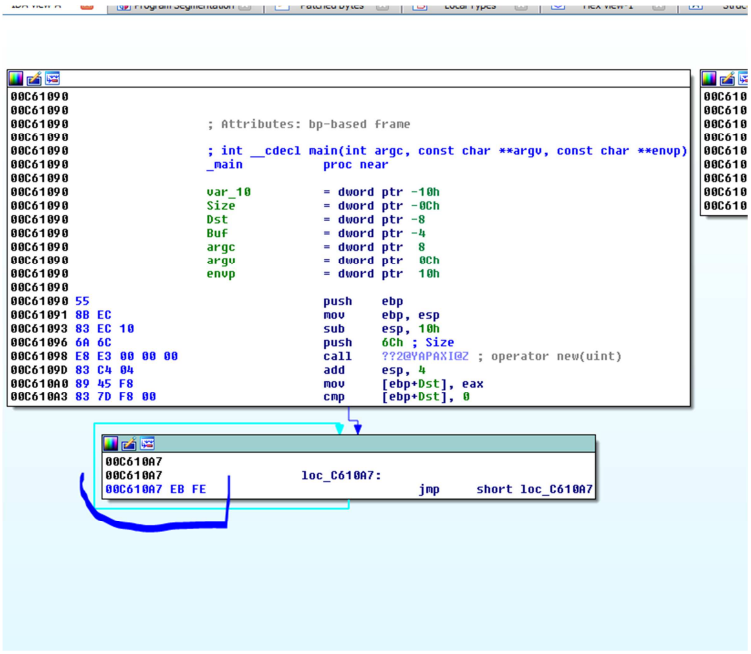
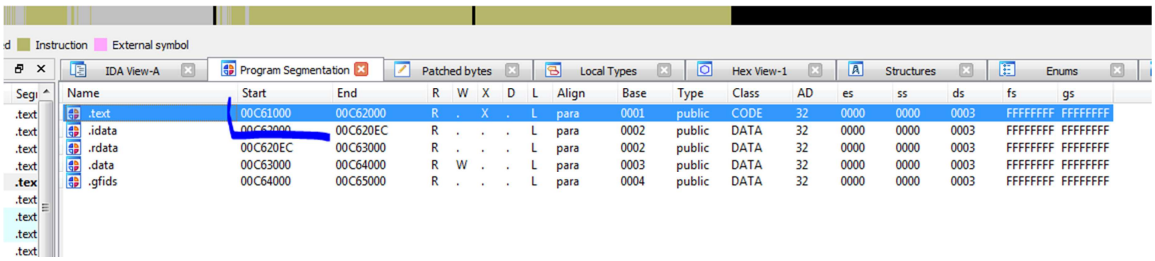
PART 45

We will continue practicing with the PRACTICA41b exercise, but this time with Windbg using Mona outside IDA. We'll use the executable we added the EB FE to remain looping. Anyways, it is good to keep it also open in IDA LOADER without debugging to have things clear.

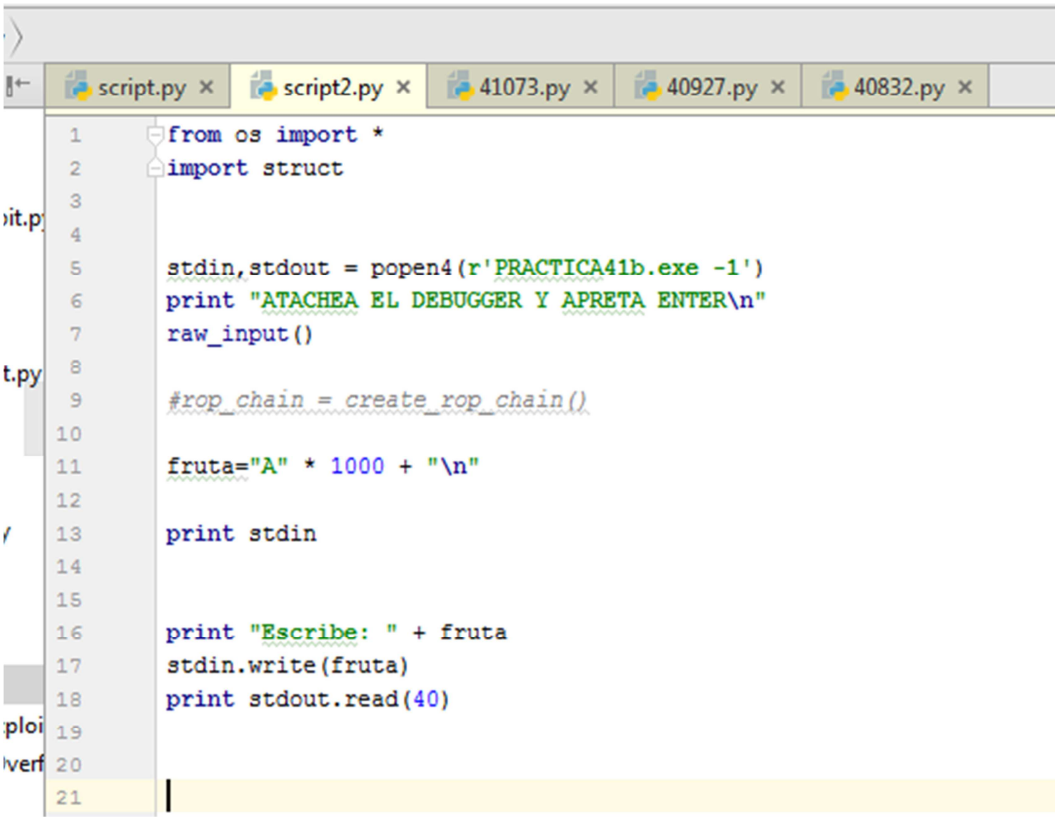
The idea is seeing what info it gives us in both cases using Windbg inside IDA or outside it with mona.



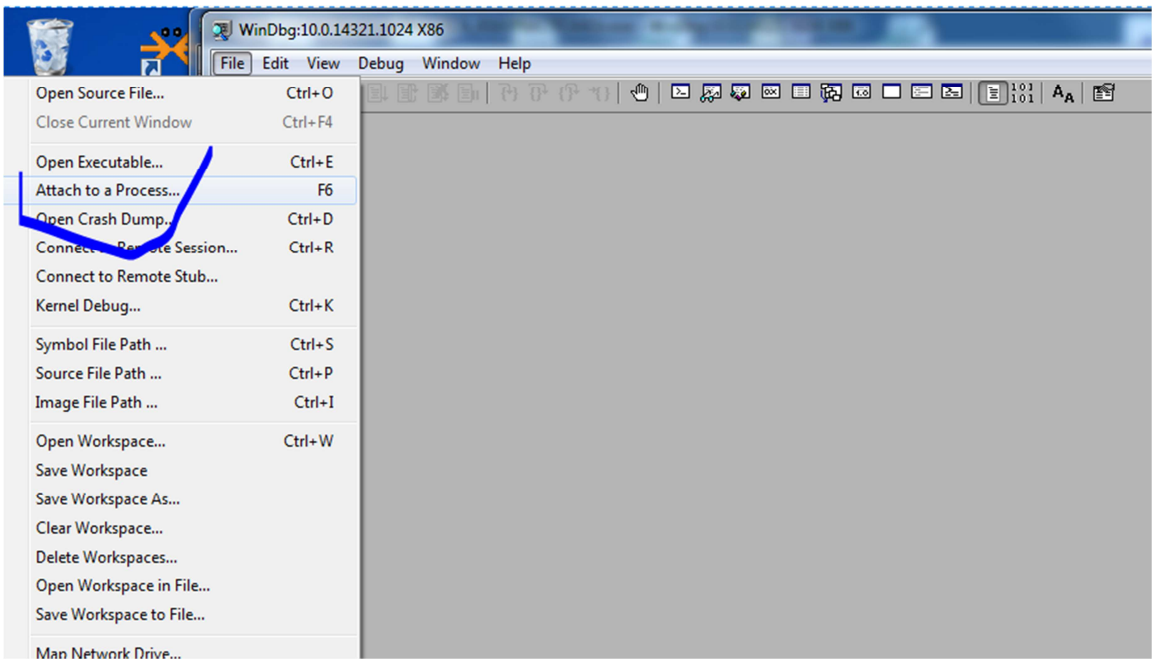
In this case, IDA, in the LOADER, shows me the **infinite loop** address in my PC: 0xc610a7 and the **code section** start in SEGMENTS.



