)  
007e7a68: 00228 . 00490 [101] - busy (483)  
007e7ef8: 00490 . 00218 [101] - busy (209)  
007e8110: 00218 . 00058 [101] - busy (4a)  
007e8168: 00058 . 00808 [101] - busy (800)  
007e8970: 00808 . 00088 [101] - busy (80)  
007e89f8: 00088 . 00448 [101] - busy (440)  
007e8e40: 00448 . 00078 [101] - busy (6c)  
007e8eb8: 00078 . 01878 [100]  
007ea730: 01878 . 00100 [101] - busy (f4)  
007ea830: 00100 . 00038 [101] - busy (2e)  
007ea868: 00038 . 00030 [101] - busy (28)  
007ea898: 00030 . 00040 [101] - busy (37)  
007ea8d8: 00040 . 00048 [101] - busy (3c)  
007ea920: 00048 . 00040 [101] - busy (31)  
007ea960: 00040 . 00030 [101] - busy (24)  
007ea990: 00030 . 00040 [101] - busy (32)  
007ea9d0: 00040 . 00038 [101] - busy (2e)  
007eaa08: 00038 . 00038 [101] - busy (2c)  
007eaa40: 00038 . 00030 [101] - busy (28)  
007eaa70: 00030 . 00030 [101] - busy (21)  
007eaaa0: 00030 . 00020 [101] - busy (15)  
007eaac0: 00020 . 00038 [101] - busy (2b)  
007eaaf8: 00038 . 00030 [101] - busy (22)  
007eab28: 00030 . 00038 [101] - busy (2e)  
007eab60: 00038 . 00048 [101] - busy (39)  
007eaba8: 00048 . 00020 [101] - busy (17)  
007eabc8: 00020 . 00040 [101] - busy (36)  
007eac08: 00040 . 00050 [101] - busy (47)  
007eac58: 00050 . 00050 [101] - busy (48)  
007eaca8: 00050 . 00020 [101] - busy (12)  
007eacc8: 00020 . 00020 [101] - busy (18)  
007eace8: 00020 . 00030 [101] - busy (24)  
007ead18: 00030 . 00038 [101] - busy (29)  
007ead50: 00038 . 00098 [101] - busy (8b)  
007eade8: 00098 . 00020 [101] - busy (17)

```
007eae08: 00020 . 00020 [101] - busy (11)
007eae28: 00020 . 00020 [101] - busy (18)
007eae48: 00020 . 00020 [101] - busy (17)
007eae68: 00020 . 00030 [101] - busy (21)
007eae98: 00030 . 00020 [101] - busy (13)
007eaeb8: 00020 . 00020 [101] - busy (14)
007eaed8: 00020 . 00020 [101] - busy (16)
007eaef8: 00020 . 00030 [101] - busy (28)
007eaf28: 00030 . 00030 [101] - busy (27)
007eaf58: 00030 . 00060 [101] - busy (52)
007eafb8: 00060 . 00028 [101] - busy (12)
007eafe0: 00028 . 00020 [111] - busy (1d)
007eb000: 000b5000 - uncommitted bytes.
```

If we see in the list, the block is there. If we find it by header address, it tells us the size.



```
007e89f0: 00088 . 00088 [101] - busy (80)
007e89f8: 00088 . 00448 [101] - busy (440)
007e8e40: 00448 . 00078 [101] - busy (6c) [ ]
007e8eb8: 00078 . 01878 [100]
007ea730: 01878 . 00100 [101] - busy (f4)
007ea830: 00100 . 00038 [101] - busy (2e)
```

```
*0x0000 Myregistry : 0x00000000 40001401
WINDBG>!heap -p -a eax
address 007e8e48 found in
- HEAP @ 7a0000
- HEAP_ENTRY Size Prev Flags     UserPtr UserSize - state
  007e8e40 000F 0000 [00]    007e8e48    0006c - (busy)
```

It doesn't show us the history, but the size, the general size is 0xF because to find the total, it multiplies it by 8 giving the following result:

```
hex(0xf *0x8)
```

```
'0x78'
```

That is the complete size with the header and the end, etc.

