REVERSING WITH IDA PRO FROM SCRATCH
PART 22

DIFFERS

A differ is a program that from two consecutive versions of the same program, tries to match the functions despite the changes and tries to show which functions were changed and where.

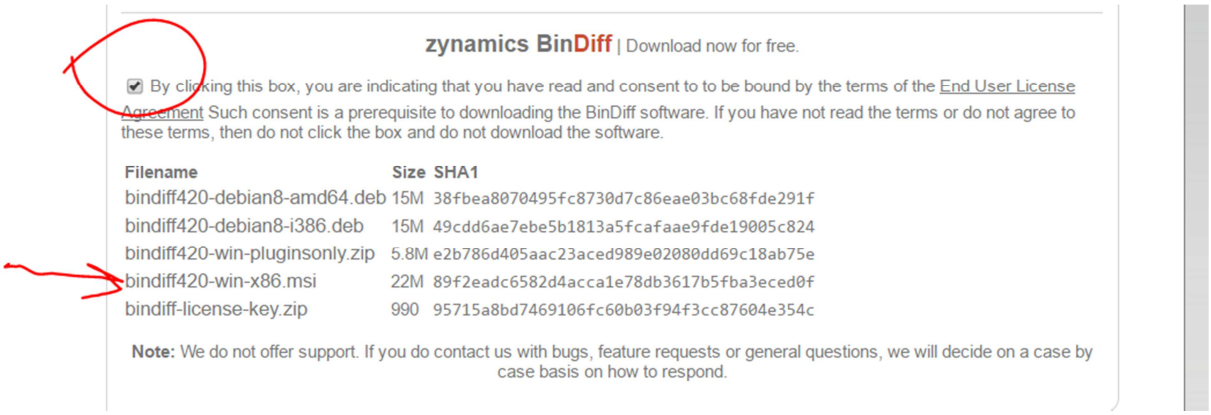
Obviously this work is not simple, especially when from one version to another there have been many changes which go from the application of some security patch to solve some vulnerability, as well as there may be improvements in the program, added in the interface, general changes , etc.

Since we have to work with differs we know that the bigger and more changes in the executable, the more ungrateful is the work since the Differ makes some mistakes when matching.

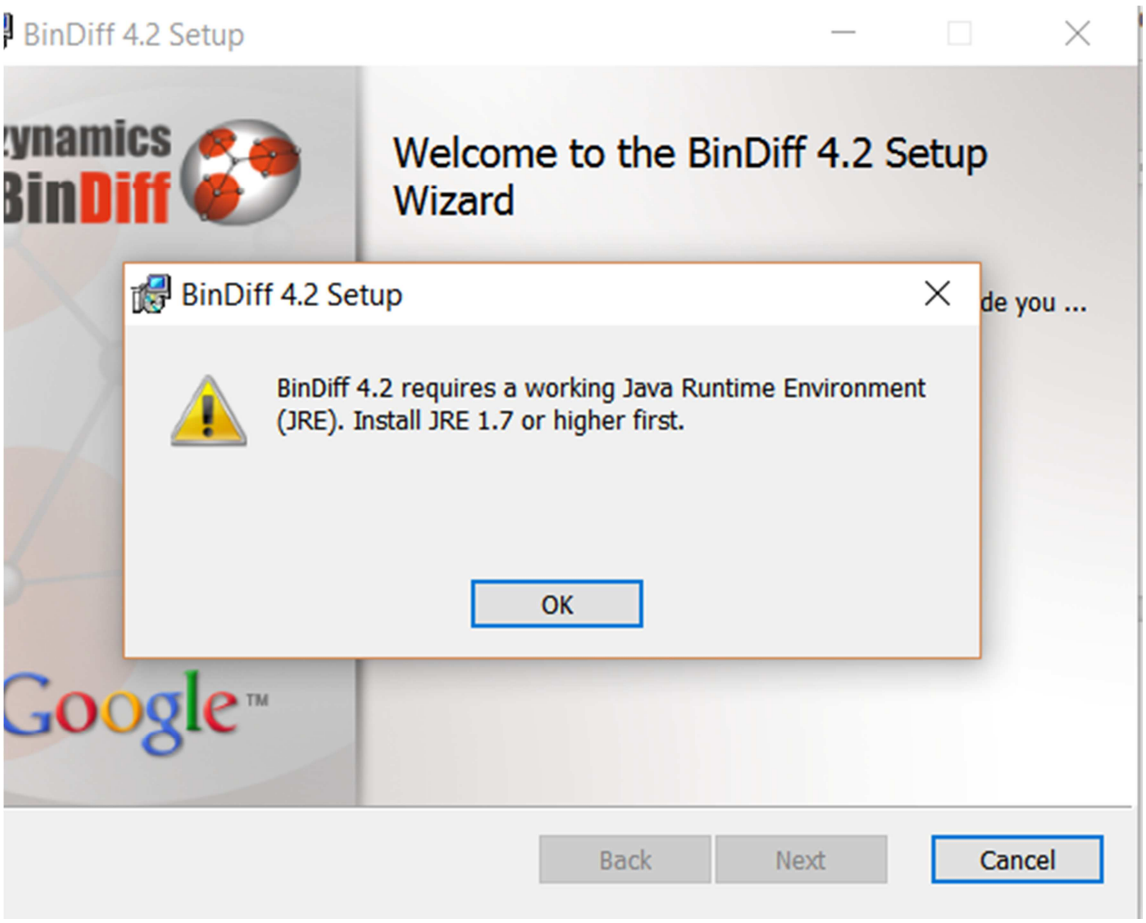
Let's see the three best known differs we will use in general, to install and know them, each has its strength and weakness. Sometimes more than one has to be used in complex cases to try to clear up.

The first one we are going to install is the BINDIFF.