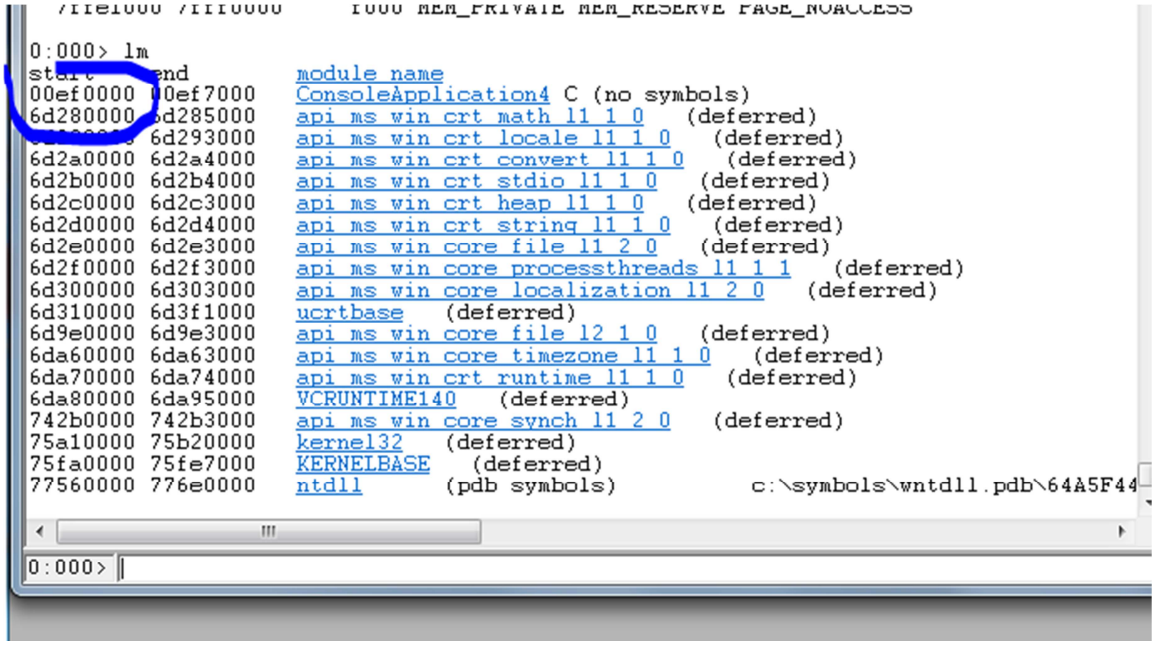
If I do the subtraction from the code section start, the loop is 0xA7 more ahead and from the imagebase 0xc60000 will be 0x10A7.



I run the script and the process will be looping. So, I open the Windbg that has Mona installed as we saw in the previous parts.

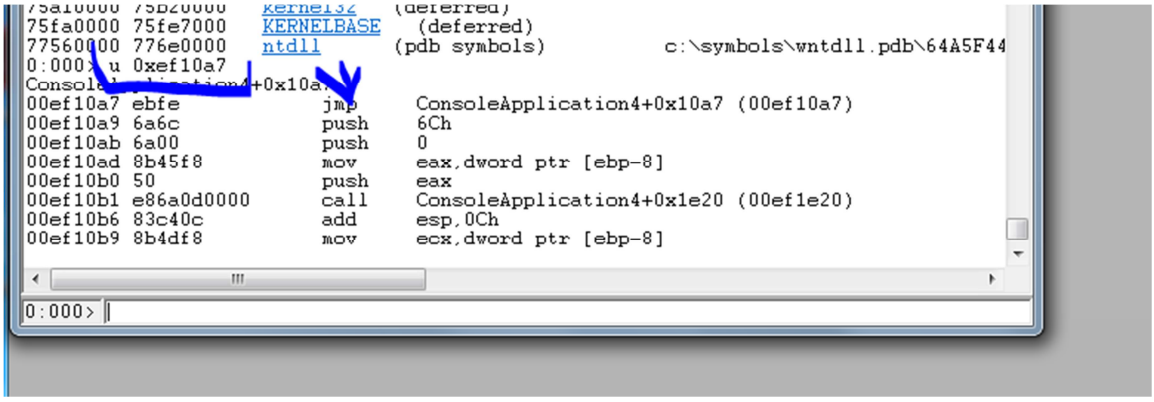


When I attach it and it stops, I see the imagebase because the executable module has ASLR. I see the module list with **lm**.

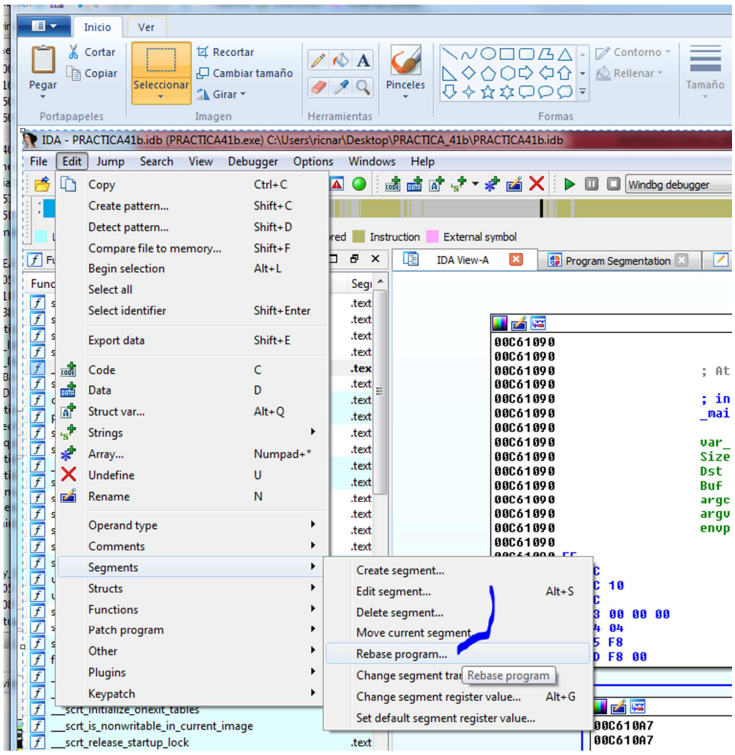


The name, in this case, is the original one with what it was compiled.

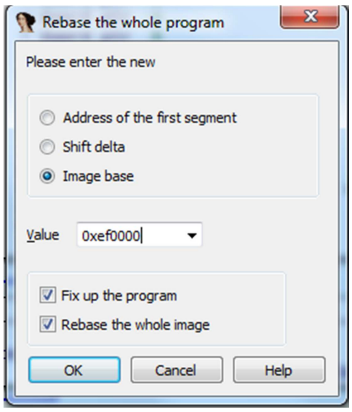
If I add 0x10a7 to the imagebase 0xef0000, it will be 0xef10a7 that is where the JMP is.

