PART 8

We saw a review of the main instructions. Let's continue with the LOADER now.

When we don't have much experience with static reversing, we need to comment and rename some things to have a better orientation and we do most of the work debugging.

Practicing static reversing hard helps you analyze programs without or almost without the use of the debugger.

Normally, there are heavy programs that you don't want to reverse completely, but one or some functions in some specific place.

IDA gives us the possibility to interact getting the best reversing results as no tool can do it. But it will depend on the user's experience.

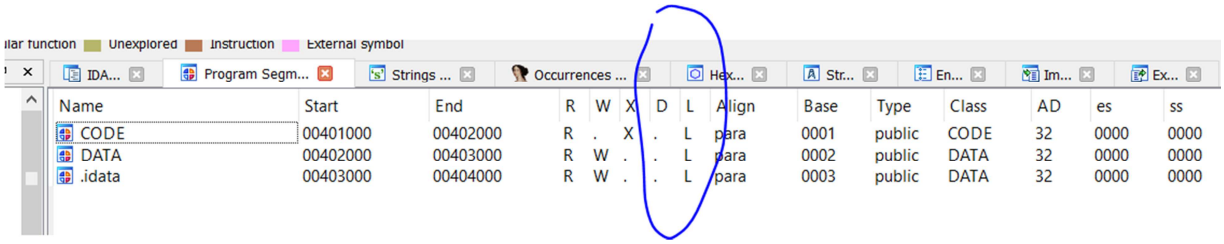
The famous phrase, "The arrow is not guilty, but the archer."
Here, that phrase applies. It is necessary to practice and improve the use of IDA. If we fail or we don't have a good result, we need to practice harder. It's not an easy tool and it has a lot of possibilities which let us learn new things everyday.

Let's analyze CRUEHEAD's CRACKME statically. It doesn't matter if we solve it or if it is necessary to debug it at the end when studying the debugging lesson. Now, we need to get used to the LOADER.

Click on VIEW-OPEN SUBVIEW-SEGMENTS.

There, we see the segments loaded automatically.

The HEADER, that would be in 0x400000 before the sections, is not loaded because when it is loaded for the first time and we click OK on everything, in options, it loads just some executable sections automatically without the header.

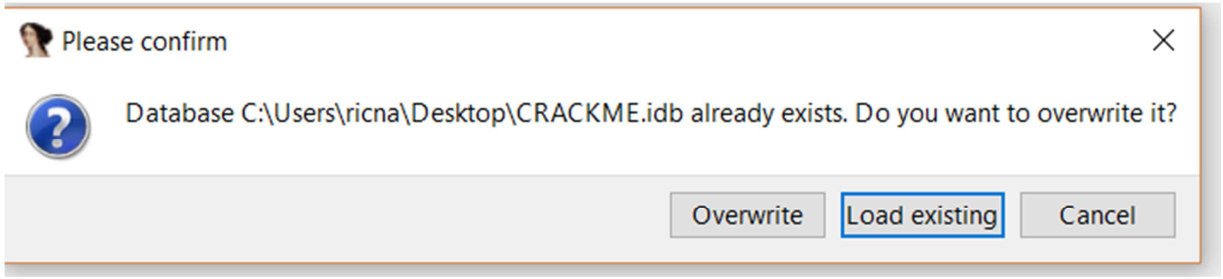


After the section **NAME**, we have the **START** and **END** addresses, then the RWX columns that tell me if it initially has READ(R), WRITE(W) or EXECUTION(X) permission.

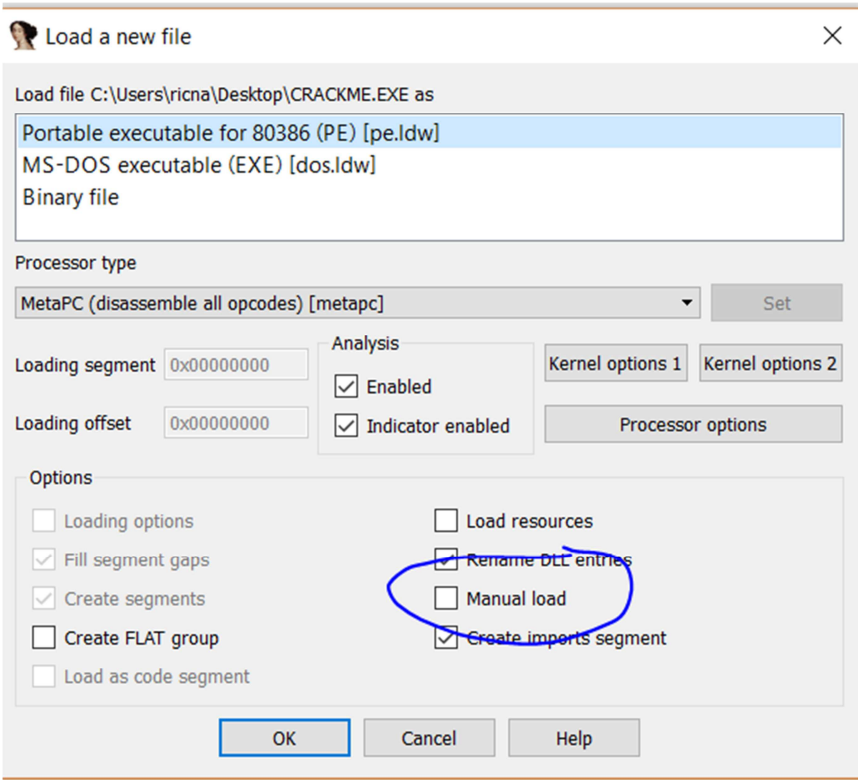
There are two D and L columns that belong to DEBUGGER and LOADER. The first one is empty because it is called when we load the program in DEBUGGER mode and shows the loaded segments. L shows what the LOADER loads and other not so important columns.

Here, it is not relevant to load the header. We saw in the previous lesson that when renaming an instruction and assembling a jump to 0x400000 it marked it in red as a non-loaded section.