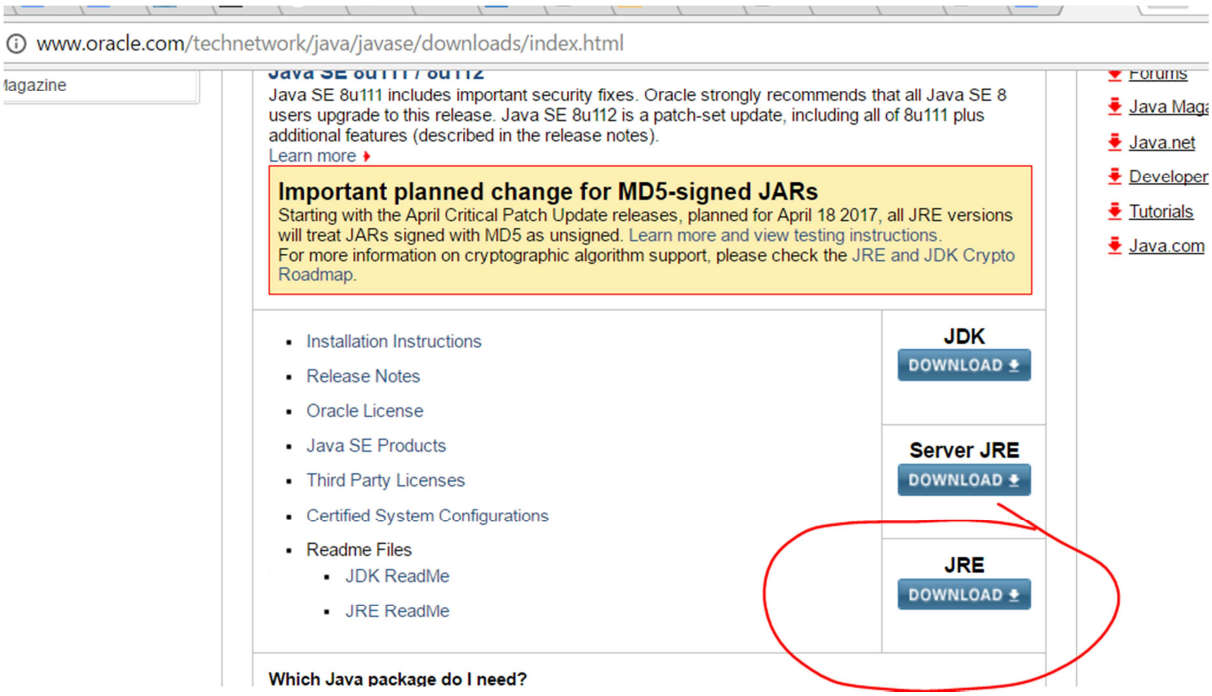
https://www.zynamics.com/software.html

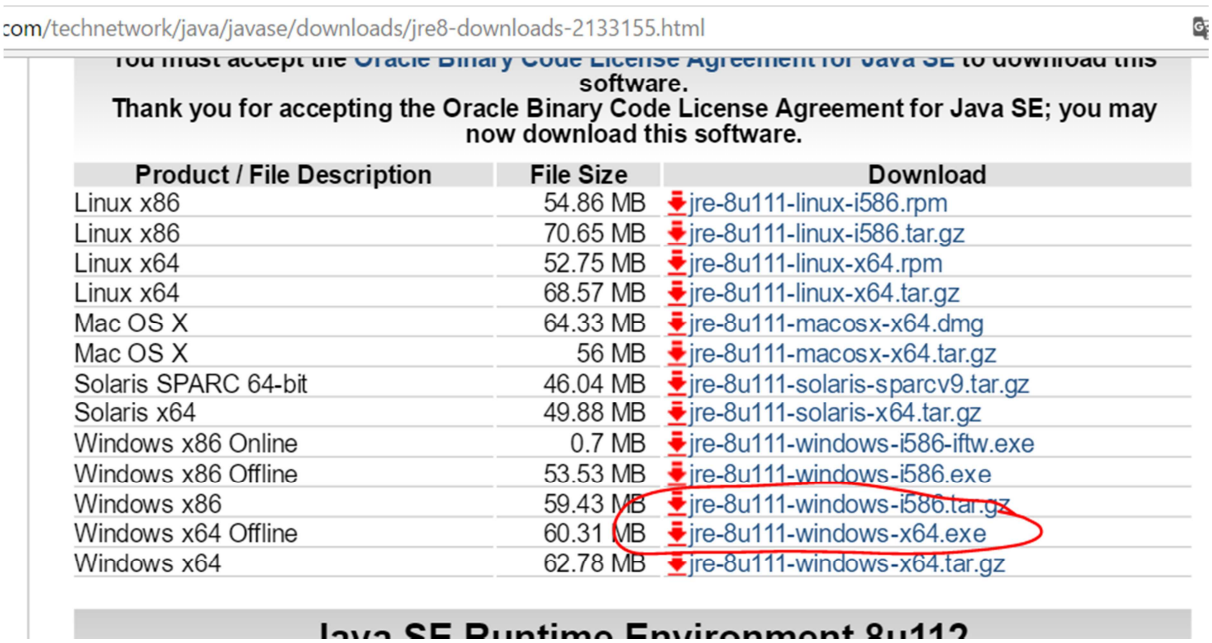


From there it can be downloaded accepting the conditions.



Well, download java if we do not have it.