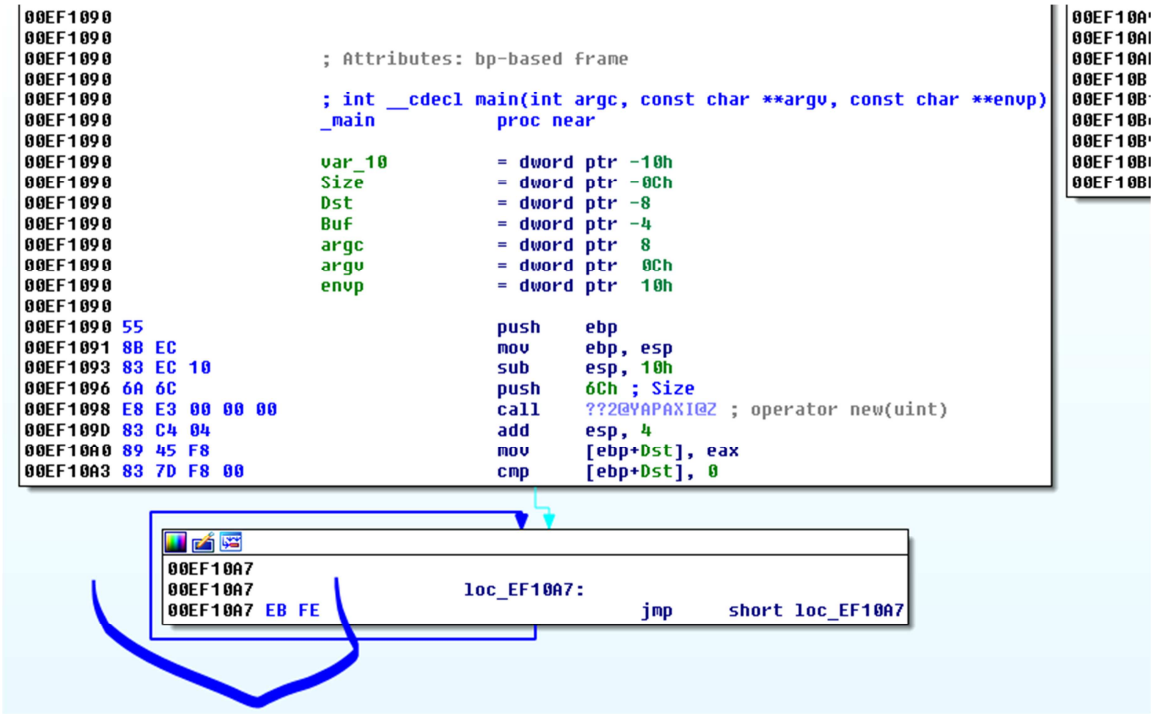


If I want that IDA matches in the addresses with the process that Windbg runs, I go to:



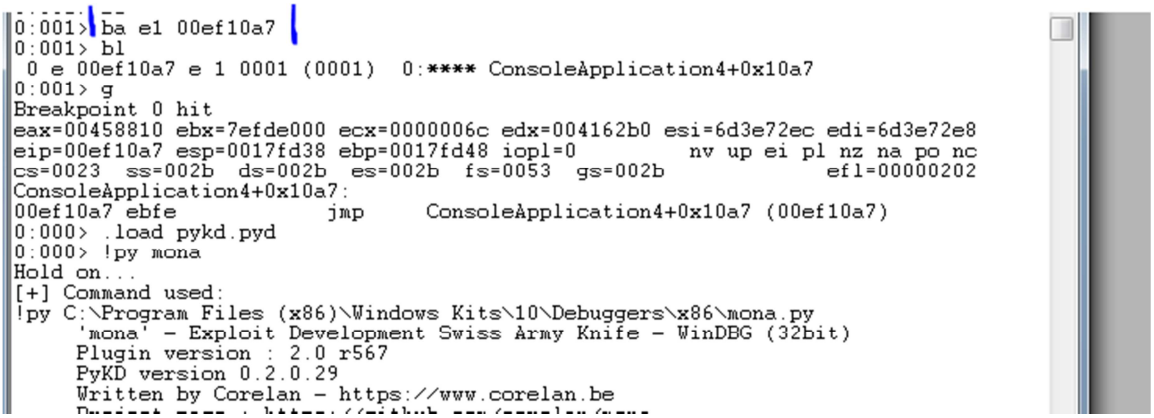
I enter the process imagebase that runs in Windbg. It was 0xef0000.



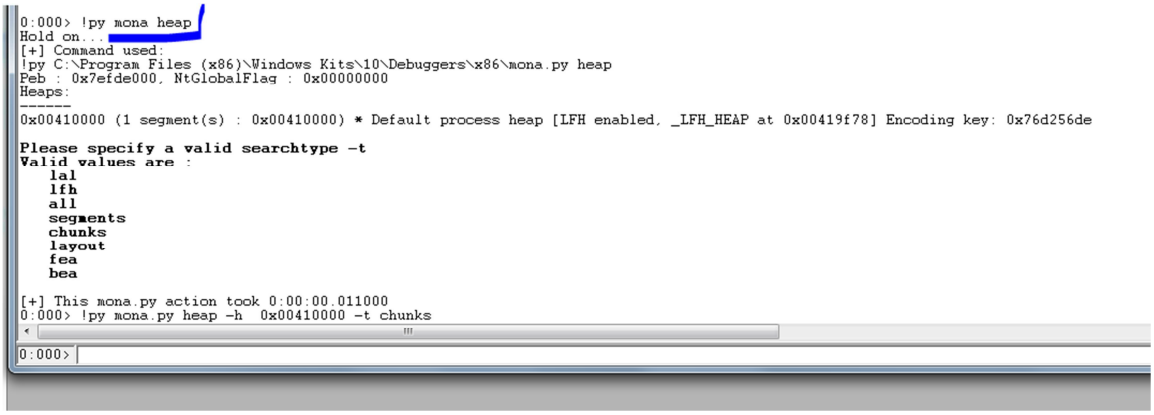


We see it matches the process address in Windbg. Obviously, each time I run it, the address will change and I have to repeat the rebase.

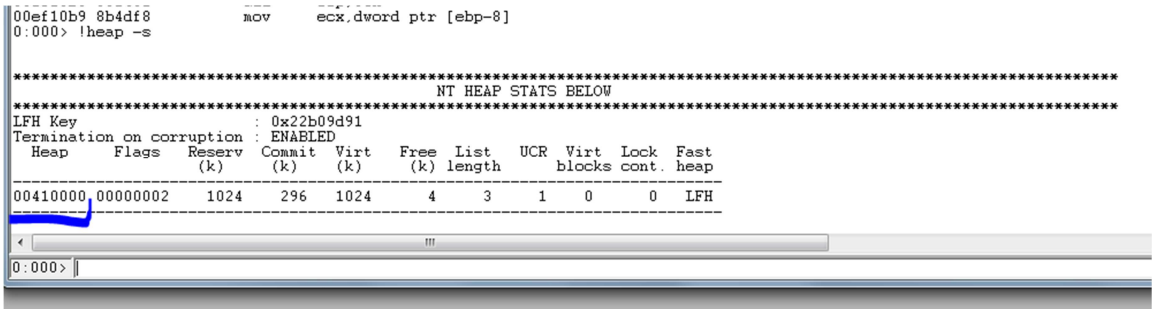
There, I see the listed instruction. I set a BP on execution with:



There, I set a BP and typed “g” to RUN it and it stops there. Then, I load Mona and use its heap command.



We compare it with Windbg result. No much info. Mona just gives us one of the encoding keys.



Let's see what Mona tells us about the chunks (blocks)

0:000> !py mona.py heap -h 0x00410000 -t chunks

Hold on...

[+] Command used:

!py C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\mona.py heap -h 0x00410000 -t chunks

Peb : 0x7efde000, NtGlobalFlag : 0x00000000

Heaps:

0x00410000 (1 segment(s) : 0x00410000) * Default process heap [LFH enabled, _LFH_HEAP at 0x00419f78] Encoding key: 0x76d256de

[+] Preparing output file 'heapchunks.txt'

- (Re)setting logfile heapchunks.txt

[+] Generating module info table, hang on...

- Processing modules

- Done. Let's rock 'n roll.