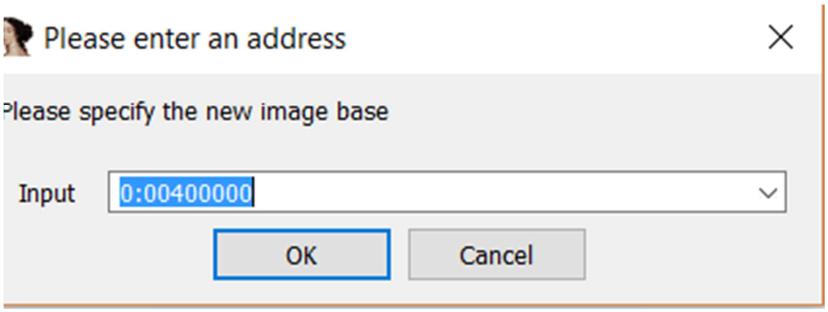
If I want to load the HEADER, I load the CRACKME.exe again.



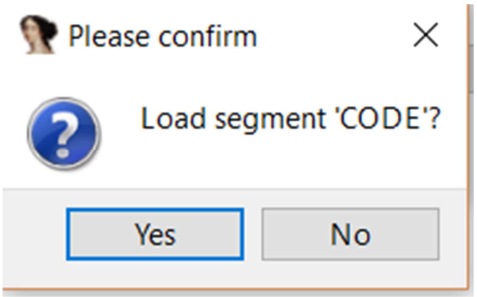
I select OVERWRITE to create a new analysis.



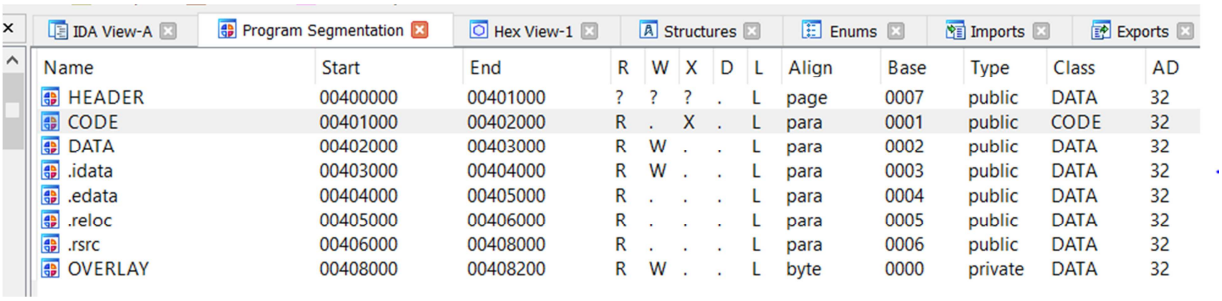
I check MANUAL LOAD and click OK.



I click OK to change the BASE where it will be loaded.



Click OK on each messagebox to load everything.

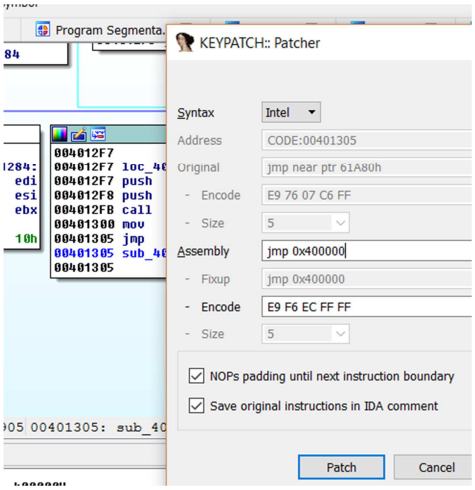


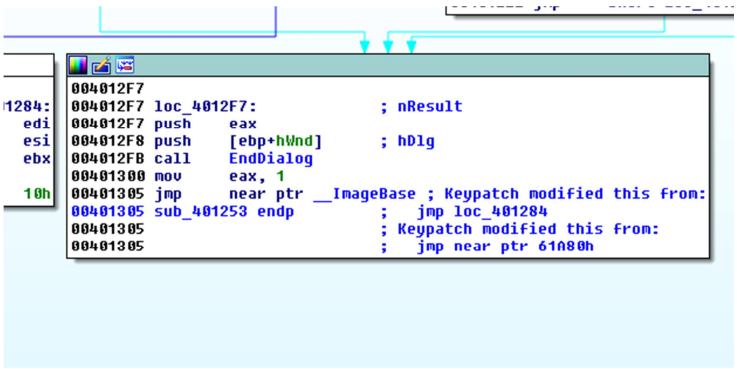
Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD
HEADER	00400000	00401000	?	?	?	.	L	page	0007	public	DATA	32
CODE	00401000	00402000	R	.	X	.	L	para	0001	public	CODE	32
DATA	00402000	00403000	R	W	.	.	L	para	0002	public	DATA	32
.idata	00403000	00404000	R	W	.	.	L	para	0003	public	DATA	32
.edata	00404000	00405000	R	.	.	.	L	para	0004	public	DATA	32
.reloc	00405000	00406000	R	.	.	.	L	para	0005	public	DATA	32
.rsrc	00406000	00408000	R	.	.	.	L	para	0006	public	DATA	32
OVERLAY	00408000	00408200	R	W	.	.	L	byte	0000	private	DATA	32

And there, we see the all the sections including the HEADER and more DATA.

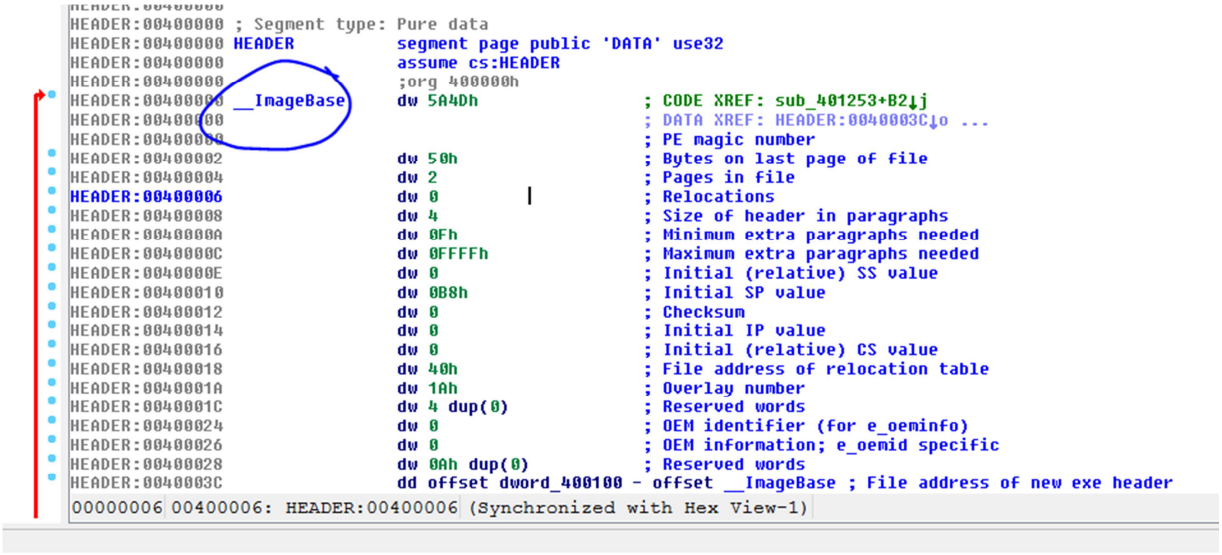
This is normally unnecessary, but it is good to have it into account.