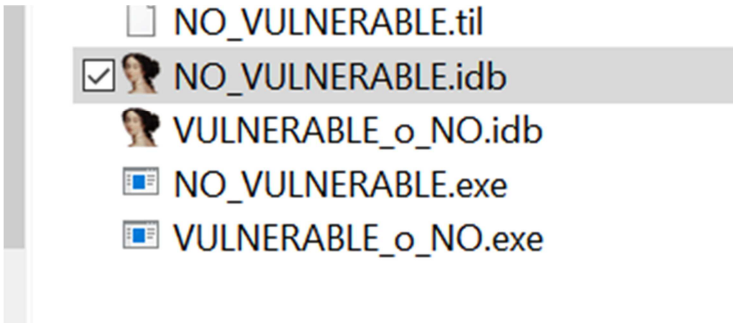




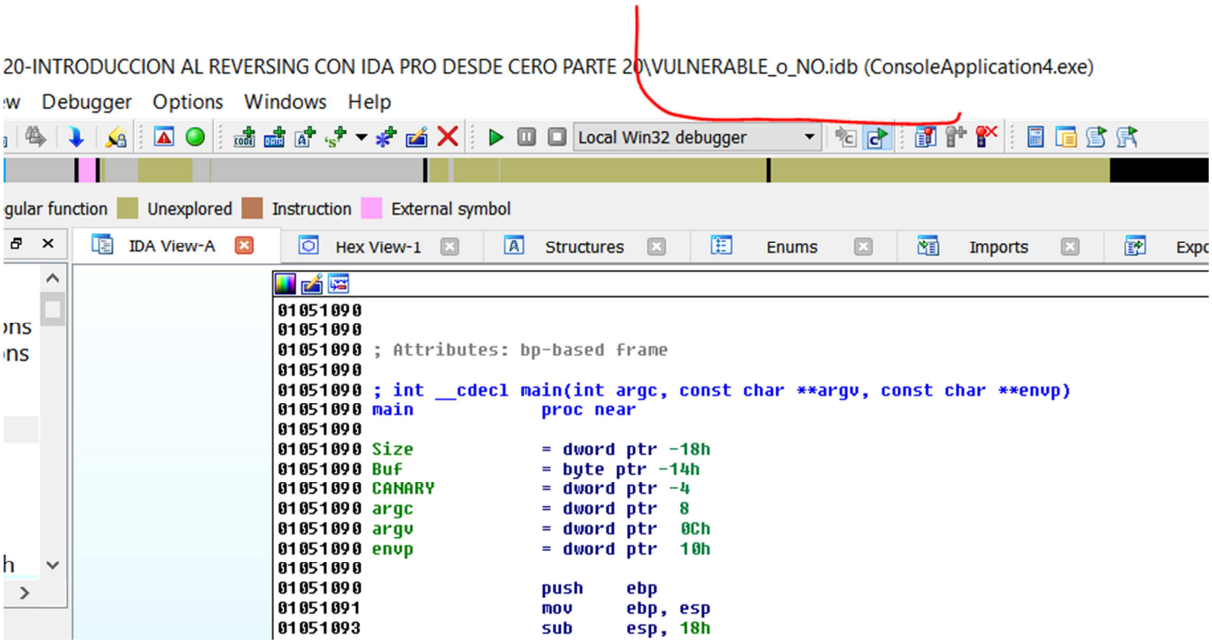
It seems to be the version downloaded and installed.

I then retry installing the bindiff and it looks like no problem, no complains.

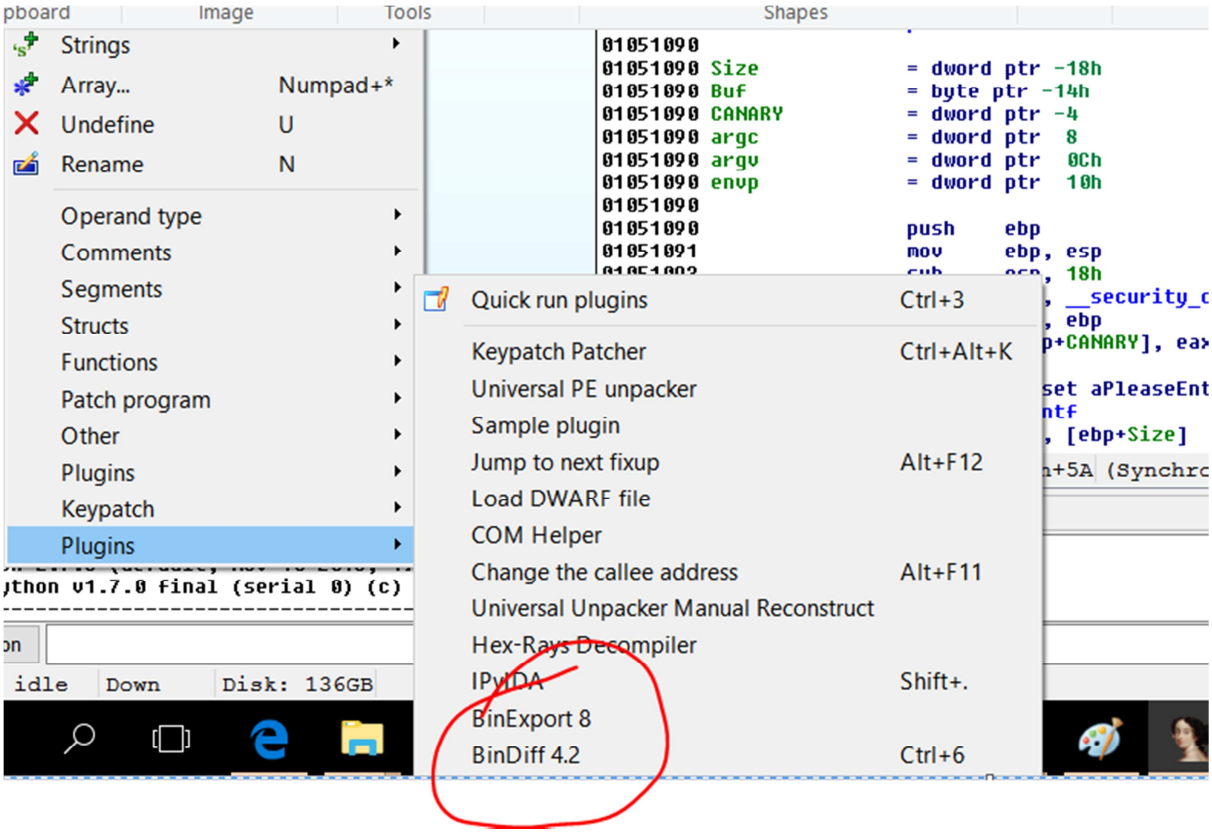
I still have the previous exercise executables vulnerable and not vulnerable, I open the new one the non-vulnerable or patched in the IDA LOADER for it to create the IDB if I did not before.



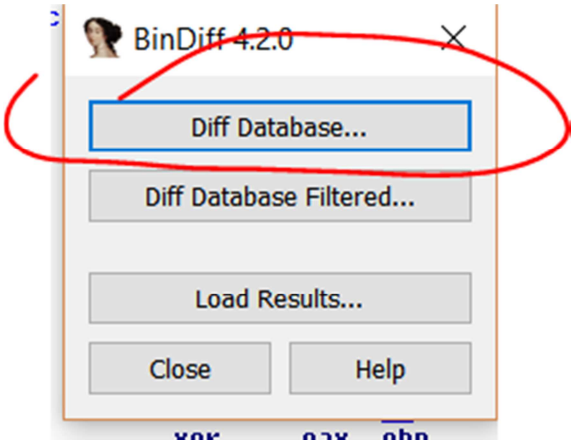
Then I open the vulnerable.