[+] Processing heap 0x00410000 [LFH]

Segment List for heap 0x00410000:

Segment 0x00410588 - 0x00510000 (FirstEntry: 0x00410588 - LastValidEntry: 0x00510000): 0x000ffa78 bytes

Nr of chunks : 146

_HEAP_ENTRY psize size unused UserPtr UserSize

00410588 00000 00240 00001 00410590 0000023f (575) (Busy)

004107c8 00240 00020 00008 004107d0 00000018 (24) (Busy)

004107e8 00020 019d0 0000a 004107f0 000019c6 (6598) (Busy)

004121b8 019d0 029e8 0000a 004121c0 000029de (10718) (Busy)

00414ba0 029e8 00048 0000c 00414ba8 0000003c (60) (Busy)

00414be8 00048 00038 00008 00414bf0 00000030 (48) (Busy)

00414c20 00038 00080 00008 00414c28 00000078 (120) (Busy)

00414ca0 00080 00080 00008 00414ca8 00000078 (120) (Busy)

00414d20 00080 00048 0000c 00414d28 0000003c (60) (Busy)

00414d68 00048 00228 00008 00414d70 00000220 (544) (Busy)

00414f90 00228 00050 0000e 00414f98 00000042 (66) (Busy)

00414fe0 00050 00080 00008 00414fe8 00000078 (120) (Busy)

00415060 00080 00018 00008 00415068 00000010 (16) (Busy)

00415078 00018 00050 0000a 00415080 00000046 (70) (Busy)

004150c8 00050 00080 00008 004150d0 00000078 (120) (Busy)

00415148 00080 00018 00008 00415150 00000010 (16) (Busy)

00415160 00018 00018 00008 00415168 00000010 (16) (Busy)

00415178 00018 00040 0000b 00415180 00000035 (53) (Busy)

004151b8 00040 00070 0000c 004151c0 00000064 (100) (Busy)

00415228 00070 00208 00008 00415230 00000200 (512) (Busy)

00415430 00208 00208 00008 00415438 00000200 (512) (Busy)

00415638 00208 00030 0000c 00415640 00000024 (36) (Busy)

00415668 00030 00030 0000c 00415670 00000024 (36) (Busy)

00415698 00030 00038 00008 004156a0 00000030 (48) (Busy)

004156d0 00038 00028 00008 004156d8 00000020 (32) (Busy)

004156f8 00028 00028 00008 00415700 00000020 (32) (Busy)

00415720 00028 00028 00008 00415728 00000020 (32) (Busy)

00415748 00028 00028 00008 00415750 00000020 (32) (Busy)

00415770 00028 00018 00008 00415778 00000010 (16) (Busy)

00415788 00018 00080 00008 00415790 00000078 (120) (Busy)

00415808 00080 00080 00008 00415810 00000078 (120) (Busy)

00415888 00080 00018 00008 00415890 00000010 (16) (Busy)

004158a0 00018 00020 0000c 004158a8 00000014 (20) (Busy)

004158c0 00020 00020 00010 004158c8 00000010 (16) (Busy)

004158e0 00020 00078 0000c 004158e8 0000006c (108) (Busy)

00415958 00078 00080 00008 00415960 00000078 (120) (Busy)

004159d8 00080 00018 00008 004159e0 00000010 (16) (Busy)

004159f0 00018 00018 00008 004159f8 00000010 (16) (Busy)

00415a08 00018 00050 0000e 00415a10 00000042 (66) (Busy)

00415a58 00050 00010 00000 00415a60 00000010 (16) (Free)

00415a68 00010 00058 0000e 00415a70 0000004a (74) (Busy)

00415ac0 00058 00080 00008 00415ac8 00000078 (120) (Busy)

00415b40 00080 00080 00008 00415b48 00000078 (120) (Busy)

00415bc0 00080 00018 00008 00415bc8 00000010 (16) (Busy)

00415bd8 00018 00020 00010 00415be0 00000010 (16) (Busy)

00415bf8 00020 00018 00008 00415c00 00000010 (16) (Busy)

00415c10 00018 00010 00000 00415c18 00000010 (16) (Free)

00415c20 00010 00080 00008 00415c28 00000078 (120) (Busy)

00415ca0 00080 00080 00008 00415ca8 00000078 (120) (Busy)

00415d20 00080 00088 0000c 00415d28 0000007c (124) (Busy)

00415da8 00088 00018 00008 00415db0 00000010 (16) (Busy)

00415dc0 00018 00078 00008 00415dc8 00000070 (112) (Busy)

00415e38 00078 00020 00010 00415e40 00000010 (16) (Busy)

00415e58 00020 00018 00008 00415e60 00000010 (16) (Busy)

00415e70 00018 00080 00008 00415e78 00000078 (120) (Busy)

00415ef0 00080 00018 00008 00415ef8 00000010 (16) (Busy)

00415f08 00018 00018 00008 00415f10 00000010 (16) (Busy)

00415f20 00018 00020 00010 00415f28 00000010 (16) (Busy)

00415f40 00020 00070 00008 00415f48 00000068 (104) (Busy)

00415fb0 00070 00080 00008 00415fb8 00000078 (120) (Busy)

00416030 00080 00018 00008 00416038 00000010 (16) (Busy)

00416048 00018 00028 00008 00416050 00000020 (32) (Busy)

00416070 00028 00080 00008 00416078 00000078 (120) (Busy)

004160f0 00080 00028 00008 004160f8 00000020 (32) (Busy)

00416118 00028 00028 00008 00416120 00000020 (32) (Busy)

00416140 00028 00070 00008 00416148 00000068 (104) (Busy)

004161b0 00070 00028 00008 004161b8 00000020 (32) (Busy)

004161d8 00028 00078 0000e 004161e0 0000006a (106) (Busy)

00416250 00078 03d20 00001 00416258 00003d1f (15647) (Busy)

00419f70 03d20 378b0 00008 00419f90 000378a8 (227496) (Internal,Busy (LFH))

00451820 378b0 00400 00008 00451840 000003f8 (1016) (Internal,Busy (LFH))

00451c20 00400 00400 00008 00451c40 000003f8 (1016) (Internal,Busy (LFH))

00452020 00400 00080 00008 00452028 00000078 (120) (Busy)

004520a0 00080 00080 00008 004520a8 00000078 (120) (Busy)

00452120 00080 00028 00008 00452128 00000020 (32) (Busy)

00452148 00028 00028 00008 00452150 00000020 (32) (Busy)

00452170 00028 00070 0000a 00452178 00000066 (102) (Busy)

004521e0 00070 00080 00008 004521e8 00000078 (120) (Busy)

00452260 00080 00028 00008 00452268 00000020 (32) (Busy)

00452288 00028 00028 00008 00452290 00000020 (32) (Busy)

004522b0 00028 00070 00008 004522b8 00000068 (104) (Busy)

00452320 00070 00078 0000e 00452328 0000006a (106) (Busy)

00452398 00078 00210 00008 004523a0 00000208 (520) (Busy)

004525a8 00210 00028 00008 004525b0 00000020 (32) (Busy)

004525d0 00028 00028 00008 004525d8 00000020 (32) (Busy)

004525f8 00028 00028 00008 00452600 00000020 (32) (Busy)

00452620 00028 00028 00008 00452628 00000020 (32) (Busy)

00452648 00028 00028 00008 00452650 00000020 (32) (Busy)

00452670 00028 00028 00008 00452678 00000020 (32) (Busy)

00452698 00028 00078 0000c 004526a0 0000006c (108) (Busy)

00452710 00078 02000 00008 00452730 00001ff8 (8184) (Internal,Busy (LFH))

00454710 02000 00070 0000a 00454718 00000066 (102) (Busy)

00454780 00070 00078 0000e 00454788 0000006a (106) (Busy)

004547f8 00078 00408 00008 00454800 00000400 (1024) (Busy)

00454c00 00408 00800 00008 00454c20 000007f8 (2040) (Internal,Busy (LFH))

00455400 00800 006d0 00008 00455408 000006c8 (1736) (Busy)

