# REVERSING WITH IDA PRO FROM SCRATCH

## PART 1

The idea of these series of tutorials is updating our original reversing course but using IDA PRO. Learning how to use it from scratch and work with the last Windows versions. In this case, I'm using Windows 10 Anniversary Update 64 bits with all the patches until October 29, 2016.

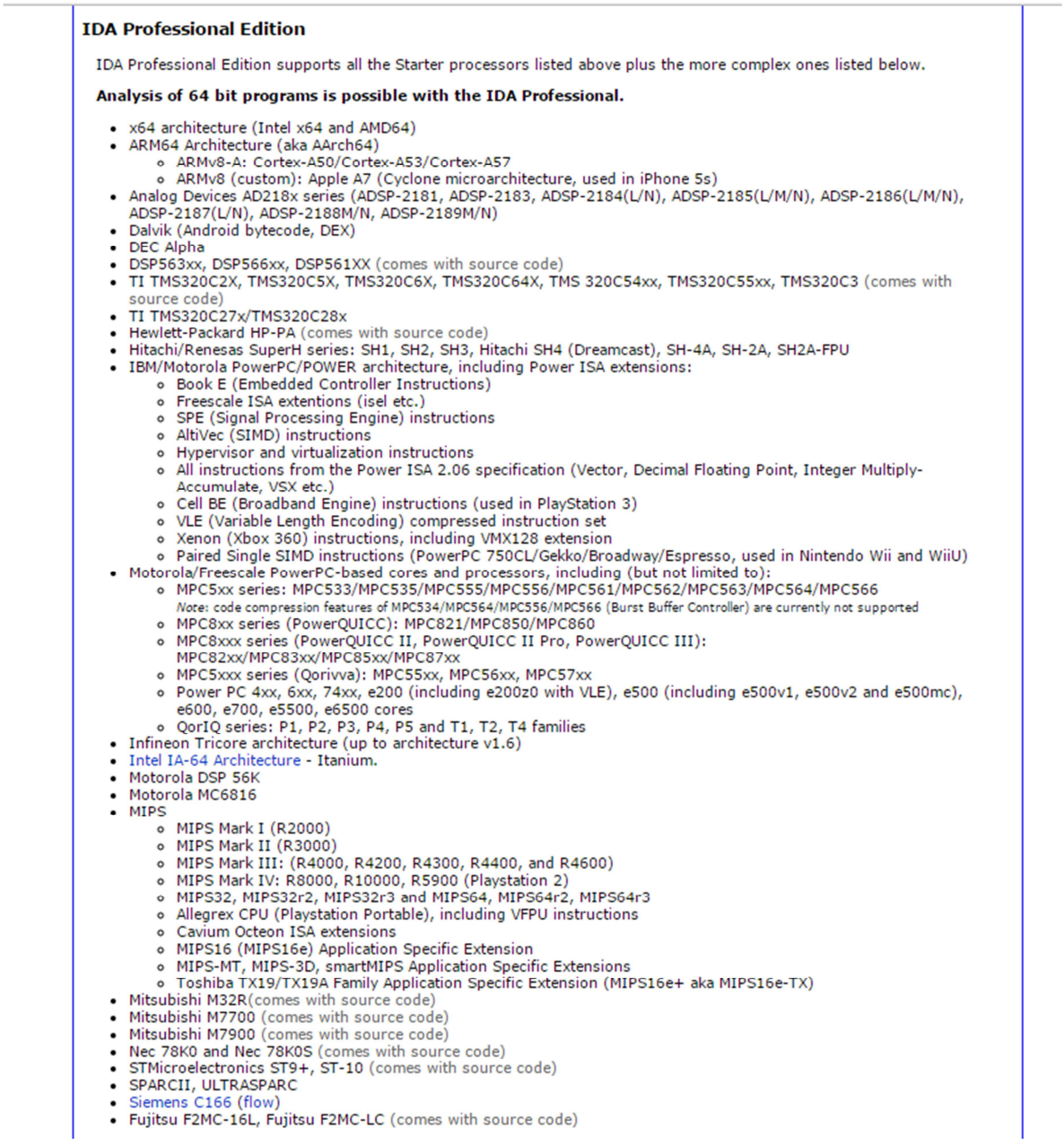
## WHY IDA PRO?

Because while OllyDBG is just a 32 bit debugger in Windows User Mode, IDA PRO is a whole reversing tool that can be used in 32/64bits as a disassembler and debugger. It permits static reversing which can't be done in OllyDBG and who learns how to use it, in spite of having a more complex learning curve, it allows him/her to work in Windows, Linux or Mac OS X natively and remotely in the following Operating Systems:

The following debugger servers are shipped with IDA

| File name | Target system | Debugged programs |
|---|---|---|
| android_server | ARM Android | 32-bit ELF files |
| armlinux_server | ARM Linux | 32-bit ELF files |
| armuclinux_server | ARM UCLinux | 32-bit ELF files |
| linux_server | Linux 32-bit | 32-bit ELF files |
| linux_serverx64 | Linux 64-bit | 64-bit ELF files |
| mac_server | Mac OS X | 32-bit Mach-O files |
| mac_serverx64 | Mac OS X | 64-bit Mach-O files |
| win32_remote.exe | MS Windows 32-bit | 32-bit PE files |
| win64_remotex64.exe | MS Windows 64-bit | 64-bit PE files |
| wince_remote.dll | Windows CE | 32-bit PE files |

To have an idea of the supported processors, here there is a list:



**IDA Professional Edition**

IDA Professional Edition supports all the Starter processors listed above plus the more complex ones listed below.