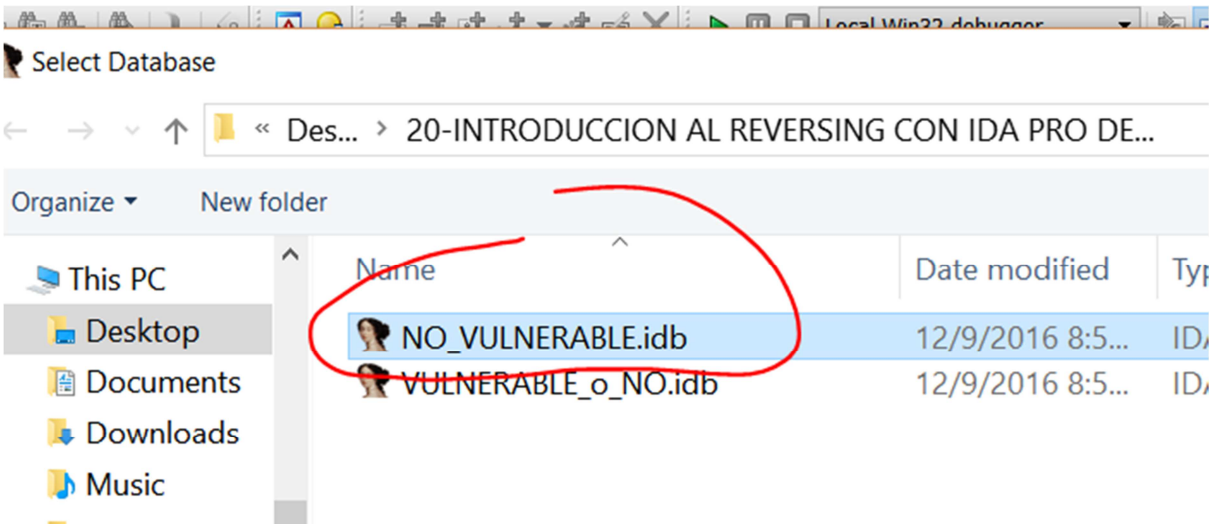
Then, I go to edit-plugins.