Now, I will save a SNAPSHOT clicking on FILE-TAKE DATABASE SNAPSHOT and I will assemble a jump to JMP 0x400000 as before.



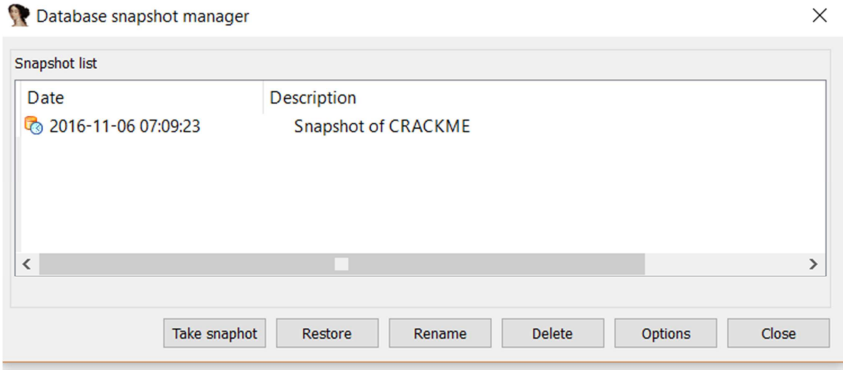


There, we see that the address 0x400000 is not marked in red and it says that the ImageBase is there. We can also go and see it by clicking on the name ImageBase.

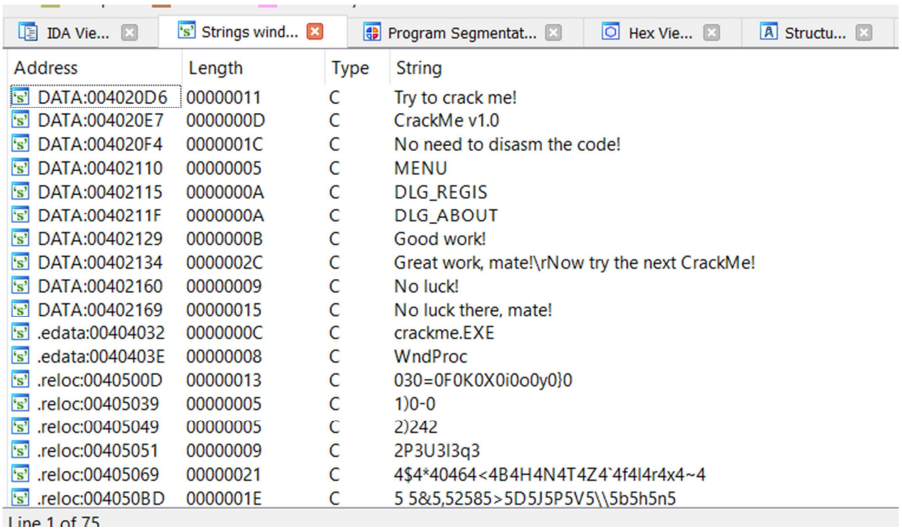


The header is there in 0x400000 with its ImageBase tag and its content is detected as header and shown with its fields.

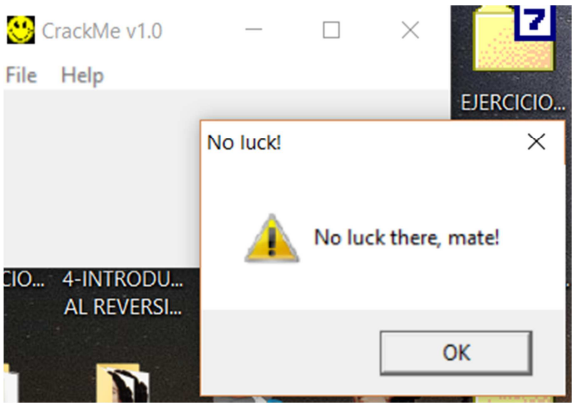
Let's go back to the previous snapshot we did before editing the exe in VIEW-DATABASE SNAPSHOT MANAGER and click on RESTORE.



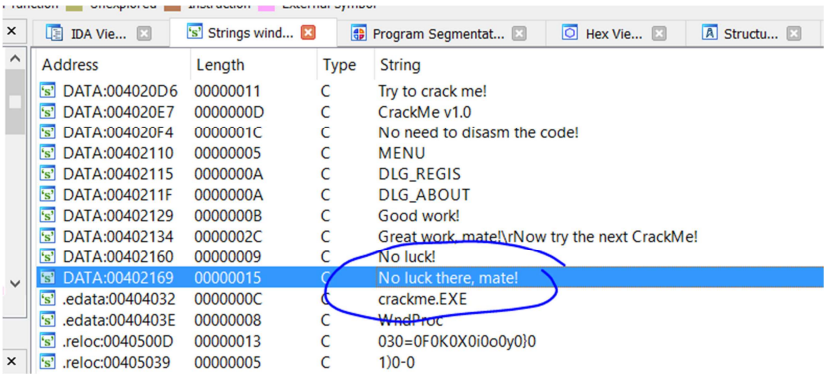
In all cracking or reversing exercises, strings are very important to find interesting places. If they are present, they can help us. Click on VIEW-OPEN SUBVIEW-STRING.