00455ad0 006d0 00c08 00008 00455ad8 00000c00 (3072) (Busy)

004566d8 00c08 00800 00008 004566f8 000007f8 (2040) (Internal,Busy (LFH))

00456ed8 00800 00228 00008 00456ee0 00000220 (544) (Busy)

00457100 00228 00228 00008 00457108 00000220 (544) (Busy)

00457328 00228 004c0 00008 00457330 000004b8 (1208) (Busy)

004577e8 004c0 00038 0000f 004577f0 00000029 (41) (Busy)

00457820 00038 00098 0000d 00457828 0000008b (139) (Busy)

004578b8 00098 00020 00009 004578c0 00000017 (23) (Busy)

004578d8 00020 00020 00008 004578e0 00000018 (24) (Busy)

004578f8 00020 00030 0000f 00457900 00000021 (33) (Busy)

00457928 00030 00020 0000c 00457930 00000014 (20) (Busy)

00457948 00020 00020 0000a 00457950 00000016 (22) (Busy)

00457968 00020 00030 00008 00457970 00000028 (40) (Busy)

00457998 00030 00030 00009 004579a0 00000027 (39) (Busy)

004579c8 00030 00060 0000e 004579d0 00000052 (82) (Busy)

00457a28 00060 00048 00008 00457a30 00000040 (64) (Busy)

00457a70 00048 00020 0000e 00457a78 00000012 (18) (Busy)

00457a90 00020 00058 0000e 00457a98 0000004a (74) (Busy)

00457ae8 00058 00808 00008 00457af0 00000800 (2048) (Busy)

004582f0 00808 00088 00008 004582f8 00000080 (128) (Busy)

00458378 00088 00048 0000b 00458380 0000003d (61) (Busy)

004583c0 00048 00448 00008 004583c8 00000440 (1088) (Busy)

00458808 00448 00078 0000c 00458810 0000006c (108) (Busy)

00458880 00078 01168 00000 00458888 00001168 (4456) (Free)

004599e8 01168 000f0 0000c 004599f0 000000e4 (228) (Busy)

00459ad8 000f0 00038 0000a 00459ae0 0000002e (46) (Busy)

00459b10 00038 00030 00008 00459b18 00000028 (40) (Busy)

00459b40 00030 00040 00009 00459b48 00000037 (55) (Busy)

00459b80 00040 00048 0000c 00459b88 0000003c (60) (Busy)

00459bc8 00048 00040 0000f 00459bd0 00000031 (49) (Busy)

00459c08 00040 00030 0000c 00459c10 00000024 (36) (Busy)

00459c38 00030 00020 00009 00459c40 00000017 (23) (Busy)

00459c58 00020 00040 0000e 00459c60 00000032 (50) (Busy)

00459c98 00040 00038 0000a 00459ca0 0000002e (46) (Busy)

00459cd0 00038 00038 0000c 00459cd8 0000002c (44) (Busy)

00459d08 00038 00030 00008 00459d10 00000028 (40) (Busy)

00459d38 00030 00030 0000f 00459d40 00000021 (33) (Busy)

00459d68 00030 00020 0000b 00459d70 00000015 (21) (Busy)

00459d88 00020 00038 0000d 00459d90 0000002b (43) (Busy)

00459dc0 00038 00030 0000e 00459dc8 00000022 (34) (Busy)

00459df0 00030 00038 0000a 00459df8 0000002e (46) (Busy)

00459e28 00038 00048 0000f 00459e30 00000039 (57) (Busy)

00459e70 00048 00020 00009 00459e78 00000017 (23) (Busy)

00459e90 00020 00040 0000a 00459e98 00000036 (54) (Busy)

00459ed0 00040 00050 00009 00459ed8 00000047 (71) (Busy)

00459f20 00050 00050 00008 00459f28 00000048 (72) (Busy)

00459f70 00050 00020 0000e 00459f78 00000012 (18) (Busy)

00459f90 00020 00020 00008 00459f98 00000018 (24) (Busy)

00459fb0 00020 00030 0000c 00459fb8 00000024 (36) (Busy)

00459fe0 00030 00020 00003 00459fe8 0000001d (29) (Busy)

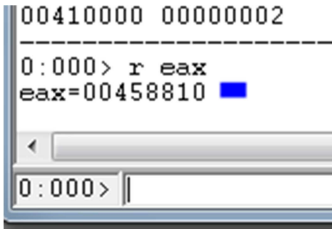
0x00459ff8 - 0x00510000 (end of segment) : 0xb6008 (745480) uncommitted bytes

Heap : 0x00410000 [LFH] : VirtualAllocdBlocks : 0

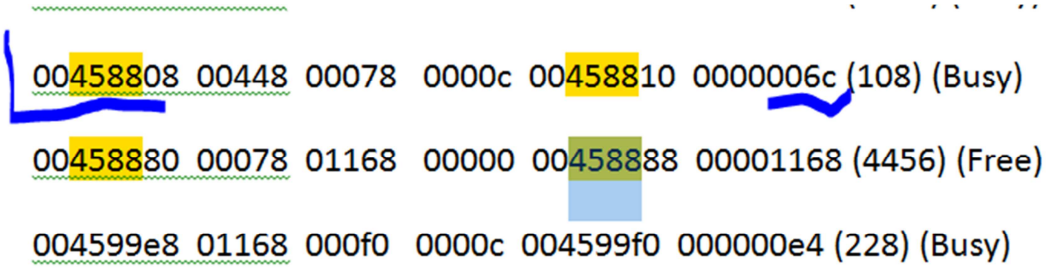
Nr of chunks : 0

[+] This mona.py action took 0:00:05.043000

Remember that in this point, EAX had the block user address (without the header)



If we find by 4588 because in the lists it is always by that address that includes the header.



There, we see the BUSY block, the user size 0x6c and the total size 0x78 (120 decimal)

If we did it with Windbg using **!heap -a 0x00410000**.

We see that the info is similar.

As before in Windbg, we see the block info.

```
00458000:      000b6000      - uncommitted bytes.
0:000> !heap -p -a  eax
    address 00458810 found in
    _HEAP @ 410000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        00458808 000f 0000  [00]   00458810    0006c - (busy)
```

The size was multiplied by 8 to see the total.

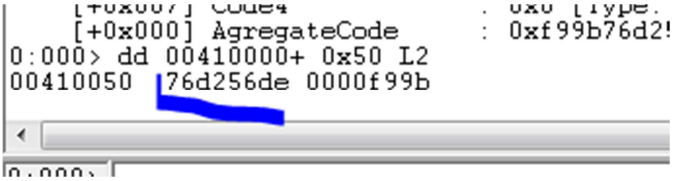
hex(0xf *0x8)

'0x78' or 120 decimal.