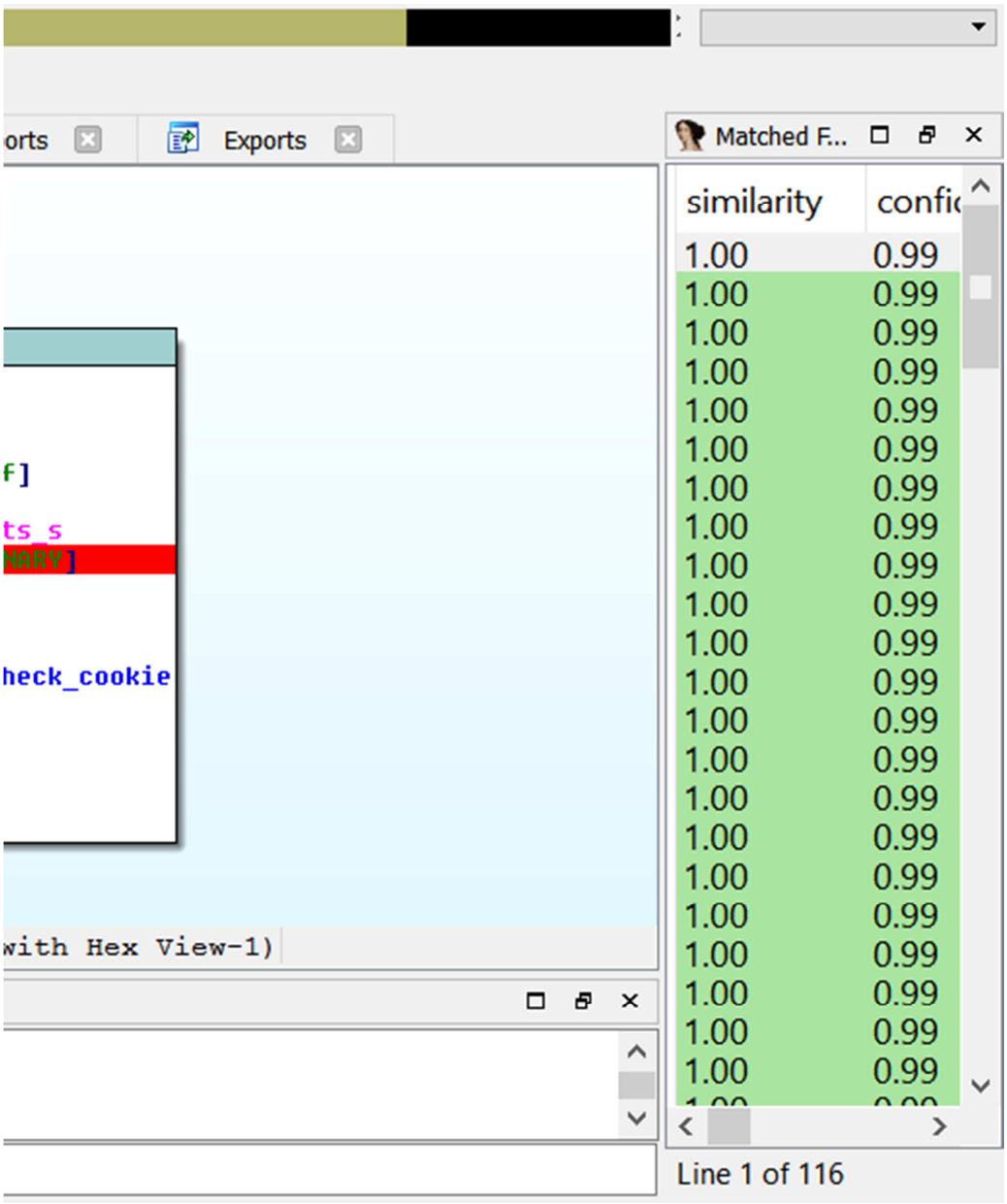


And I choose: BINDIFF

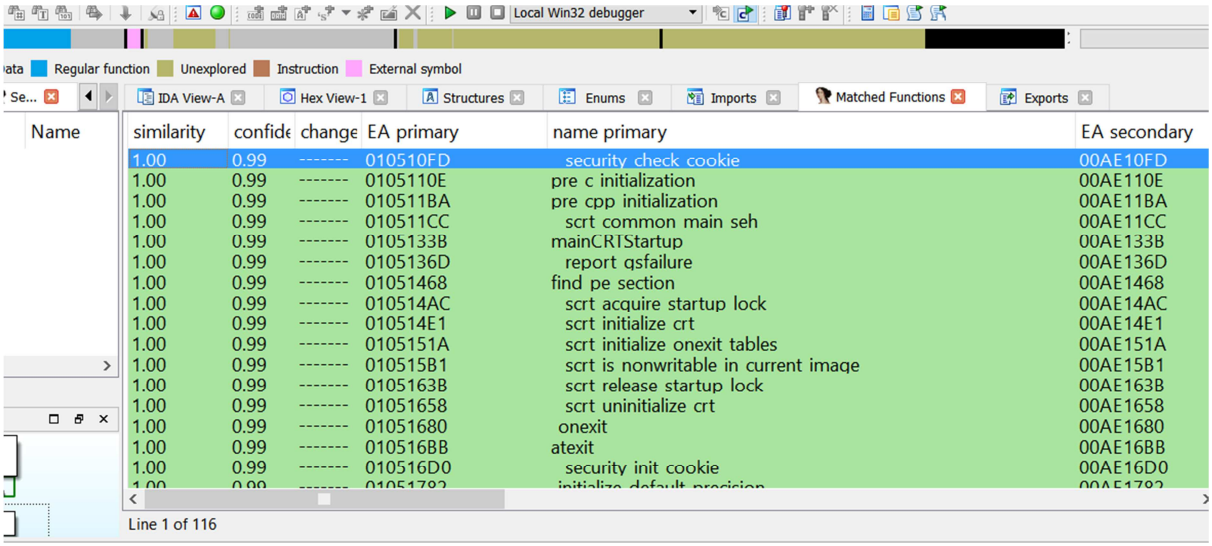


And I press DIFF DATABASE and look for the patched version to compare with it.

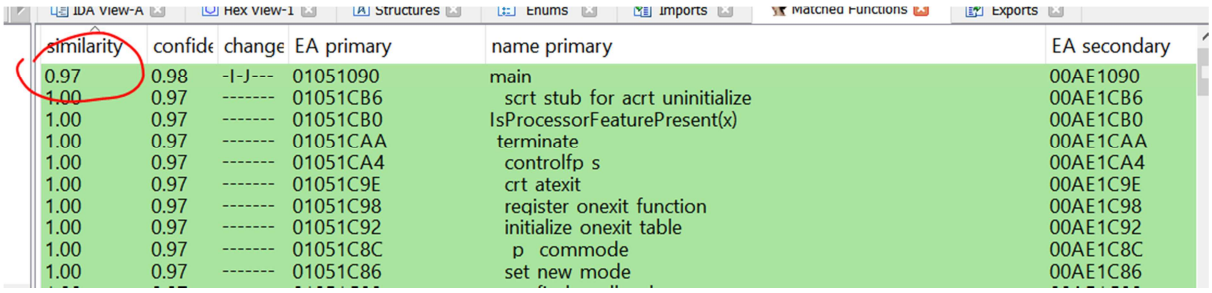




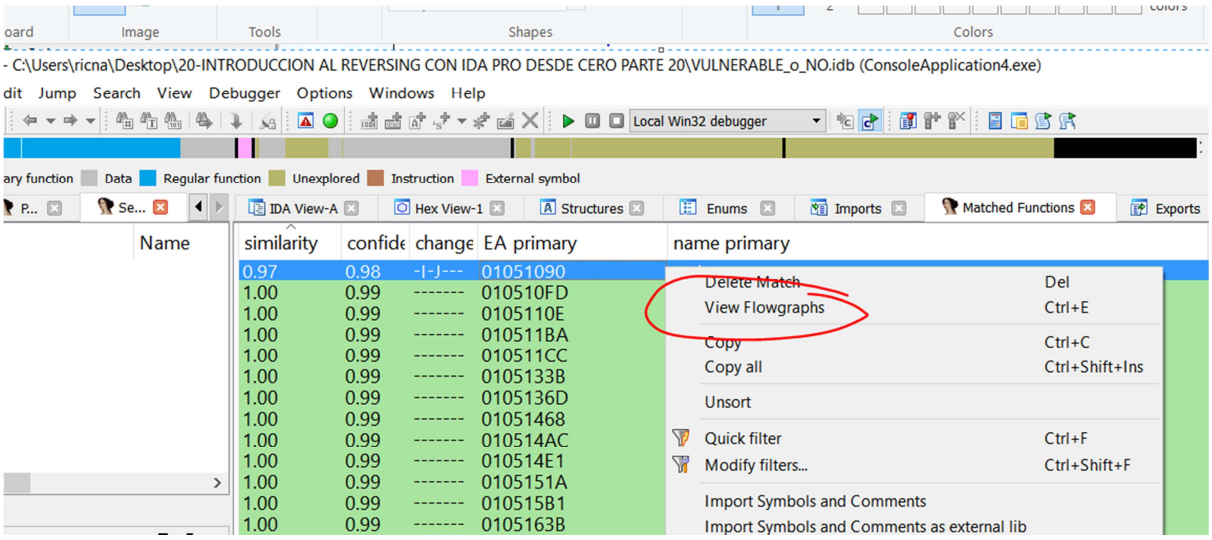
Once it is finished it shows me the results. Sometimes, it is not comfortable to see it in a column. So, I can drag it and drop it on the tab bar.



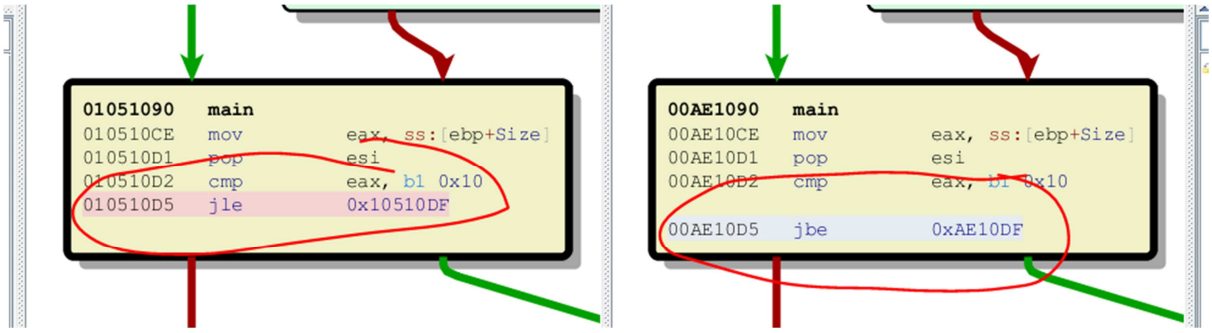
The first column shows the similarity. Those that say 1.00 are the same and the lower the number, the more different they are. It is convenient to click on the top of that column to order them from more different to more similar.



We see that there is only one with similarity smaller than 1.