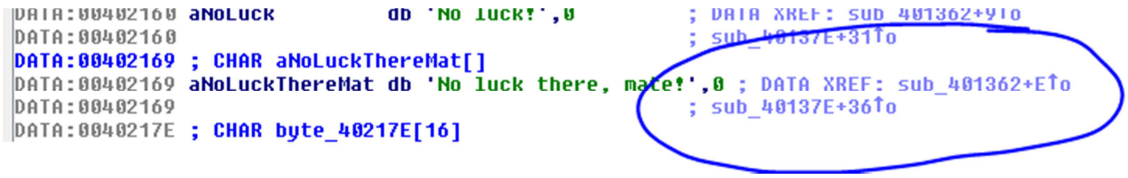
As there are more sections, we have more detected strings. Anyways, if we run the crackme.exe without IDA, we see, in the HELP menu, the REGISTER option and it asks for a user and password. If we enter anything, it shows NO LUCK THERE, MATE!



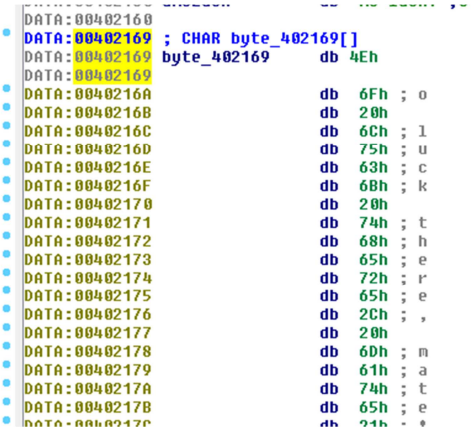
Let’s look for that string in IDA list.



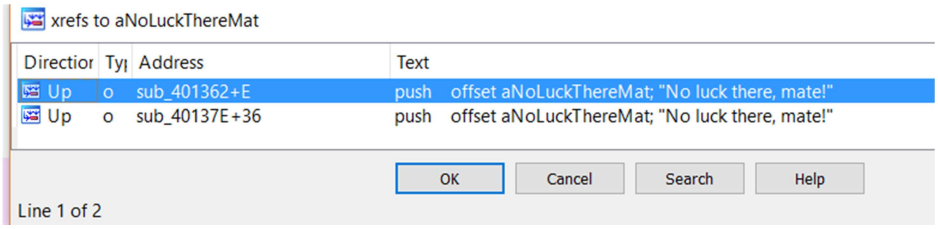
Double click on that string.



In 0x402169, we have that string. If we press D on that address, we will see the separated string bytes.

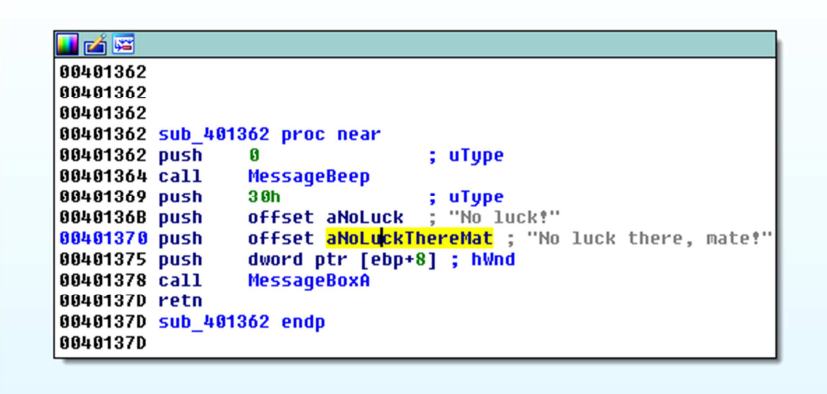


If we press A on that address, we restore it. Let's press X to see the two references on the right side comfortably.

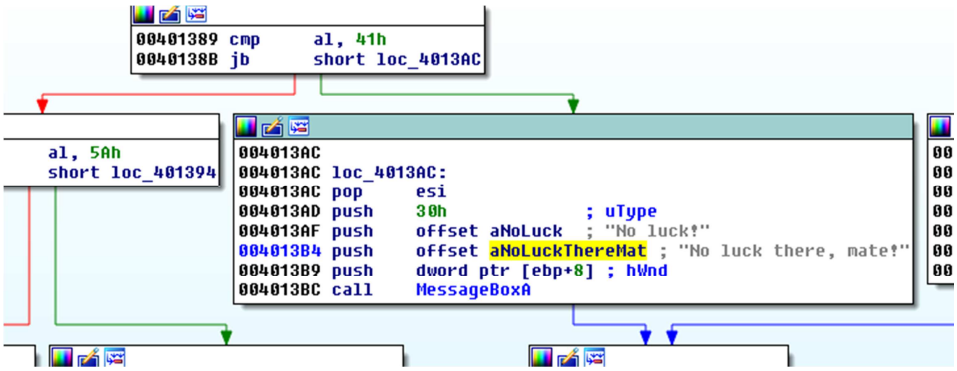


That string is called from two different functions. One from **sub_401362** and the other from **sub_40137E** and both are above the address where we are now. That's why in the **DIRECTION** column it shows **UP** in both

We know they are two different functions because IDA shows the reference addresses as functions + XXXX. If they belonged to the same function, it should change the XXXX, but keeping the first part. Here, both **sub_** are different.

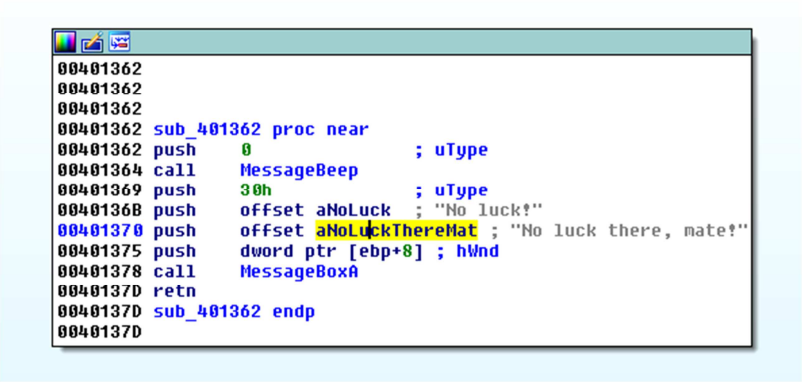


This is the first reference and the second one is below.