**Analysis of 64 bit programs is possible with the IDA Professional.**

- x64 architecture (Intel x64 and AMD64)
- ARM64 Architecture (aka AArch64)
  - ARMv8-A: Cortex-A50/Cortex-A53/Cortex-A57
  - ARMv8 (custom): Apple A7 (Cyclone microarchitecture, used in iPhone 5s)
- Analog Devices AD218x series (ADSP-2181, ADSP-2183, ADSP-2184(L/N), ADSP-2185(L/M/N), ADSP-2186(L/M/N), ADSP-2187(L/N), ADSP-2188M/N, ADSP-2189M/N)
- Dalvik (Android bytecode, DEX)
- DEC Alpha
- DSP563xx, DSP566xx, DSP561XX (comes with source code)
- TI TMS320C2X, TMS320C5X, TMS320C6X, TMS320C64X, TMS 320C54xx, TMS320C55xx, TMS320C3 (comes with source code)
- TI TMS320C27x/TMS320C28x
- Hewlett-Packard HP-PA (comes with source code)
- Hitachi/Renesas SuperH series: SH1, SH2, SH3, Hitachi SH4 (Dreamcast), SH-4A, SH-2A, SH2A-FPU
- IBM/Motorola PowerPC/POWER architecture, including Power ISA extensions:
  - Book E (Embedded Controller Instructions)
  - Freescale ISA extentions (isel etc.)
  - SPE (Signal Processing Engine) instructions
  - AltiVec (SIMD) instructions
  - Hypervisor and virtualization instructions
  - All instructions from the Power ISA 2.06 specification (Vector, Decimal Floating Point, Integer Multiply-Accumulate, VSX etc.)
  - Cell BE (Broadband Engine) instructions (used in PlayStation 3)
  - VLE (Variable Length Encoding) compressed instruction set
  - Xenon (Xbox 360) instructions, including VMX128 extension
  - Paired Single SIMD instructions (PowerPC 750CL/Gekko/Broadway/Espresso, used in Nintendo Wii and WiiU)
- Motorola/Freescale PowerPC-based cores and processors, including (but not limited to):
  - MPC5xx series: MPC533/MPC535/MPC555/MPC556/MPC561/MPC562/MPC563/MPC564/MPC566
    Note: code compression features of MPC534/MPC564/MPC556/MPC566 (Burst Buffer Controller) are currently not supported
  - MPC8xx series (PowerQUICC): MPC821/MPC850/MPC860
  - MPC8xxx series (PowerQUICC II, PowerQUICC II Pro, PowerQUICC III): MPC82xx/MPC83xx/MPC85xx/MPC87xx
  - MPC5xxx series (Qorivva): MPC55xx, MPC56xx, MPC57xx
  - Power PC 4xx, 6xx, 74xx, e200 (including e200z0 with VLE), e500 (including e500v1, e500v2 and e500mc), e600, e700, e5500, e6500 cores
  - QorIQ series: P1, P2, P3, P4, P5 and T1, T2, T4 families
- Infineon Tricore architecture (up to architecture v1.6)
- Intel IA-64 Architecture - Itanium.
- Motorola DSP 56K
- Motorola MC6816
- MIPS
  - MIPS Mark I (R2000)
  - MIPS Mark II (R3000)
  - MIPS Mark III: (R4000, R4200, R4300, R4400, and R4600)
  - MIPS Mark IV: R8000, R10000, R5900 (Playstation 2)
  - MIPS32, MIPS32r2, MIPS32r3 and MIPS64, MIPS64r2, MIPS64r3
  - Allegrex CPU (Playstation Portable), including VFPU instructions
  - Cavium Octeon ISA extensions
  - MIPS16 (MIPS16e) Application Specific Extension
  - MIPS-MT, MIPS-3D, smartMIPS Application Specific Extensions
  - Toshiba TX19/TX19A Family Application Specific Extension (MIPS16e+ aka MIPS16e-TX)
- Mitsubishi M32R(comes with source code)
- Mitsubishi M7700 (comes with source code)
- Mitsubishi M7900 (comes with source code)
- Nec 78K0 and Nec 78K0S (comes with source code)
- STMicroelectronics ST9+, ST-10 (comes with source code)
- SPARCII, ULTRASPARC
- Siemens C166 (flow)
- Fujitsu F2MC-16L, Fujitsu F2MC-LC (comes with source code)

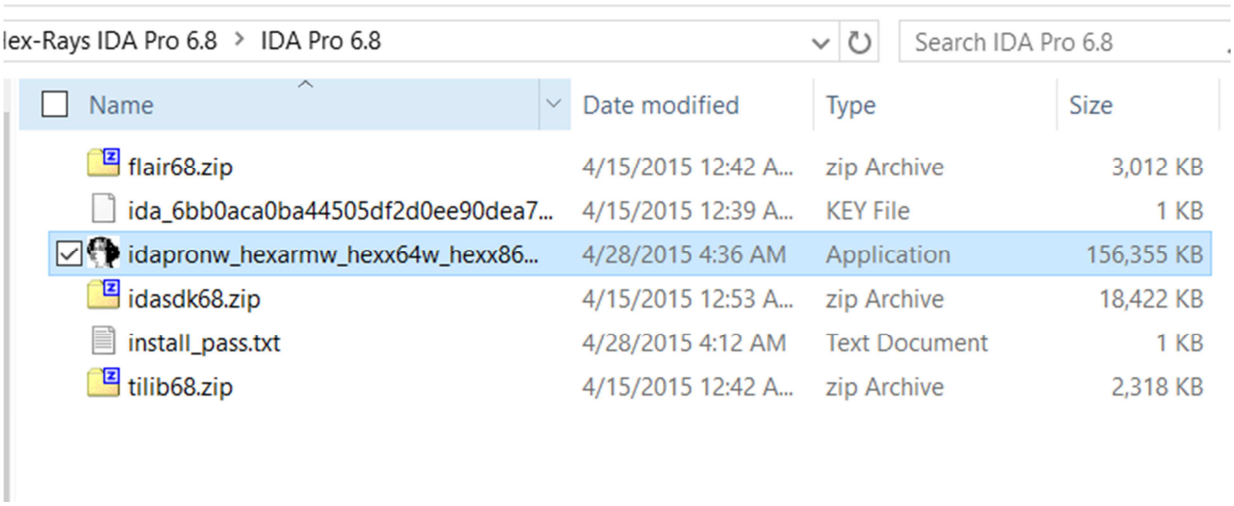
As we can see, learning how to use IDA allows us to improve our working universe although here, we will focus on 32/64-bit Windows in User Mode and sometimes in Kernel Mode. This will let us to adapt to any use easily.

Here, we will see most of the things we saw in The Introduction to OllyDBG, but in IDA. Trying to go further away from the beginning.

There will be everything: Static and dynamic reversing, cracking, exploiting and unpacking. I will try to write about  any important detail from scratch.

## FIRST THINGS FIRST

First, we need IDA PRO. The problem is that it is a commercial program and we should pay for it and it is worth it. We cannot redistribute it, but you can search for the leaked version on Google: IDA PRO 6.8 + HEXRAYS. That is the version which we will work with.

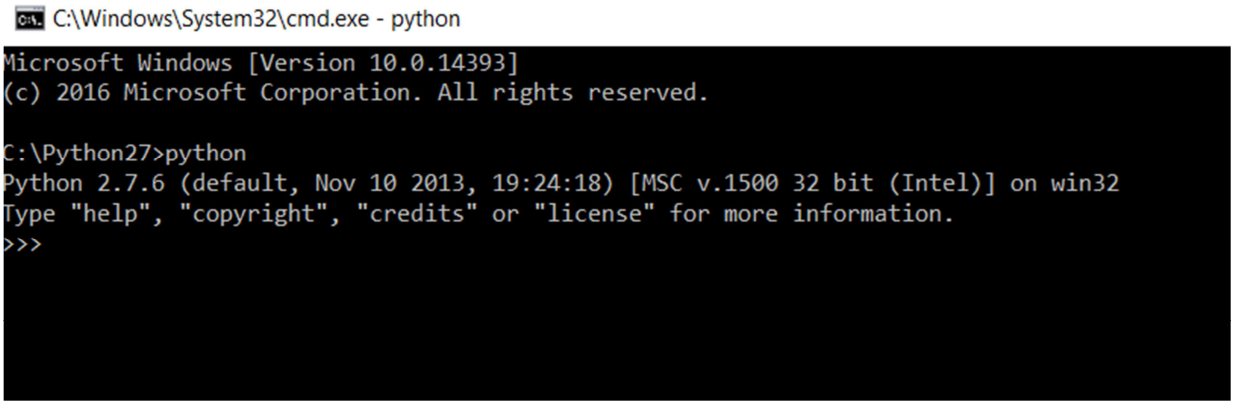



There, we can see the zip files we downloaded and the Setup is:

idapronw_hexarmw_hexx64w_hexx86w_150413_cb5d8b3937caf856aaae750455d2b4ae

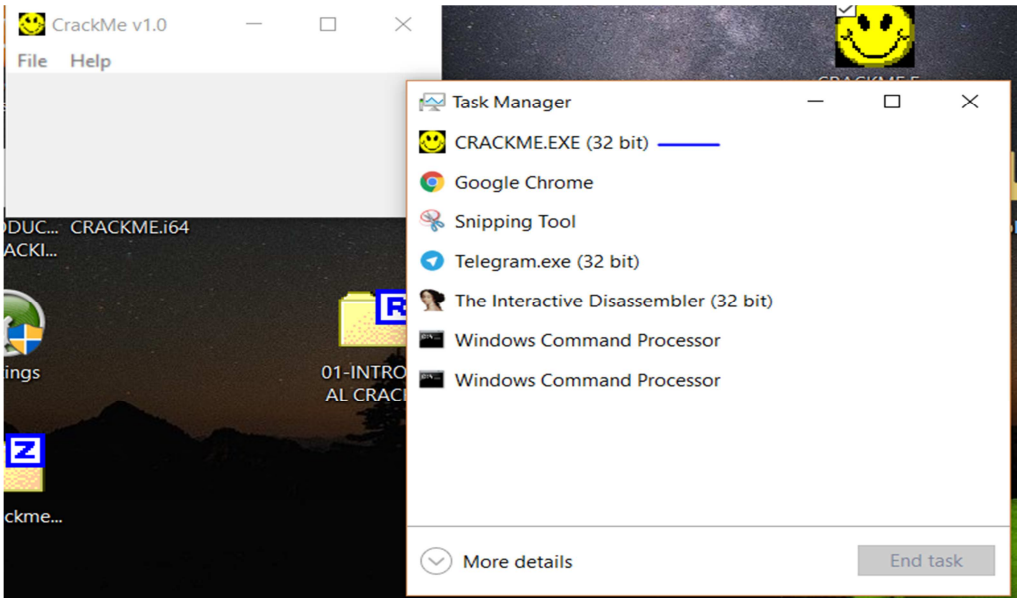
The installation password is in the file **install_pass.txt**.

It will also install Python 2.7.6. It is recommended to avoid problems using the included IDA version. If you install Python standalone, it has to be the same version IDA uses.




After installing IDA, we load the Cruehead crackme included in the tutorial.


As it is a 32-bit executable, we load it with IDA for 32 bits from a direct access.



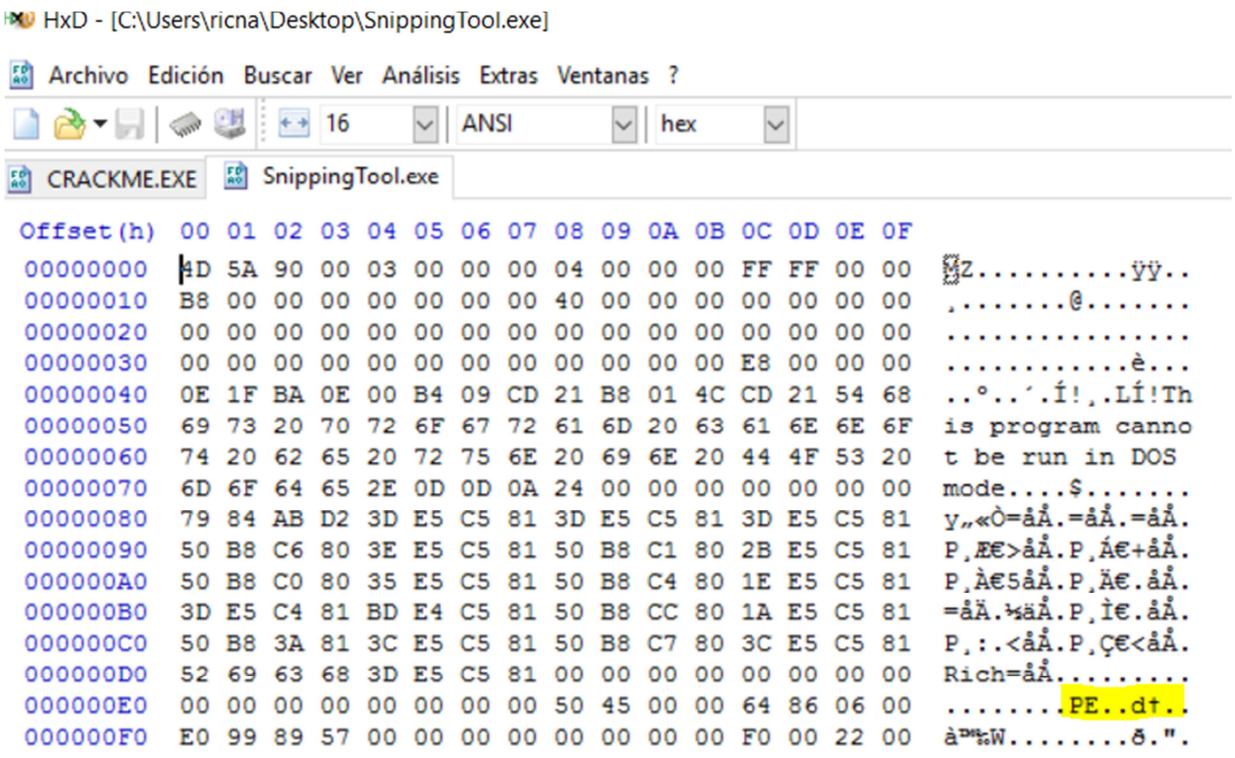
If we run the crackme outside IDA, we see in the Task Manager that it is a 32-bit process. If we want to know if it is a 32 or 64 bits without executing it, we can use an hexadecimal editor. For example:

https://mh-nexus.de/en/downloads.php?product=HxD

Download and install hxd in English.

http://files.downloadnow.com/s/software/11/01/89/28/HxDSetupEN.zip?token=1478941265_7980da68bc96694a971a3be076392a0f&fileName=HxDSetupEN.zip

An easy way to open a file in a hex editor to know if it is 32 or 64 bits is this one:

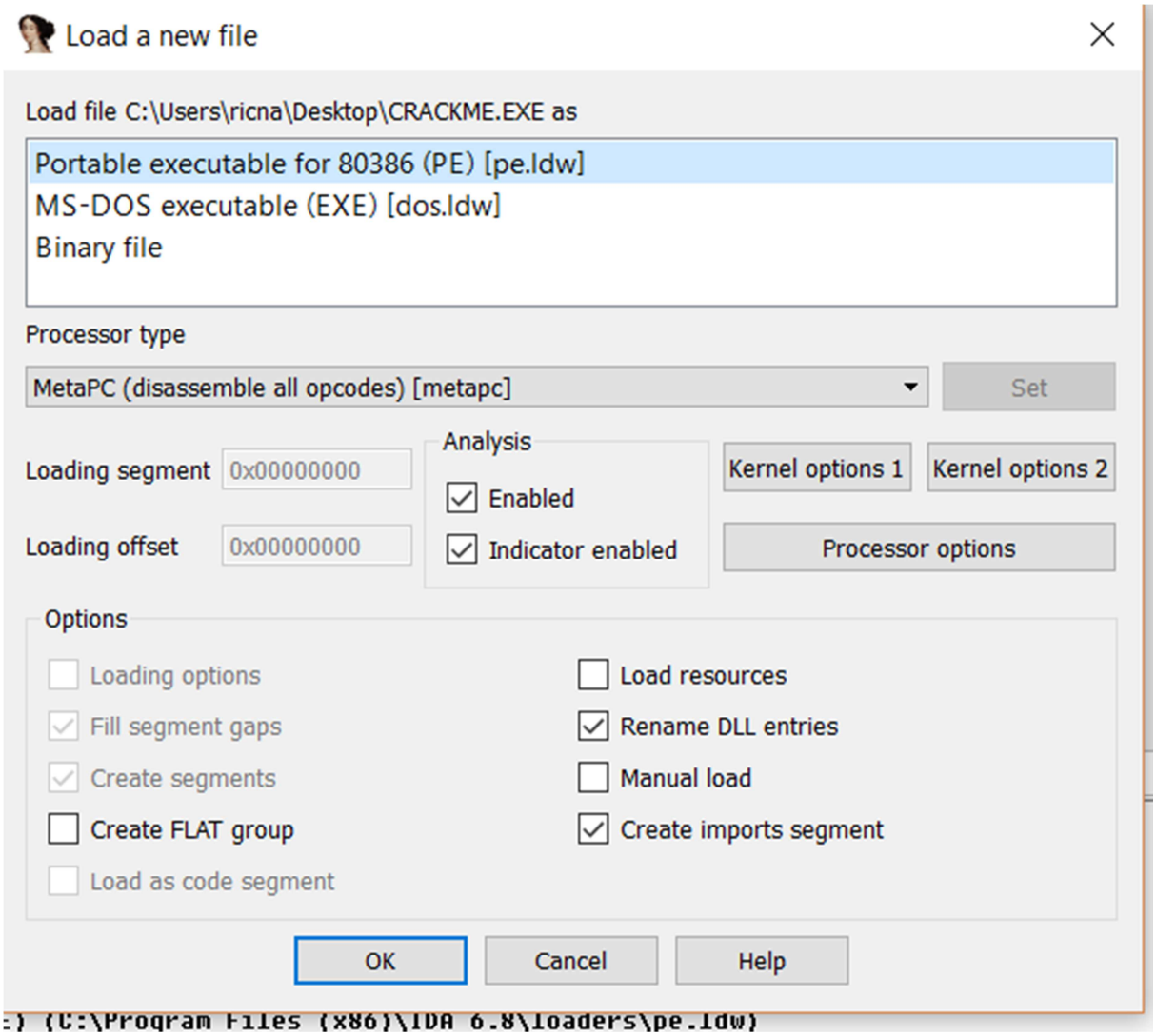



This is Snipping tool. A native 64-bit executable. We see that after the word PE it has: **PE..d†**

While the Cruehead crackme, that is 32 bits, after the word PE, it has: **PE..L**