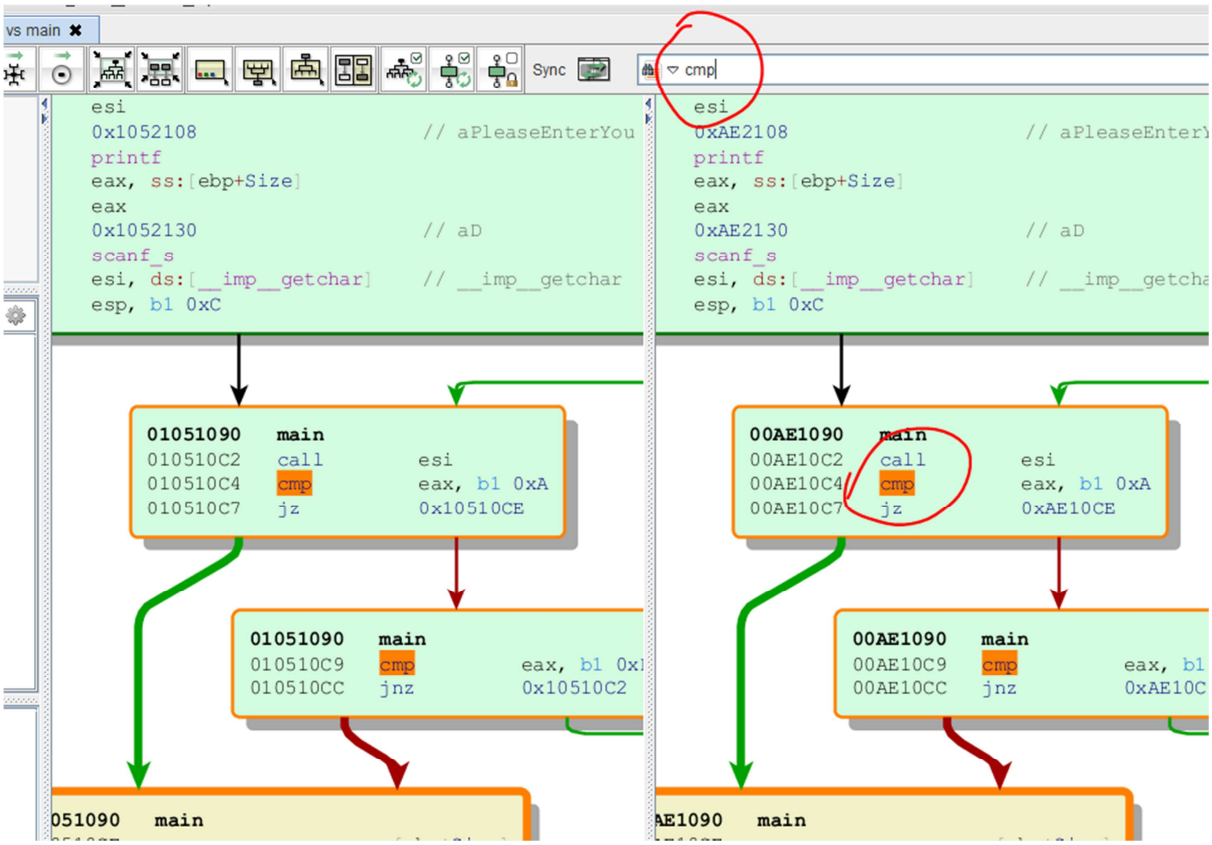


The green blocks are similar. The yellow blocks have some change and the red or gray ones are added.

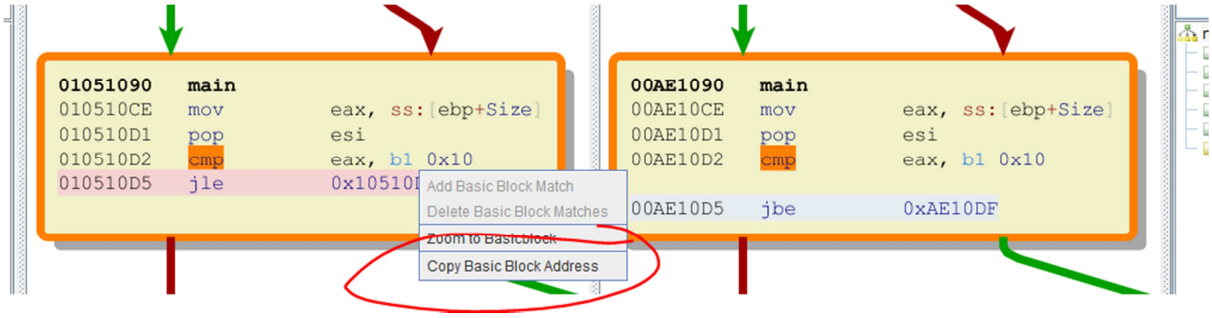


There we see the change. As we know to change a JLE jump by JBE, it is something that can avoid a BUFFER OVERFLOW, therefore if in a program of which we have the vulnerable version and the patched version and looking at the changed functions in some of them we find this we will know that we will have to reverse that function statically to see if it really is the vulnerable of the program.

One of the advantages of BINDIFF over the other two is that the graph is interactive, not an image only, it has a search engine above.