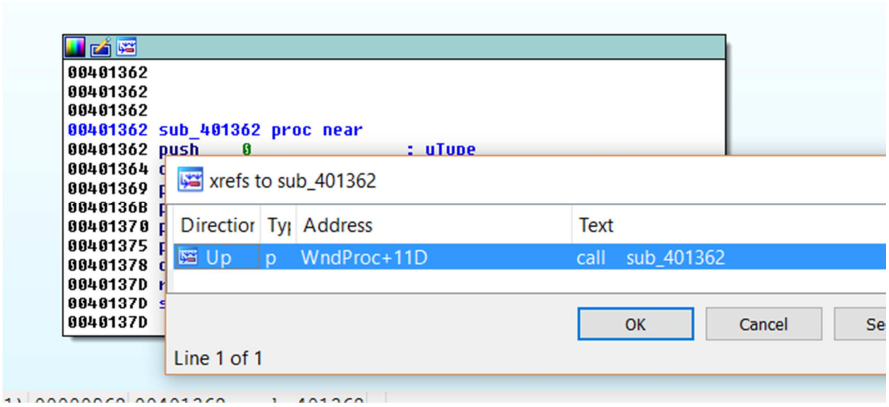
We have places where it shows the bad message. As we are going to try to advance without debugging, if you want to see this part in Olly, place a breakpoint in both addresses and see if it stops when entering an invalid password. This help is useful, but the idea is doing everything in IDA. We can place breakpoint (BP) in both blocks with IDA and see if it stops using the IDA DEBUGGER, but we will do that more ahead.

Let's take the first reference.

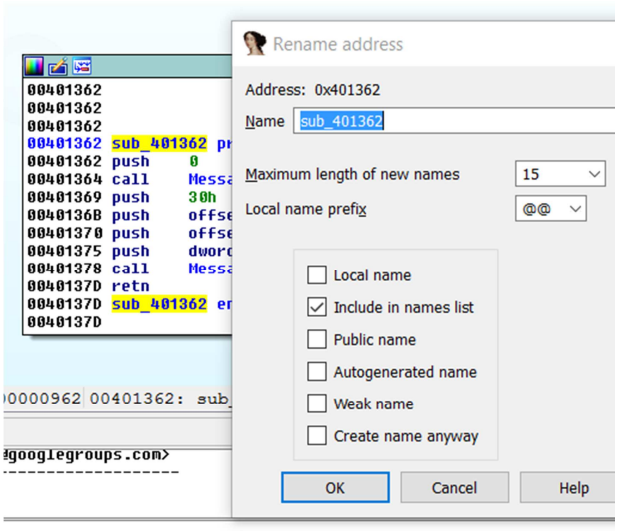


There is a CALL to the MessageBox API which shows those messages like NO LUCK THERE MATE and it receives the NO LUCK string as an argument for the window title and NO LUCK THERE MATE for the text.

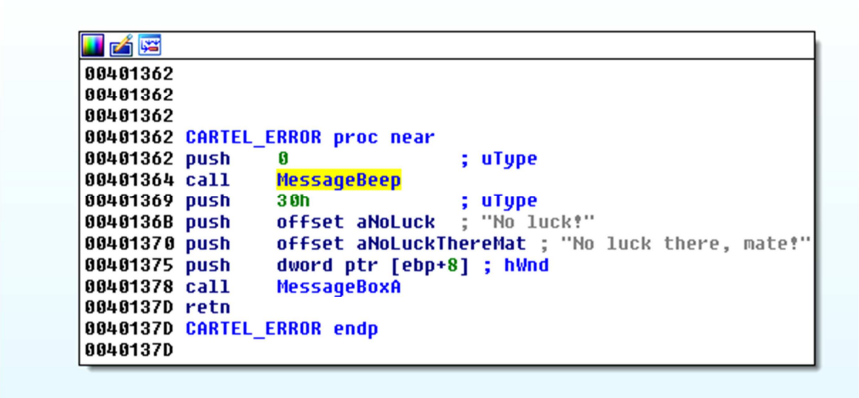
The other reference is exactly the same. It means that the bad boy message can be triggered from two different places, maybe evaluating different things and to show the good boy message we will have to avoid both of them.



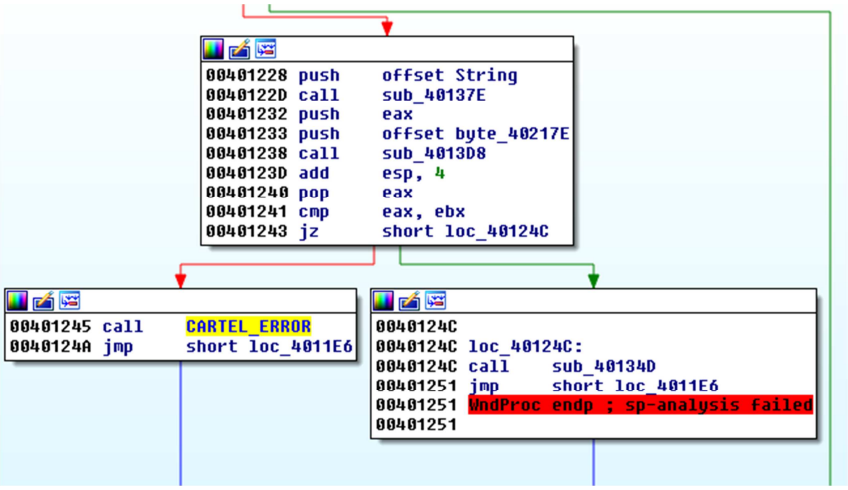
If we see the 0x401362 references with X, there is just a place. Before going there, let's rename the 0x401362 function with a tag that tells us what it does, for example, CARTEL_ERROR.