In the case of the normal heap, to see the values, we have to use:

```
WINDBG>dt _heap_entry 007e8e40
ntdll!_HEAP_ENTRY
+0x000 Size : 0x14c1
+0x002 Flags : 0xdf
+0x003 SmallTagIndex : 0x40 '@'
+0x000 SubSegmentCode : 0x40df14c1 Void
+0x004 PreviousSize : 0x685f
+0x006 SegmentOffset : 0 ''
+0x006 LFHFlags : 0 ''
+0x007 UnusedBytes : 0xc ''
+0x008 FunctionIndex : 0x14c1
+0x002 ContextValue : 0x40df
+0x000 InterceptorValue : 0x40df14c1
+0x004 UnusedBytesLength : 0x685f
+0x006 EntryOffset : 0 ''
+0x007 ExtendedBlockSignature : 0xc ''
+0x000 Code1 : 0x40df14c1
+0x004 Code2 : 0x685f
+0x006 Code3 : 0 ''
+0x007 Code4 : 0xc ''
+0x000 AgregateCode : 0x0c 00685f`40df14c1
```

The thing is that they are encoded (xored) with a constant where we can find the constant to unxor them.

```
!7E8E70 6C 00 65 00 73 00 28 00 78 00 38 00 36 00 29 00 1.e.s.(.x.8.6.)
!7E8E80 3D 00 43 00 3A 00 5C 00 50 00 72 00 6F 00 67 00 =.C.:.\P.r.o.g.
KNOWN 007E8E40: debug033:off_7E8E40
Output window
Output window
WINDBG>dt _heap 007a0000
ntdll!_HEAP
+0x000 Entry : _HEAP_ENTRY
+0x008 SegmentSignature : 0xffffeffe
+0x00c SegmentFlags : 0
+0x010 SegmentListEntry : _LIST_ENTRY [ 0x7a00a8 - 0x7a00a8 ]
+0x018 Heap : 0x007a0000 _HEAP
+0x01c BaseAddress : 0x007a0000 Void
+0x020 NumberOfPages : 0x100
+0x024 FirstEntry : 0x007a0588 _HEAP_ENTRY
+0x028 LastValidEntry : 0x008a0000 _HEAP_ENTRY
+0x02c NumberOfUnCommittedPages : 0xb5
+0x030 NumberOfUnCommittedRanges : 1
+0x034 SegmentAllocatorBackTraceIndex : 0
+0x036 Reserved : 0
+0x038 UCRSegmentList : _LIST_ENTRY [ 0x7eaff0 - 0x7eaff0 ]
+0x040 Flags : 2
+0x044 ForceFlags : 0
+0x048 CompatibilityFlags : 0
+0x04c EncodeFlagMask : 0x100000
+0x050 Encoding : _HEAP_ENTRY
+0x058 PointerKey : 0x6e0919cb
+0x05c Interceptor : 0
+0x060 VirtualMemoryThreshold : 0xfe00
+0x064 Signature : 0xeeffefff
+0x068 SegmentReserve : 0x100000
+0x06c SegmentCommit : 0x2000
```

The Offset 0x50 of the heap structure is called encoding.

```
WINDBG>dd 007a0000+ 0x50 L2
007a0050 4ede14ce 000068d6
```

They are those DWORDS which xored the info.

| 007A0020          | 00 01 00 00 88 05 7A 00 00 00 8A 00 B5 00 00 00 | ...è.z...è.Á...  |
|-------------------|---|------------------|
| 007A0030          | 01 00 00 00 00 00 00 F0 AF 7E 00 F0 AF 7E 00    | ....>~>~.        |
| 007A0040          | 02 00 00 00 00 00 00 00 00 00 00 00 00 10 00    | .....            |
| 007A0050          | CE 14 DE 4E D6 68 00 00 CB 19 09 6E 00 00 00 00 | +.ÍNÍh._-n...    |
| 007A0060          | 00 FE 00 00 FF EE FF EE 00 00 10 00 00 20 00 00 | .I...- - - -     |
| 007A0070          | 00 08 00 00 00 20 00 00 1B 03 00 00 FF EF FD 7F | .....- - - -     |
| 007A0080          | 01 00 38 01 00 00 00 00 00 00 00 00 00 00 00 00 | .8.....          |
| 007A0090          | E8 AF 7E 00 E8 AF 7E 00 0F 00 00 00 F8 FF FF FF | b>~.b>~.----o--- |
| UNKNOWN 007A0053: | debug033:007A0053                               |                  |

If we do the same in the header dividing it in two DWORDS:

dd 007E8E40 L2

I have the two DWORDS that I have to xor with the two ones of the entry.

```
WINDBG>dd 007a0000+ 0x50 L2  
007a0050 4ede14ce 000068d6
```

```
WINDBG>dd 007E8E40 L2  
007e8e40 40df14c1 0c00685f
```