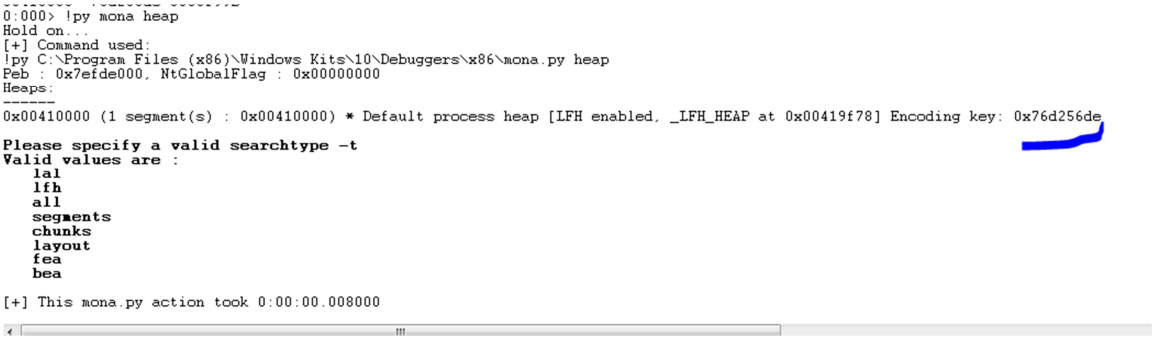
In Mona:

```
0:000> dt _HEAP 410000
ntdll!_HEAP
   +0x000 Entry            : _HEAP_ENTRY
   +0x008 SegmentSignature : 0xffeeffee
   +0x00c SegmentFlags     : 0
   +0x010 SegmentListEntry : _LIST_ENTRY [ 0x4100a8 - 0x4100a8 ]
   +0x018 Heap             : 0x00410000 _HEAP
   +0x01c BaseAddress      : 0x00410000 Void
   +0x020 NumberOfPages    : 0x100
   +0x024 FirstEntry       : 0x00410588 _HEAP_ENTRY
   +0x028 LastValidEntry   : 0x00510000 _HEAP_ENTRY
   +0x02c NumberOfUnCommittedPages : 0xb6
   +0x030 NumberOfUnCommittedRanges : 1
   +0x034 SegmentAllocatorBackTraceIndex : 0
   +0x036 Reserved         : 0
   +0x038 UCRSegmentList   : _LIST_ENTRY [ 0x459ff0 - 0x459ff0 ]
   +0x040 Flags            : 2
   +0x044 ForceFlags       : 0
   +0x048 CompatibilityFlags : 0
   +0x04c EncodeFlagMask   : 0x100000
   +0x050 Encoding         : _HEAP_ENTRY
   +0x058 PointerKey       : 0x6e76c3fe
   +0x05c Interceptor      : 0
   +0x060 VirtualMemoryThreshold : 0xfe00
   +0x064 Signature        : 0xeeffeeff
   +0x068 SegmentReserve   : 0x100000
   +0x06c SegmentCommit    : 0x2000
   +0x070 DeCommitFreeBlockThreshold : 0x800
   +0x074 DeCommitTotalFreeThreshold : 0x2000
   +0x078 TotalFreeSize    : 0x231
   +0x07c MaximumAllocationSize : 0x7ffdefff
   +0x080 ProcessHeapsListIndex : 1
   +0x082 HeaderValidateLength : 0x138
   +0x084 HeaderValidateCopy : (null)
   +0x088 NextAvailableTagIndex : 0
   +0x08a MaximumTagIndex  : 0
   +0x08c TagEntries       : (null)
   +0x090 UCRList          : _LIST_ENTRY [ 0x459fe8 - 0x459fe8 ]
   +0x098 AlignRound       : 0xf
   +0x09c AlignMask        : 0xfffffff8
   +0x0a0 VirtualAllocdBlocks : _LIST_ENTRY [ 0x4100a0 - 0x4100a0 ]
   +0x0a8 SegmentList      : _LIST_ENTRY [ 0x410010 - 0x410010 ]
   +0x0b0 AllocatorBackTraceIndex : 0
```

As always in position 0x50, we have the keys to xor them. Let's see them.

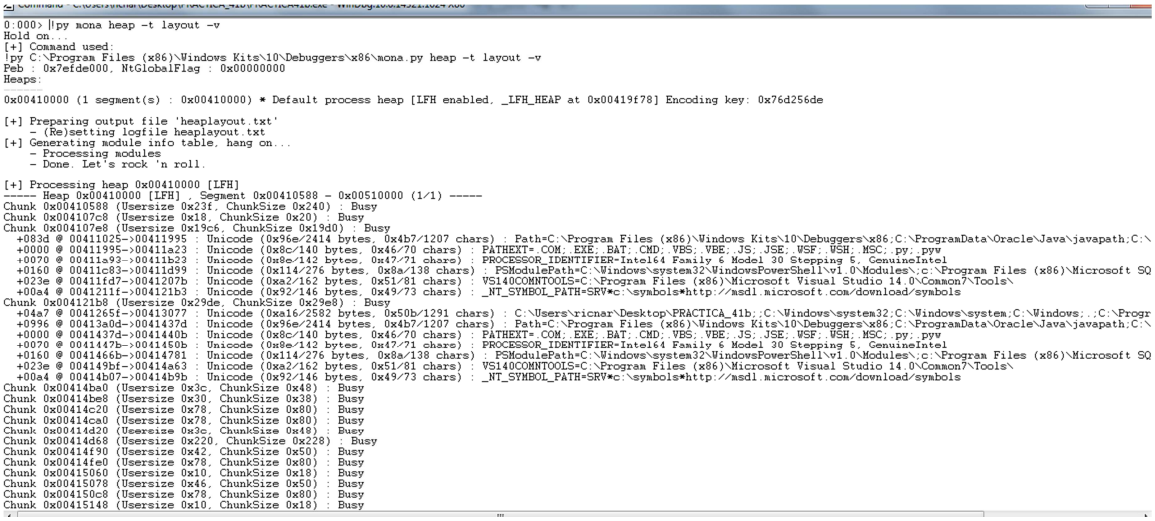


We see that one of the two keys matches the one shown by Mona.

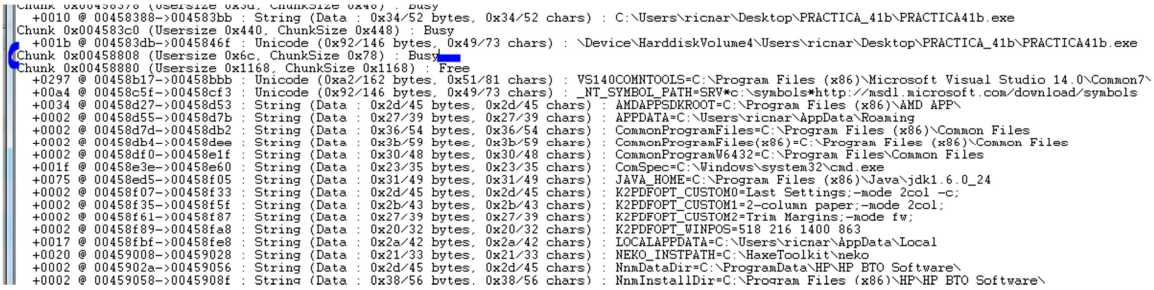


We have a command in Mona that I don't know if Windbg has it.

```
!py mona heap -t layout -v
```



The output is too long, but it tries to see for what it uses the heap blocks and lists them.



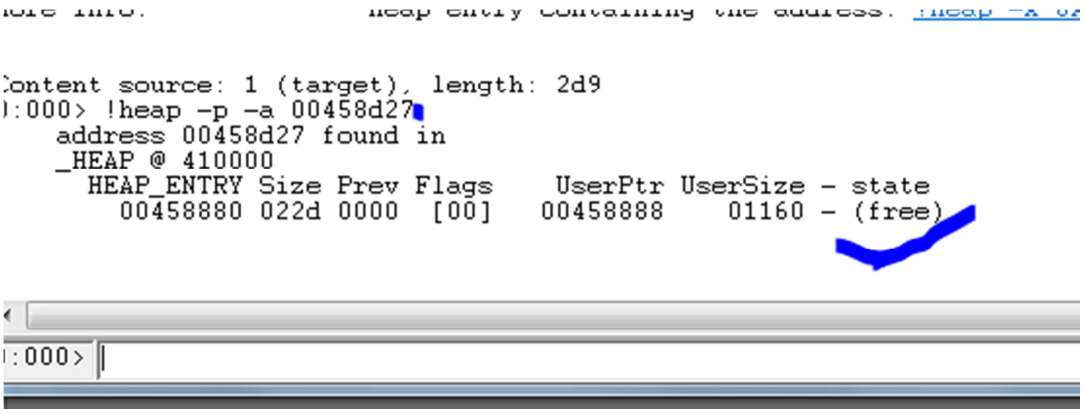
There, it is our block.

If I see the content of some of the others…

0:000> da 00458d27

00458d27 "AMDAPPSDKROOT=C:\Program Files ("…..