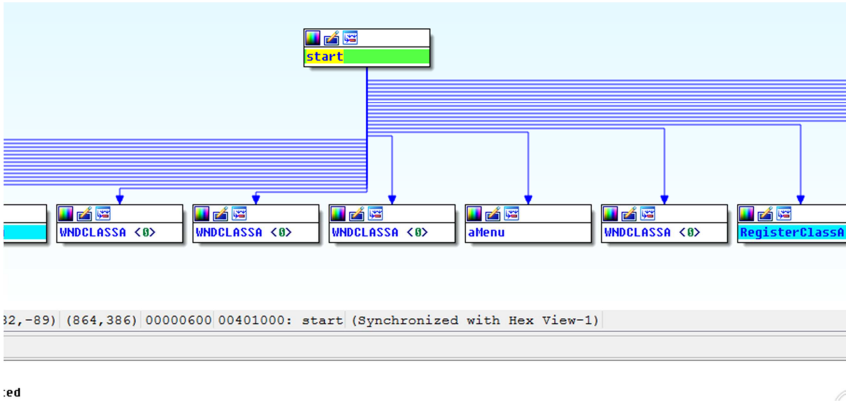
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00  MZP.........ÿÿ..
00000010   B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00  ¸.......@.......
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030   00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
00000040   BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90  º....´.Í!¸.LÍ!..
00000050   54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73  This program mus
00000060   74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57  t be run under W
00000070   69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00  in32..$7........
00000080   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100   50 45 00 00 4C 01 06 00 29 24 D9 0A 00 00 00 00  PE..L...)$Ù.....
00000110   00 00 00 00 E0 00 8E 81 0B 01 02 19 00 06 00 00  ....à.Ž.........
```

So that, we already know that we have to load it with 32-bit IDA using the direct Access I mentioned previously. When the IDA windows
IDA QUICK START shows up, we choose NEW to open a new file, we look for the crackme and select it.

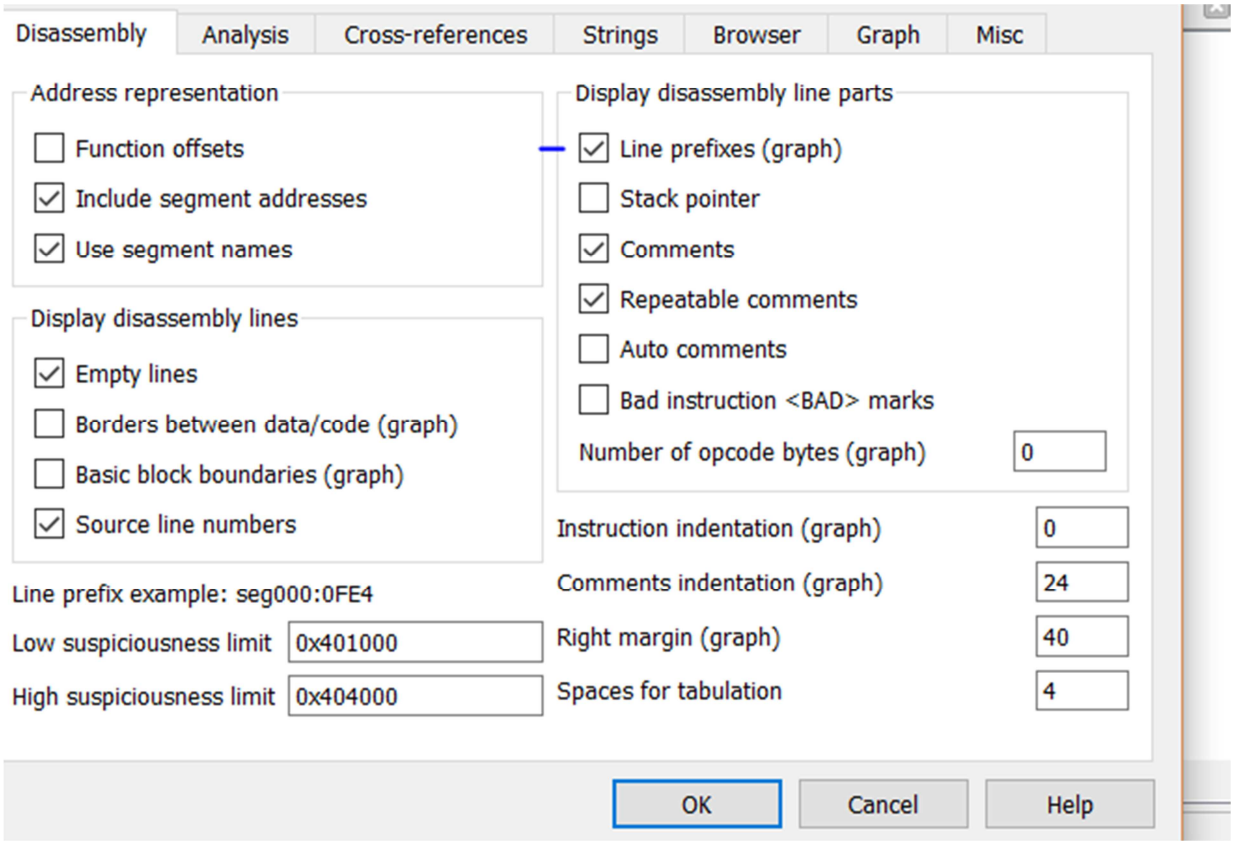