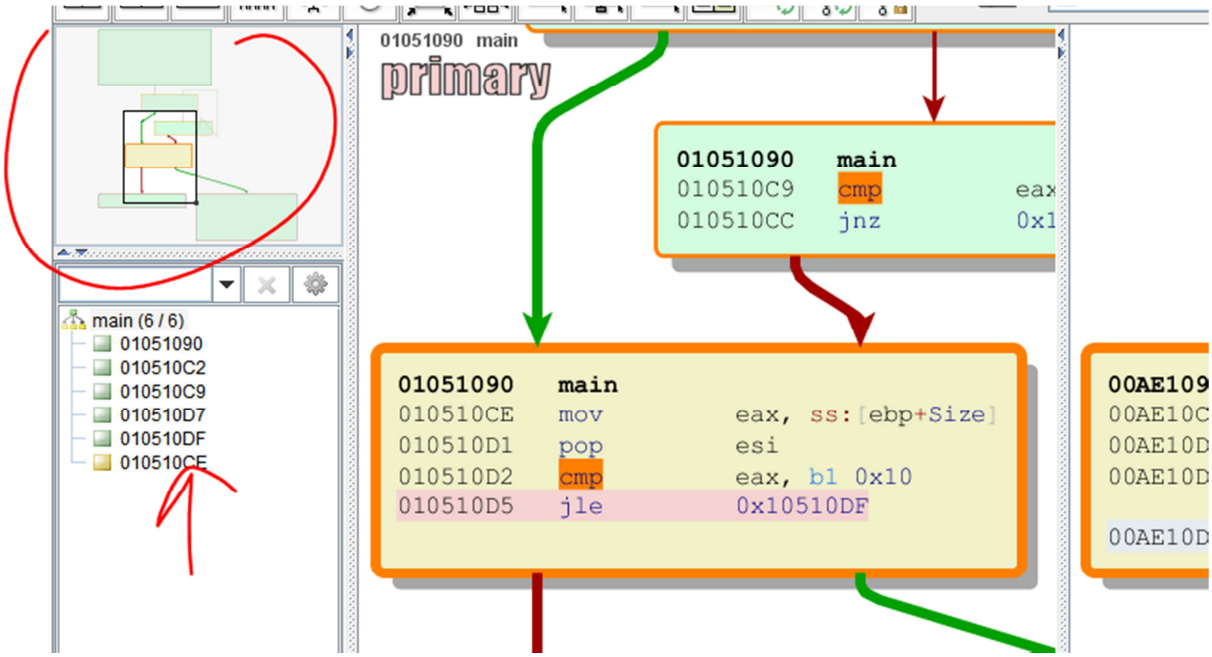


This is very useful, and you can also find addresses and any text that is on the chart.



We can copy the address of the block to paste it in IDA and go there.

We also have a graphical browser to navigate the function and list of blocks.

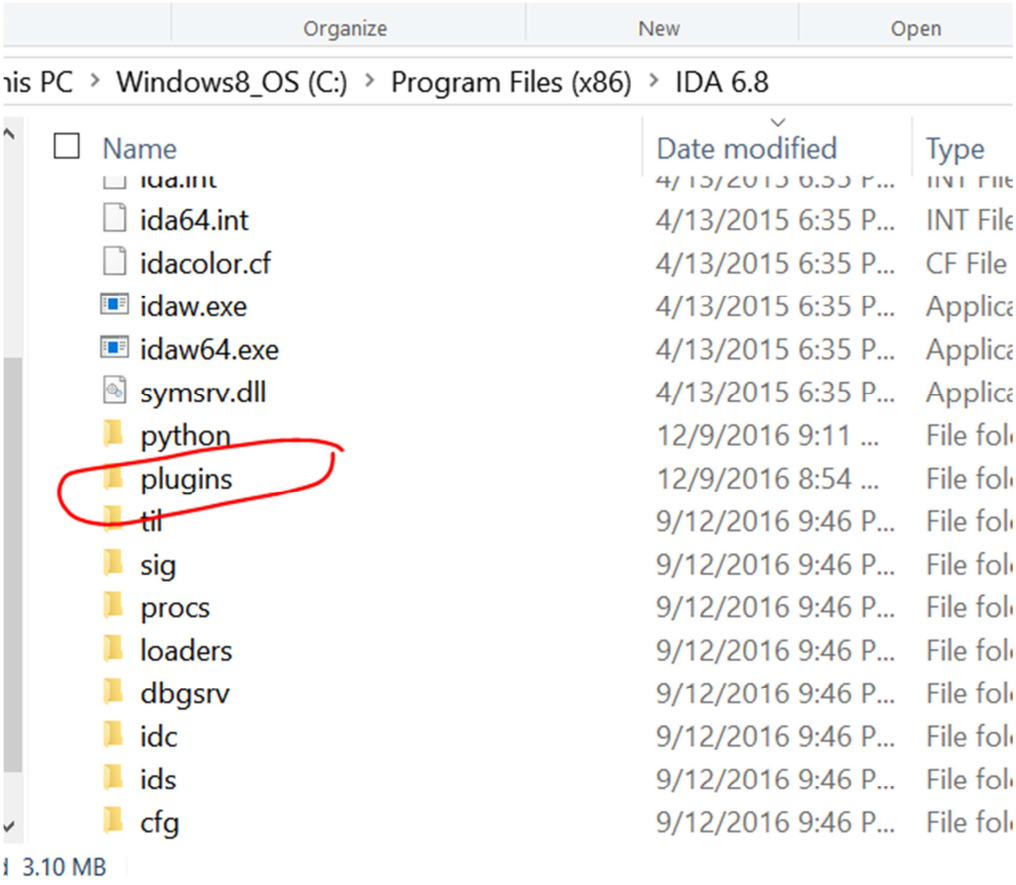


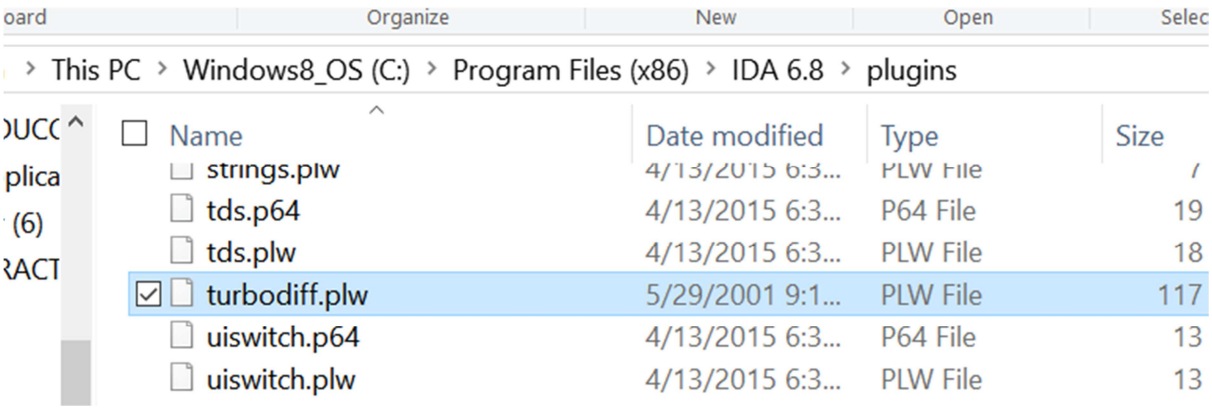
We can mark a block and in the menu we have **select ancestor** or **select succesors** so that we obscure the blocks of the path inside the function to reach the same initial block, or from there, in this case is a simple function, but in large and complex finding the way to a block is very important.

There are many good things about the Bindiff in the graphic section, it has some problems of match in big programs, but it is one of the best options to take into account.