I xor it.

```
WINDBG>? 4ede14ce ^ 40df14c1  
Evaluate expression: 234946575 = 0e01000f  
WINDBG>? 68d6 ^ 0c00685f  
Evaluate expression: 201326729 = 0c000089
```

We can create the little table. Obviously, we'll break the program and it won't run more, but just to see something.

|          | Hex View-1   |
|----------|--|
| 007E8E00 | 72 00 6F 00 67 00 72 00 61 00 6D 00 44 00 61 00 r.o.g.r.a.m.D.a. |
| 007E8E10 | 74 00 61 00 00 00 50 00 72 00 6F 00 67 00 72 00 t.a...P.r.o.g.r. |
| 007E8E20 | 61 00 6D 00 46 00 69 00 6C 00 65 00 73 00 3D 00 a.m.F.i.l.e.s.=. |
| 007E8E30 | 43 00 3A 00 5C 00 50 00 72 00 6F 00 67 00 72 00 C._.\.P.r.o.g.r. |
| 007E8E40 | C1 14 DF 40 5F 68 00 8C C4 00 7A 00 68 2C 7E 00 _._@_h..-z.^~.   |
| 007E8E50 | 20 00 28 00 78 00 38 00 36 00 29 00 00 00 50 00 _.(x.8.6.)...P.  |
| 007E8E60 | 72 00 6F 00 67 00 72 00 61 00 6D 00 46 00 69 00 r.o.g.r.a.m.F.i. |
| 007E8E70 | 6C 00 65 00 73 00 28 00 78 00 38 00 36 00 29 00 l.e.s.(x.8.6.).  |
| UNKNOWN  | 007E8E40: debug033:007E8E40                                      |

I replace the values for the xored ones.

```

debug033:007E8E50 db 20h
debug033:007E8E51 db 0

UNKNOWN 007E8E40: debug033:007E8E40 (Synchronized with EIP)

Hex View-1

007E8E30 43 00 3A 00 5C 00 50 00 72 00 6F 00 67 00 72 00
007E8E40 9F 00 [REDACTED] 89 00 00 0C C4 00 7A 00 60 2C 7E 00
007E8E50 20 00 28 00 78 00 38 00 36 00 29 00 00 00 50 00
007E8E60 72 00 6F 00 67 00 72 00 61 00 60 00 46 00 69 00
007E8E70 6C 00 65 00 73 00 28 00 78 00 38 00 36 00 29 00
007E8E80 3D 00 43 00 3A 00 5C 00 50 00 72 00 6F 00 67 00
007E8E90 72 00 61 00 6D 00 29 00 46 00 69 00 6C 00 65 00
007E8EA0 73 00 20 00 28 00 78 00 38 00 36 00 29 00 00 00

UNKNOWN 007E8E40: debug033:007E8E40

Output window

007E8E30 EncryptedSector . .
+0x007 ExtendedBlockSignature : 0x89 ''
+0x000 Code1 : 0xf00010e
+0x004 Code2 : 0xc
+0x006 Code3 : 0 ''
+0x007 Code4 : 0x89 ''
+0x000 AggregateCode : 0x8900000c`0f00010e

WINDBG>dt _heap_entry 007e8e40
ntdll!_HEAP_ENTRY
+0x000 Size : 0xF
+0x002 Flags : 0x1 ''
+0x003 SmallTagIndex : 0xe ''
+0x000 SubSegmentCode : 0x0e01000f Void
+0x004 PreviousSize : 0x89
+0x006 SegmentOffset : 0 ''
+0x006 LFHFlags : 0 ''
+0x007 UnusedBytes : 0xc ''
+0x000 FunctionIndex : 0xF
+0x002 ContextValue : 0x0001
+0x006 IntercepterValue : 0xe01000f
+0x004 UnusedBytesLength : 0x89
+0x006 EntryOffset : 0 ''
+0x007 ExtendedBlockSignature : 0xc ''
+0x000 Code1 : 0xe01000f
+0x004 Code2 : 0x89
+0x006 Code3 : 0 ''
+0x007 Code4 : 0xc ''
+0x000 AggregateCode : 0x0c000089`0e01000f

WINDBG>? 0xF* 8
Evaluate expression: 120 = 00000078

```

As in this case, the size is 0xF, we have to multiply it by 8 to find the total size and it gives 0x78 that is the same it gave at the beginning what includes the header and the end.

We go slowly observing and getting familiar with the heap blocks. In the next part, we'll see what mona tells us about this. If it helps or not. ☺



**Ricardo Narvaja**

**Translated by: @IvinsonCLS**