By now, we don’t touch the settings because it detects that it is an executable PE correctly and we click OK.

If we click YES on PROXIMITY VIEW, the program tree view will appear.




To change to graphic mode or a non-graphic instruction list we can do it alternating the space bar.

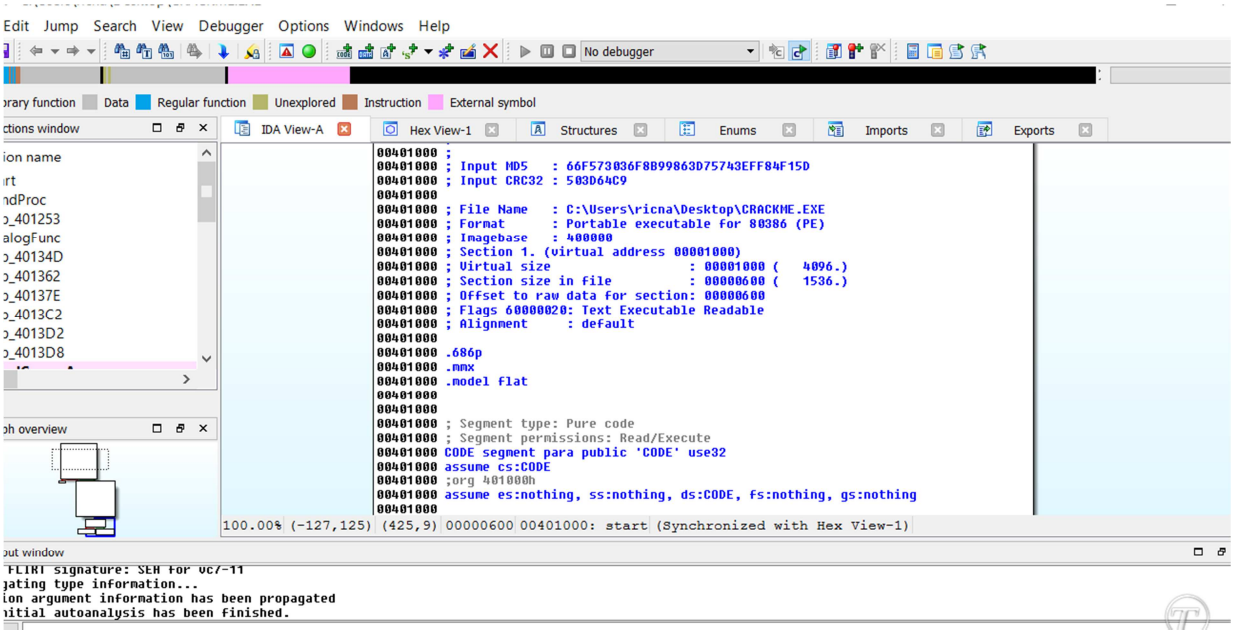



Also in OPTIONS - DEBUGGING OPTIONS -LINE PREFIXES we can add the addresses in front of the graphic view.

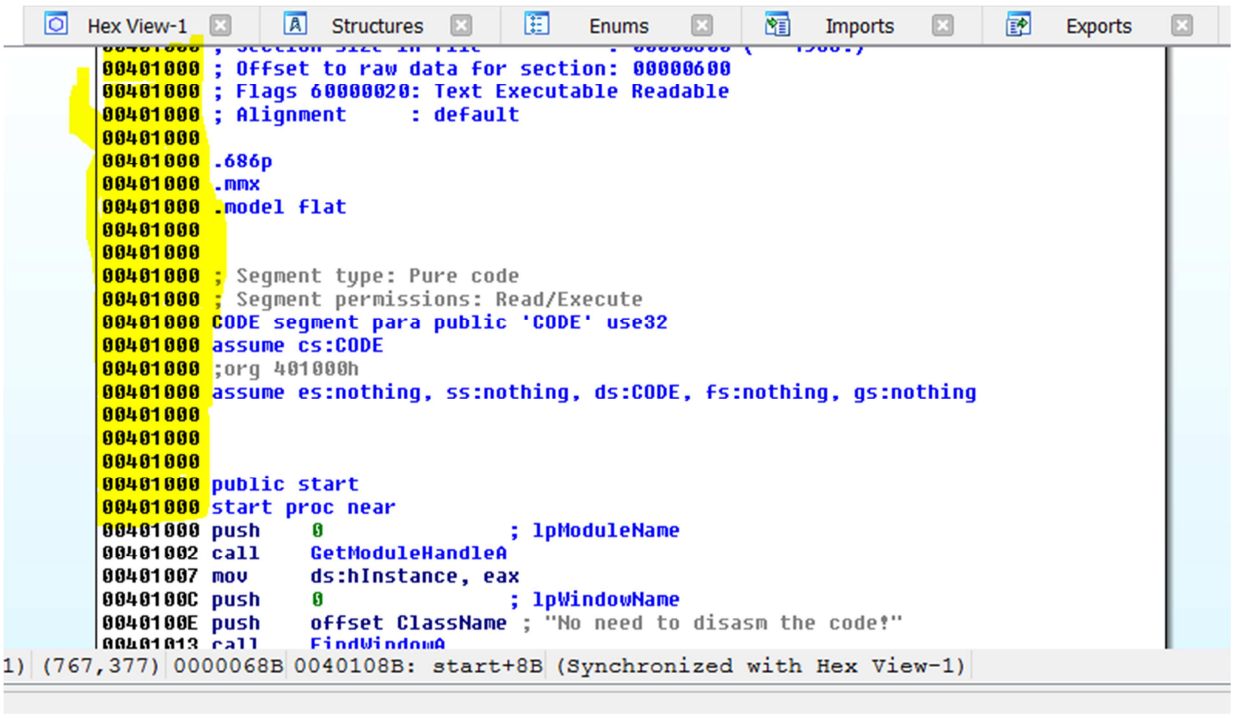
To load an executable, the first thing opened is the Disassembly View or Loader which doesn't execute the program. It just analizes it with reversing purposes creating an IDB or DataBase file.

To debug a program we must choose among all the debuggers possibilities included in IDA and run it in Debugger Mode which we will see later.

There are taps with many options in IDA. For example, if we click on the menu  VIEW-OPEN SUBVIEW, we can control the tabs we wish to show.



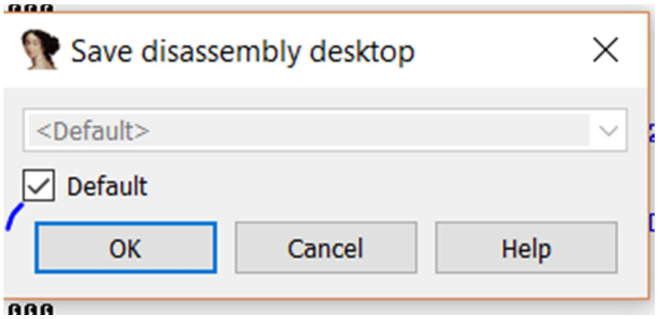

One of the confusions or discomfort using IDA, when we are not used to it, it is that there are parts of the graphic where there are some references to the same address, for example, at the beginning of a function the address is repeated some times.