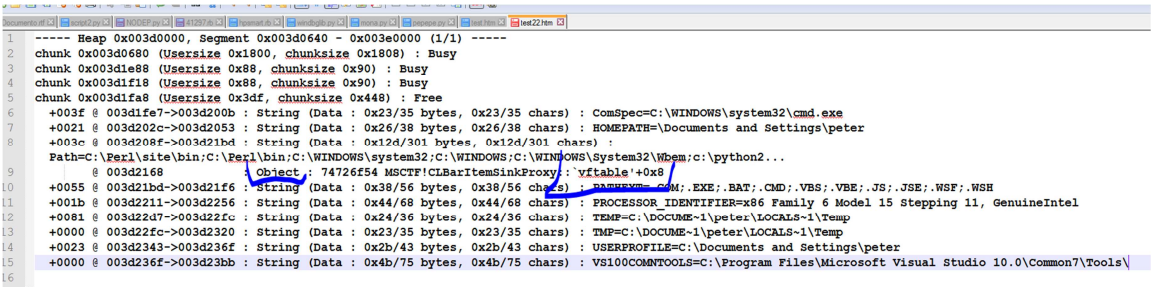
We see that the content is the same shown in the list.



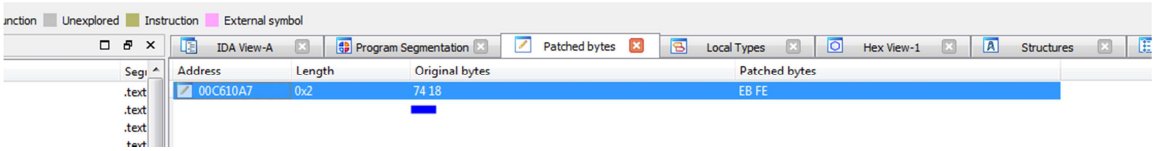
The data match. It is free. This means it saved info there, but the block was freed for a new use.

Not in this case, but we have to pay attention to the allocated objects because they have vtables that are virtual tables that can be stepped and could make us jump to control the execution.

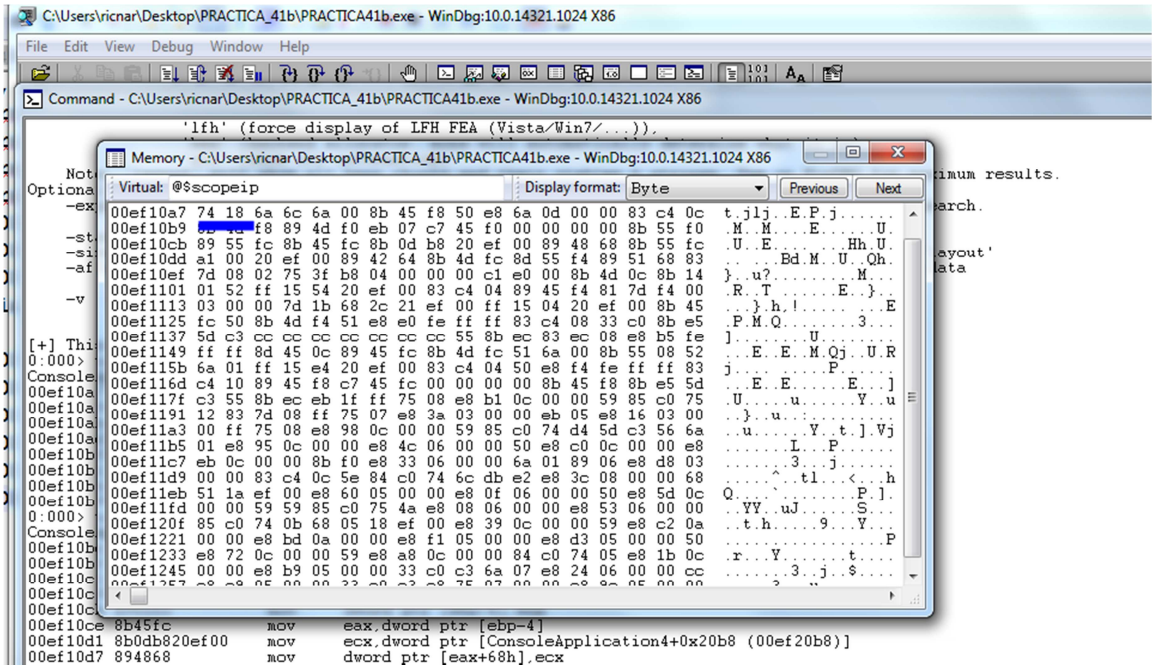
This is an example of the WEB for you to see that it says OBJECT and VFTABLE there. A good target to step if there is overflow.



In IDA, we can see the bytes we had changed to set the infinite loop with **EDIT-PATCHED BYTES.**



In Windbg memory tab, I go to the address where the **EB FE** are and I change them by **74 18**.



If I now do **u eip**, I see that it changes to the conditional jump it had.

```
0ef10d1 8bdd820ef00     mov     ecx,dword ptr [ConsoleApplication4+0x20d8 (00ef20d8)]
0ef10d7 894868          mov     dword ptr [eax+68h],ecx
:000> u eip
onsoleApplication4+0x10a7:
0ef10a7 7418            je      ConsoleApplication4+0x10c1 (00ef10c1)
0ef10a9 [redacted]      push    6Ch
0ef10ab 6a00            push    0
0ef10ad 8b45f8          mov     eax,dword ptr [ebp-8]
0ef10b0 50              push    eax
0ef10b1 e86a0d0000      call    ConsoleApplication4+0x1e20 (00ef1e20)
0ef10b6 83c40c          add     esp,0Ch
0ef10b9 8b4df8          mov     ecx,dword ptr [ebp-8]
:000>
```

I can press RUN o G and accept the script enter and that it continues until crashing. We know that **page heap** is not set as **full**. So, there is no history and it won't crash when writing. It will crash when jumping to execute.

I ran it again because I had to restart the PC. I will get to the thing as before although the addresses will be different.

```
ntdll!DbgBreakPoint:
7757000c cc              int     3
0:001> lm
start    end        module name
00ef0000 00ef7000   PRACTICA41b   (deferred)
6d280000 6d285000   api_ms_win_crt_math_l1_1_0   (deferred)
6d290000 6d293000   api_ms_win_crt_locale_l1_1_0   (deferred)
6d2a0000 6d2a4000   api_ms_win_crt_convert_l1_1_0   (deferred)
6d2b0000 6d2b4000   api_ms_win_crt_stdio_l1_1_0   (deferred)
6d2c0000 6d2c3000   api_ms_win_crt_heap_l1_1_0   (deferred)
6d2d0000 6d2d4000   api_ms_win_crt_string_l1_1_0   (deferred)
6d2e0000 6d2e3000   api_ms_win_core_file_l1_2_0   (deferred)
6d2f0000 6d2f3000   api_ms_win_core_processthreads_l1_1_1   (deferred)
6d300000 6d303000   api_ms_win_core_localization_l1_2_0   (deferred)
6d310000 6d3f1000   ucrtbase   (deferred)
6d9e0000 6d9e3000   api_ms_win_core_file_l2_1_0   (deferred)
6da60000 6da63000   api_ms_win_core_timezone_l1_1_0   (deferred)
6da70000 6da74000   api_ms_win_crt_runtime_l1_1_0   (deferred)
6da80000 6da95000   VCRUNTIME140   (deferred)
742b0000 742b3000   api_ms_win_core_synch_l1_2_0   (deferred)
75a10000 75b20000   kernel32   (deferred)
75fa0000 75fe7000   KERNELBASE   (deferred)
77560000 776e0000   ntdll      (pdb symbols)          c:\symbols\wntdll.pdb\64A5F447CE044B55A80F5E817890D5732\wntdll.pdb
0:001>
```

This time, it did appear the name. ☺

```
77560000 776e0000   ntdll      (pdb symbols)          c:\symbols\wntdll.pdb\64A5F447CE044B55A80F5E817890D5732\wntdll.pdb
0:001> u ef10a7
*** WARNING: Unable to verify checksum for C:\Users\ricnar\Desktop\PRACTICA_41b\PRACTICA41b.exe
*** ERROR: Module load completed but symbols could not be loaded for C:\Users\ricnar\Desktop\PRACTICA_41b\PRACTICA41b.ex
PRACTICA41b+0x10a7:
00ef10a7 ebfe            jmp     PRACTICA41b+0x10a7 (00ef10a7)
00ef10a9 6a6c            push    6Ch
00ef10ab 6a00            push    0
00ef10ad 8b45f8          mov     eax,dword ptr [ebp-8]
00ef10b0 50              push    eax
00ef10b1 e86a0d0000      call    PRACTICA41b+0x1e20 (00ef1e20)
00ef10b6 83c40c          add     esp,0Ch
00ef10b9 8b4df8          mov     ecx,dword ptr [ebp-8]
0:001>
```

I see the heap with Mona.