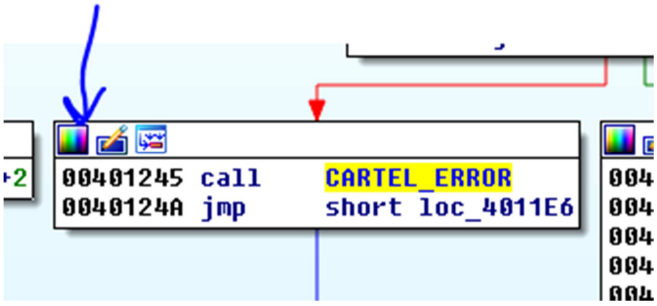
Let's press N on the address and write our new name.



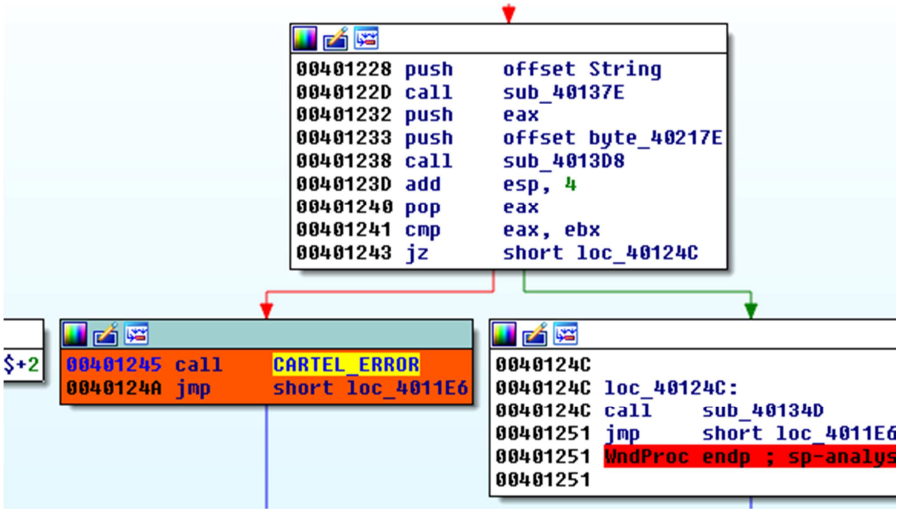
Now, go to the reference.



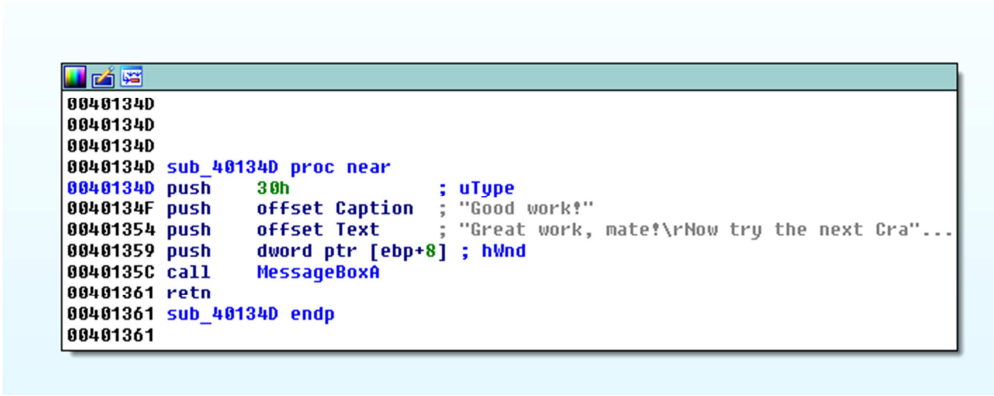
There, we see the block that takes us to CARTEL_ERROR and there is a decision before. To see everything better, I color the good and bad blocks.



There, we have the icon to change the color. You may think this is not relevant, but in complex functions, it will help a lot.

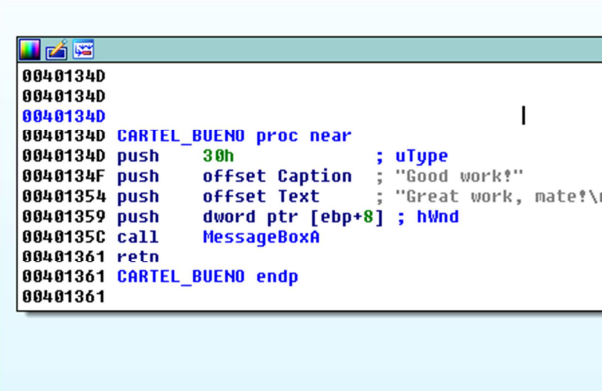


There is a conditional jump that we won't analyze now because it jumps from one place to other, but let's press ENTER on 0x40124C to enter the CALL 0x40124D.

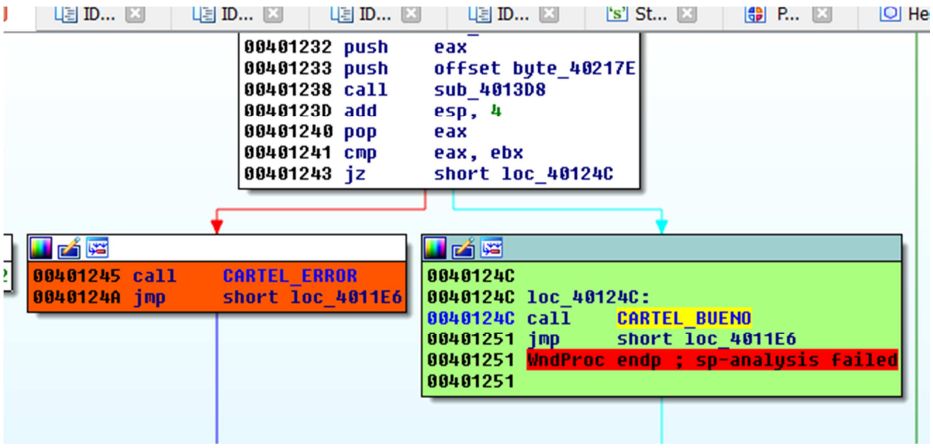


There are possible good boy strings that were also in the string list, but we thought that it was better to start with the bad boy message because this could be a false string, but as we see the program decides to come here or to the bad boy zone this is possible the correct one

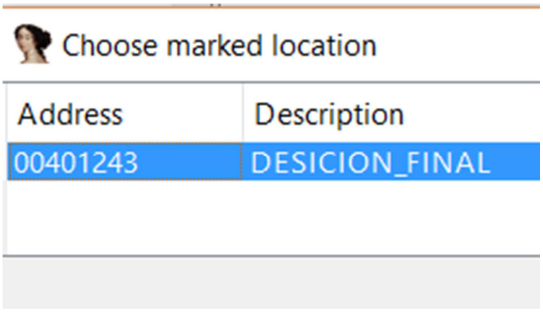
Let's rename this as CARTEL_BUENO or GOOD_BOY in English.