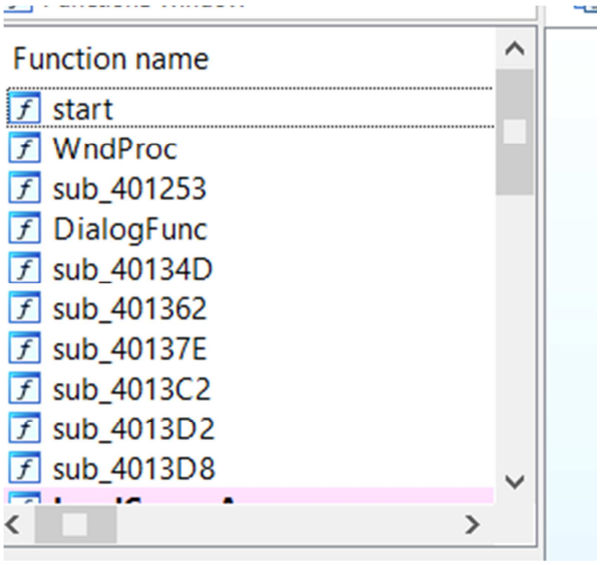

A trick is to take the last repetition as the beginning. There, we'll find the start of the disassembly list. In this case, the correspondent instruction of 401000 is the PUSH 0.

In IDA you can customize the Loader and Debugger interface separately.

Once we set, for example, the Loader windows and taps we use frequently, we go to WINDOWS-SAVE DESKTOP and select Default it will save the our settings. You can do the same settings when running the Debugger and using a different setting from the Loader.

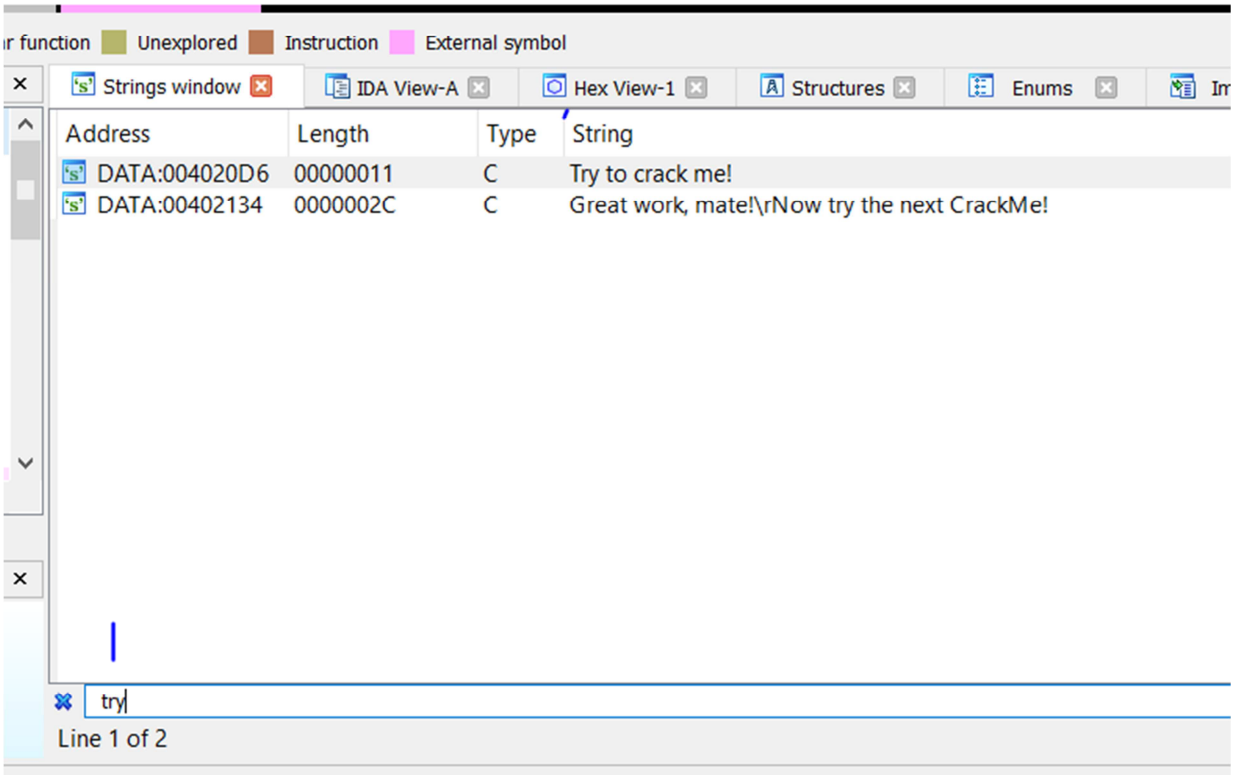



In any of the IDA taps where there are lists, for example:
FUNCTIONS, STRINGS, NAMES, etc.

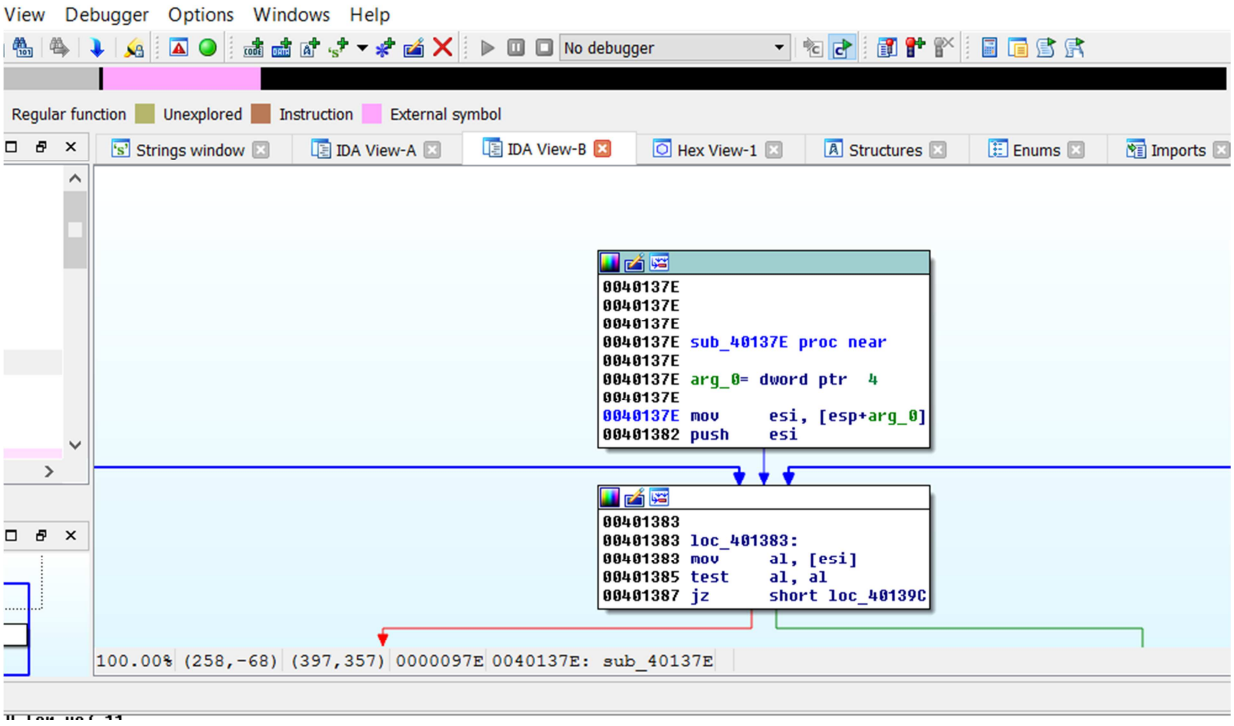



We can search with CTRL+F and a search window appears filtrating anything according to the characters we type.

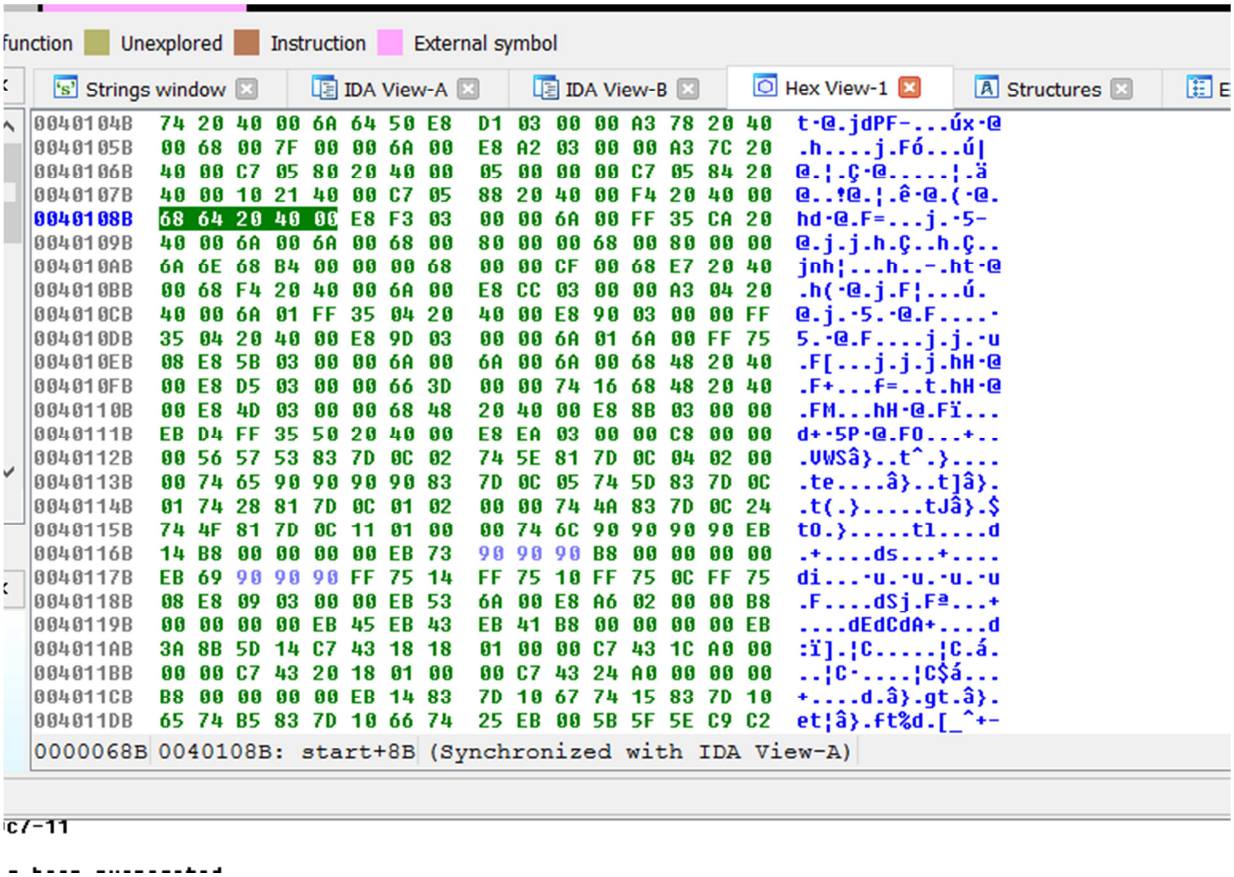
In VIEW- OPEN SUBVIEW-STRINGS we can also see all the strings containing the letters “try”.



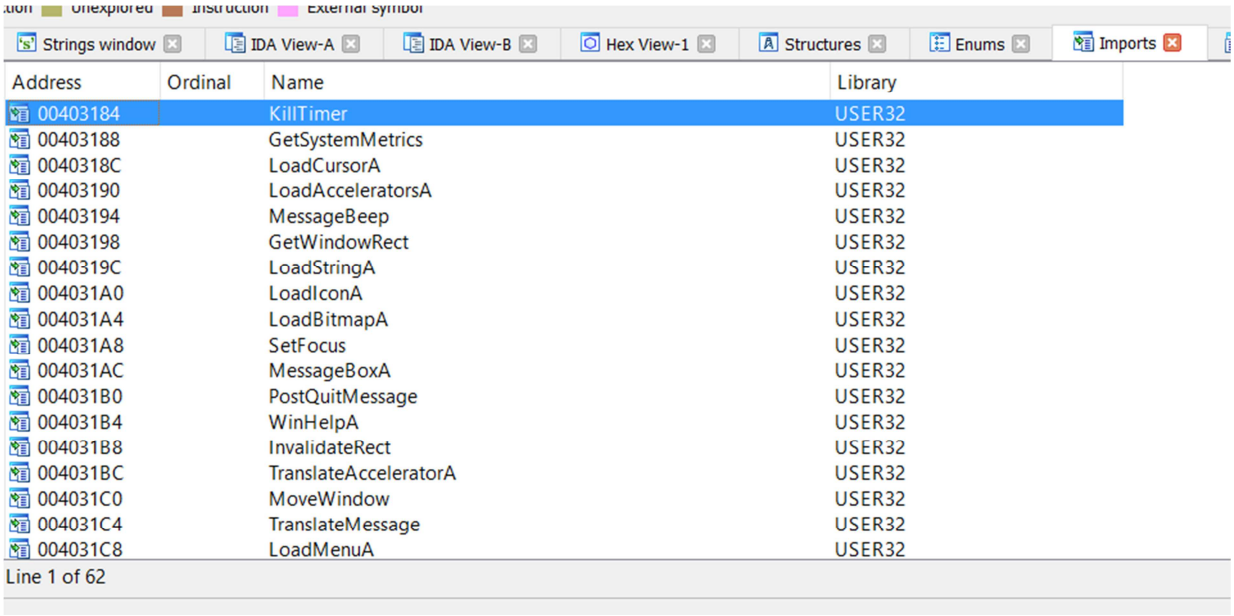