0:000> !py mona heap -a

Hold on...

[+] Command used:

!py C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\mona.py heap -a

Peb : 0x7efde000, NtGlobalFlag : 0x00000000

Heaps:

0x004f0000 (1 segment(s) : 0x004f0000) * Default process heap [LFH enabled, _LFH_HEAP at 0x004fb348] Encoding key: 0x13f60872

And I see the blocks.

!py mona.py heap -h 0x004f0000 -t chunks

Y EAX vale

0:000> r eax

EAX=0053a720

So, the block is now:

0053a718 00448 00078   0000c 0053a720 0000006c (108) (Busy)

0053a790 00078 01728   00000 0053a798 00001728 (5928) (Free)

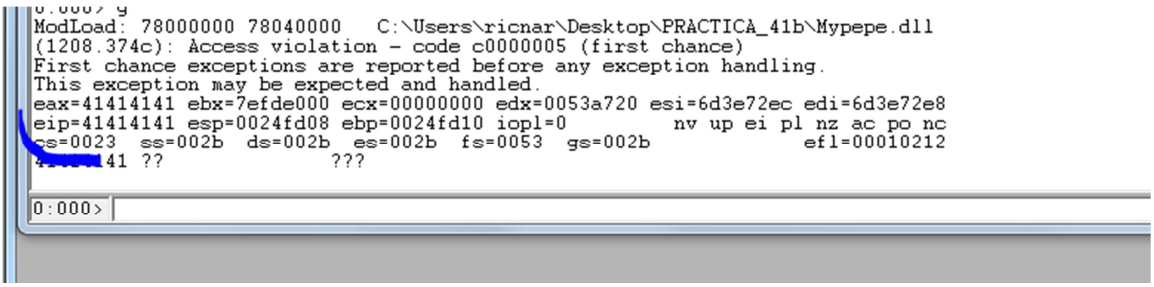
0053beb8 01728 00128   00014 0053bec0 00000114 (276) (Busy)

0053bfe0 00128 00020   00003 0053bfe8 0000001d (29) (Busy)

I change the infinite loop by the conditional jump.

00ef10a7 7418   je   PRACTICA41b+0x10c1 (00ef10c1)

I press G and then script ENTER to continue.



It jumped to execute. Let’s see what it tells us about the block.

```
0053a248  00808  00088   00008  0053a250  00000080 (128) (Busy)
0053a2d0  00088  00448   00008  0053a2d8  00000440 (1088) (Busy)
0053a718  00448  00078   0000c  0053a720  0000006c (108) (Busy)
0053a790  00078  01728   00000  0053a798  00001728 (5928) (Free)
0053beb8  01728  00128   00014  0053bec0  00000114 (276) (Busy)
0053bfe0  00128  00020   00003  0053bfe8  0000001d (29) (Busy)
0x0053bff8 - 0x005f0000 (end of segment) : 0xb4008 (737288) uncommitted bytes

Heap : 0x004f0000 [LFH]  : VirtualAllocdBlocks : 0
    Nr of chunks : 0
```

The same thing. Let’s see the layout.

```
 +0182 @ 0053a417->0053a51b : String (Data : 0x25/37 bytes, 0x25/37 chars) : xDesktop\PRACTICA_41b\PRACTICA41b.exe
 +001c @ 0053a537->0053a719 : Unicode (0x1e0/480 bytes, 0xf0/240 chars) : ments\COMMIT\2017\components\src\modules\classic\modules\shippal
Chunk 0x0053a718 (Usersize 0x6c, ChunkSize 0x78) : Busy
 +0008 @ 0053a720->0053a78f : String (Data : 0x70/112 bytes, 0x70/112 chars) : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Chunk 0x00[illegible] (Usersize 0x24957, ChunkSize 0x24998) : No Coalesce,FFU-1,Last,Fill pattern,Extra present,Busy
 +0000 @ 0053a790->0053ab07 : String (Data : 0x378/888 bytes, 0x378/888 chars) : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 +04b8 @ 0053afbf->0053b10d : Unicode (0x14c/332 bytes, 0xa6/166 chars) : \Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program F
 +001a @ 0053b127->0053bb2d : Unicode (0xa04/2564 bytes, 0x502/1282 chars) : C:\Users\ricnar\Desktop\PRACTICA_41b;C:\Windows\system32;C:\U
 +004a @ 0053bb77->0053bc37 : Unicode (0xbe/190 bytes, 0x5f/95 chars) :  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`a
 +013c @ 0053bd73->0053bd99 : String (Data : 0x27/39 bytes, 0x27/39 chars) : TEMP=C:\Users\ricnar\AppData\Local\Temp
 +0002 @ 0053bd9b->0053bdc0 : String (Data : 0x26/38 bytes, 0x26/38 chars) : TMP=C:\Users\ricnar\AppData\Local\Temp
 +022a @ 0053bfea->0053c3d0 : String (Data : 0x3e7/999 bytes, 0x3e7/999 chars) : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 +0001 @ 0053c3d1->0053c46d : Unicode (0x9a/154 bytes, 0x4d/77 chars) : AS=C:\Users\ricnar\AppData\Roaming\IMPACT\components\modules\camer
```

It shows us the block full of A’s and the next one too with A’s. Windbg shows me the next block with size 0x4141. The size looks corrupted.

```
Content source: 1 (target), length: 8e8
0:000> !heap -p -a 53a718
    address 0053a718 found in
    _HEAP @ 4f0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0053a718 000f 0000  [00]   0053a720    0006c - (busy)

0:000> !heap -p -a 53a790
    address 0053a790 found in
    _HEAP @ 4f0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        0053a790 4141 0000  [00]   0053a798    1c8c7 - (busy)

0:000>
```

If we use –x, Windbg will tell us if it is corrupted.

```
0:000> !heap -x 53a718
List corrupted: (Flink->Blink = 41414141) != (Block = 004f6f20)
HEAP 004f0000 (Seg 004f0000) At 004f6f18 Error: block list entry corrupted

ERROR: Block 0053a790 previous size 83d2 does not match previous block size f
HEAP 004f0000 (Seg 004f0000) At 0053a790 Error: invalid block Previous

Entry     User      Heap      Segment       Size  PrevSize  Unused    Flags
-----------------------------------------------------------------------------
0053a718  0053a720  004f0000  004f0000        78       448         c  busy

0:000> !heap -x 53a790
List corrupted: (Flink->Blink = 41414141) != (Block = 004f6f20)
HEAP 004f0000 (Seg 004f0000) At 004f6f18 Error: block list entry corrupted

ERROR: Block 0053a790 previous size 83d2 does not match previous block size f
HEAP 004f0000 (Seg 004f0000) At 0053a790 Error: invalid block Previous
```

Windbg gives us much info. Mona a little more. It has some commands to work with objects that we can't use yet. All this will help us to practice and solve the pending exercise from part 44. We'll see it in the next part.

Ricardo Narvaja

Translated by: @IvinsonCLS