Let's paint this block in green.

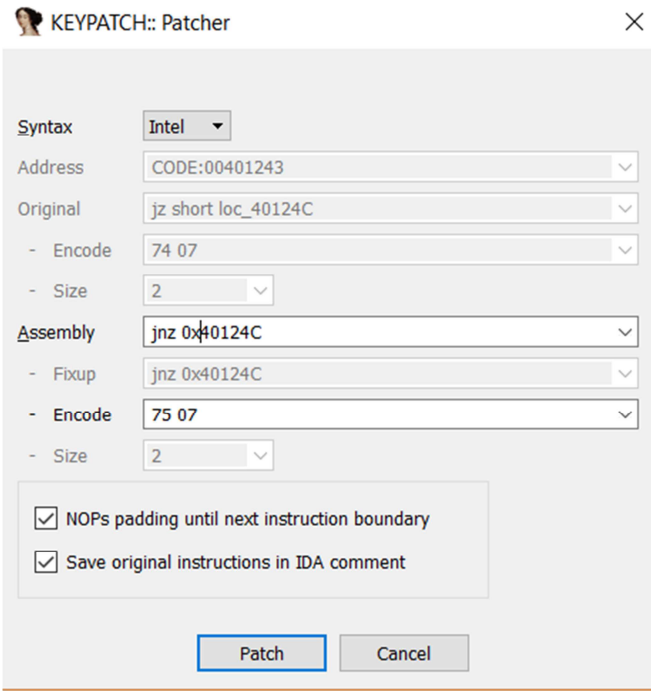


As we are also going to work with other parts of the program, select the **JZ** of **0x401243**, click on **JUMP-MARK POSITION** and name it as **DECISION_FINAL** to come back here easily.

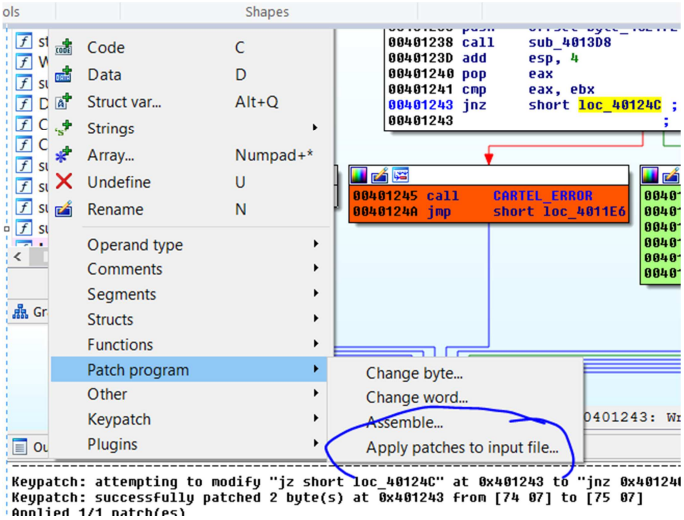


In **JUMP-JUMP TO MARKED POSITION** there is a list with all our marks to go to any place quickly.

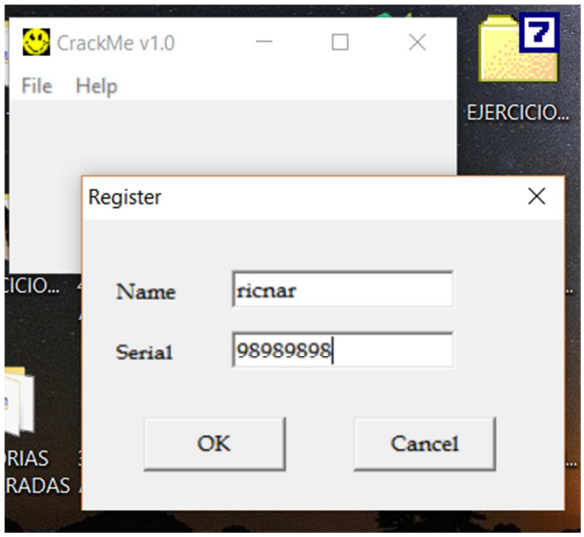
It means that if we patched that JZ changing it by JNZ it would take us to GOOD WORK using an invalid password if the previous bad boy message doesn't appear. Let's see.

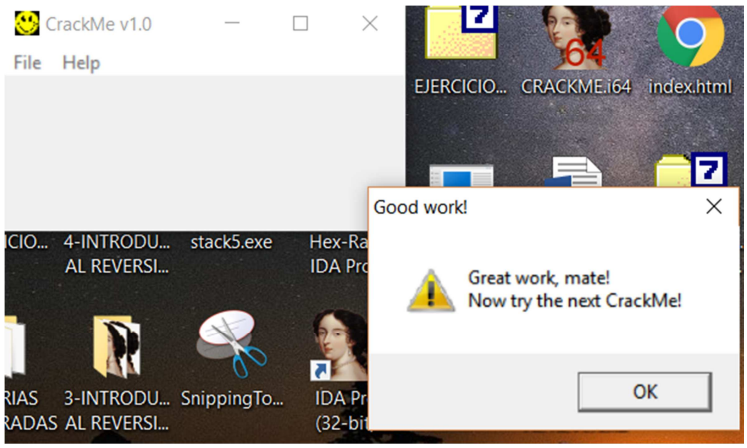


I edit it with KEYPATCH.

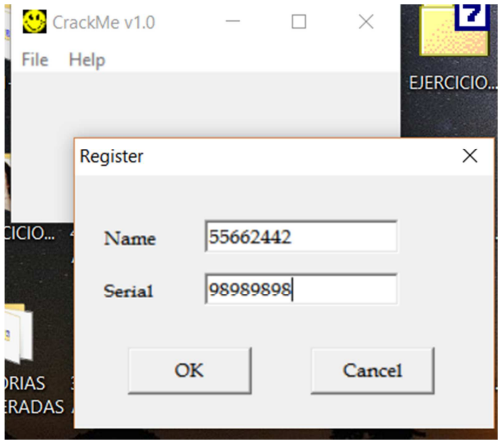


And save it with APPLY PATCH TO INPUT FILE.