Also in VIEW-OPEN SUBVIEW-DISASSEMBLY, I can open a second disassembly window showing a different function from the first one.

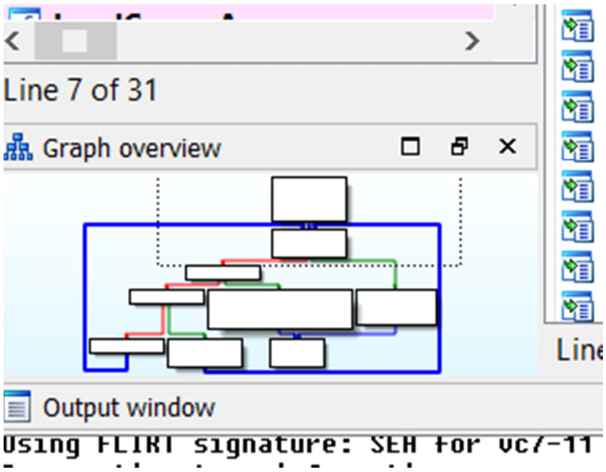



In OPEN SUBVIEW in the LOADER there is a hex view or HEX DUMP tab.




In OPEN SUBVIEW I can show the imported functions or IMPORTS tab.

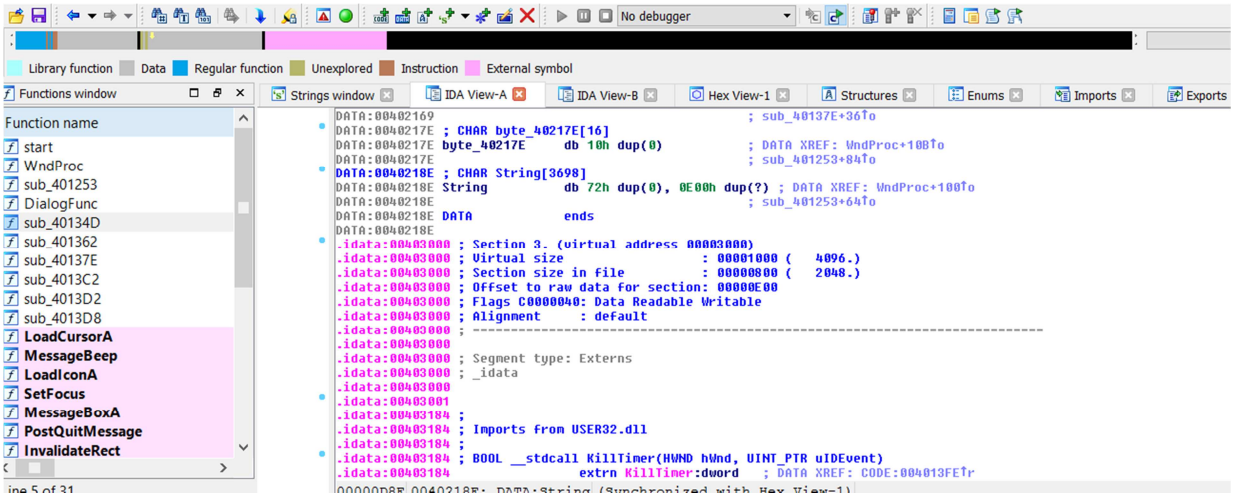



In VIEW, you can activate GRAPH OVERVIEW that is a browser for the visible function graphic to change or move the recent function on screen.

There are also tabs dedicated to STRUCTURES, EXPORTS, NAMES, SEGMENTS, etc. Which I'll explain when we start using them.

The top browser shows the different parts of an executable using a variety of colors.




Below, it says the meaning of any color. For example: gray is the **.data section** and if I click on the bar in that gray part, the graphic moves to that section whose addresses are gray.

The pink part belongs to **External Symbol** or **idata section** and the blue part are the functions detected in the **code section**.

This was a quick review. In the next tutorials we will increase step by step.

**Until part 2**
**Ricardo Narvaja**

**Translated by: @IvinsonCLS**