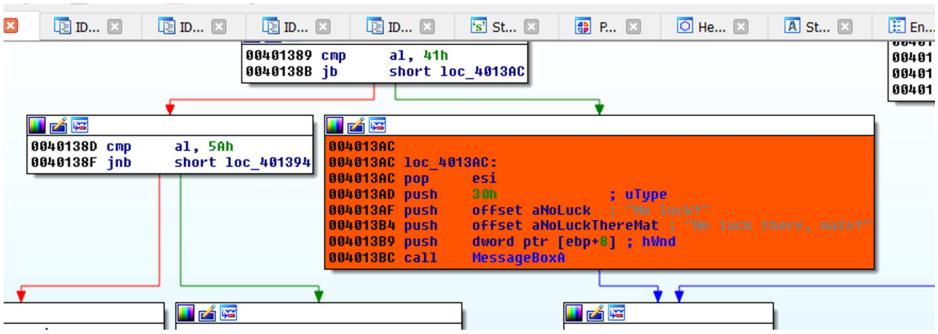




We patched it, but we need to patch the other error message.

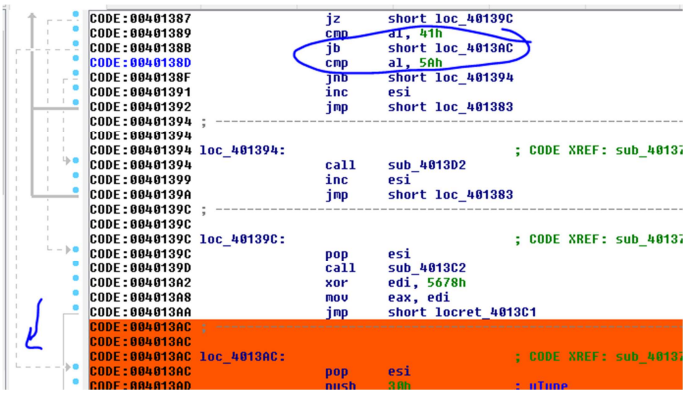


If we enter that, we see that it shows the good boy message and then the bad boy. Let's try to patch the other error message.



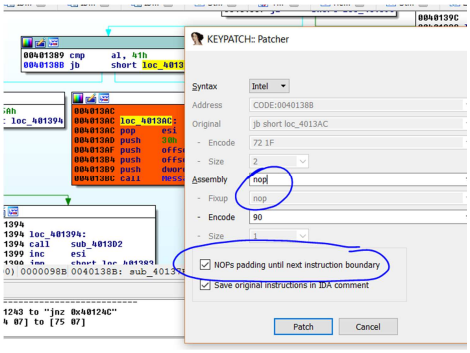
The other bad message is there. I painted it in red. We will analyze this crackme more ahead completely, but it is obvious that it compares with 41 that is A in ASCII and if it is below (JB) it shows the error. In my last try, I entered numbers in my name; obviously, numbers are below 41 (0 = 30, 1=31, etc.) So, when it detects my name has numbers, it shows the error msg. Let's patch this. Here, we cannot change JB by JNB because it would take us out when entering a name with letters.

If we quit the graphical mode with the space bar…



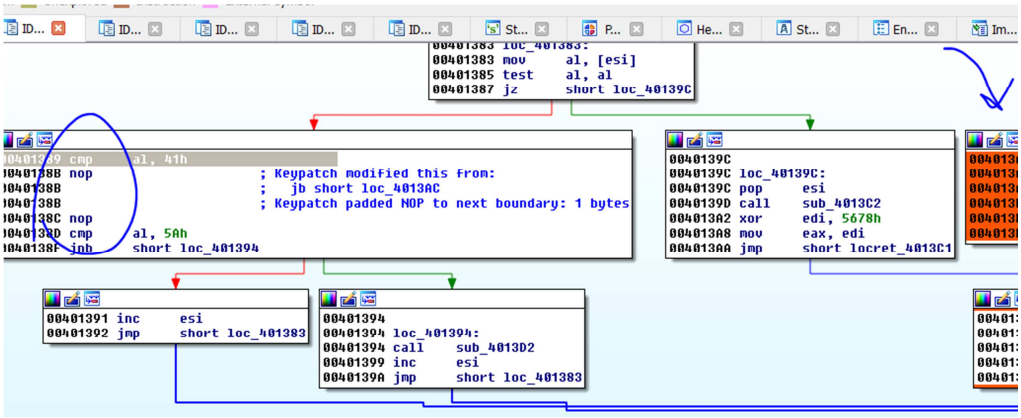
…we see that it will jump to the error msg. The pointed line on the left shows the jump. So, if I NOP it, it won't jump and it will continue to the next instruction without coming here.

Going back to the graphical mode.



It will fill that with 90 until the next instruction, so that it won't break down, I think. ☺

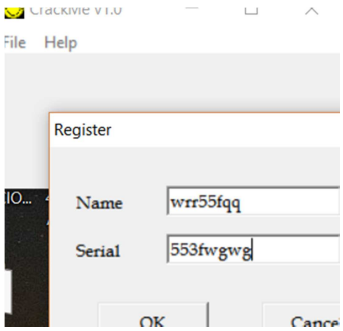
It is OK, but the blocks are messy and the view is ugly. So, right click-LAYOUT GRAPH and it will look better.

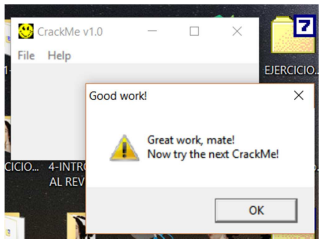


We see the NOPs there and the error block is isolated. The program cannot go there anymore.

I save the changes as before.

Let's see if it accepts anything.





This is just the beginning. We just patched it to get in touch with static reversing. More ahead, we will reverse it completely and make a keygen, but that will be in others chapters, by now, we go slowly.

Ricardo Narvaja

Translated by: @IvinsonCLS