TURBODIFF

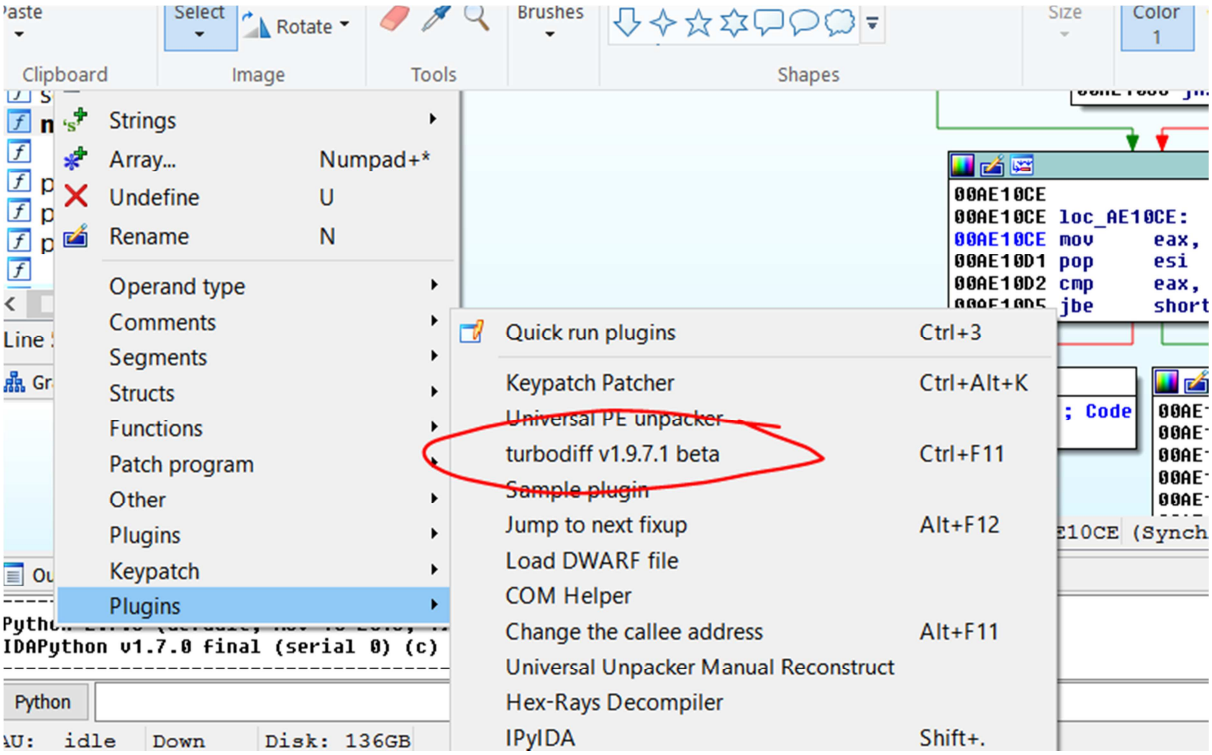
It is a plw created by my colleague Nicolas Economou from Core, it will be attached with the tutor, it can also be downloaded from the website of CORE SECURITY but it is an earlier version than the one I attached.The PLW is copied to the plugins folder inside the IDA installation folder.



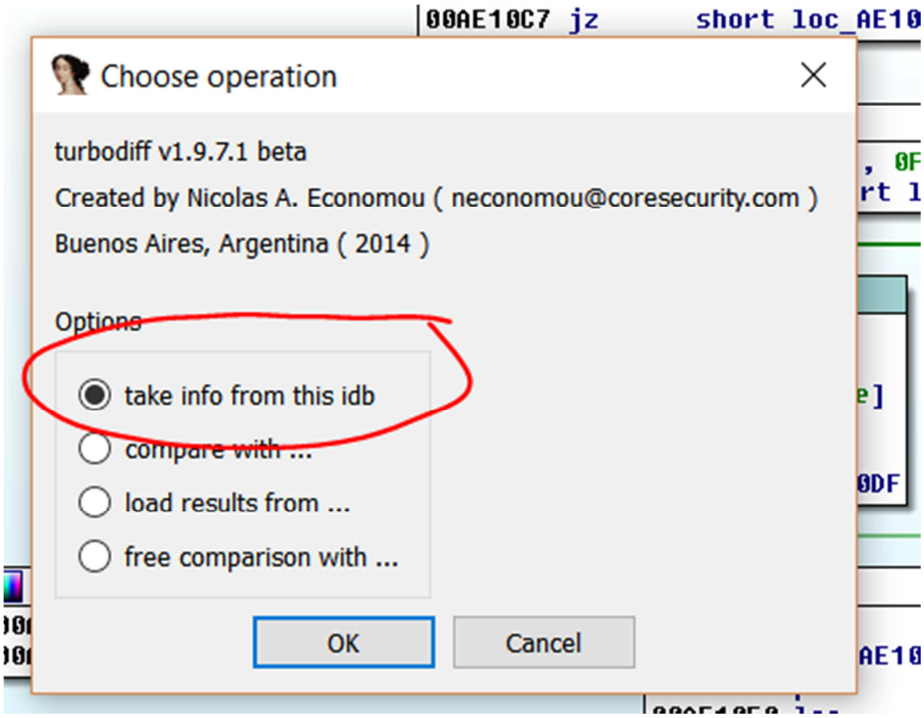


I will have to restart the IDA to load it.

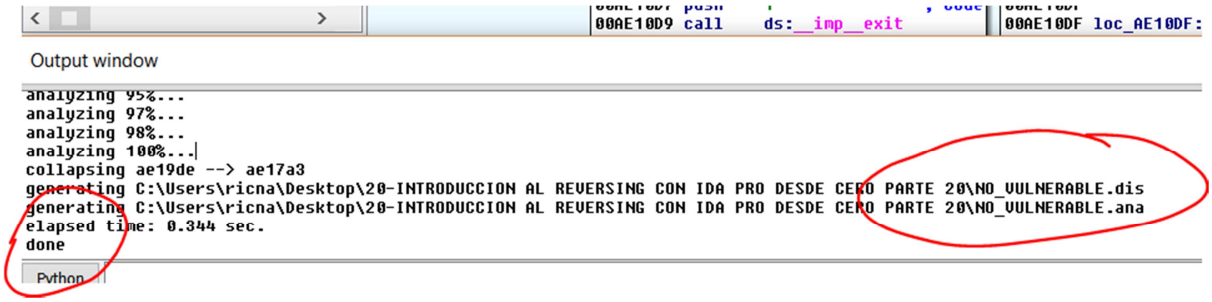
As always I load the non-VULNERABLE version first.



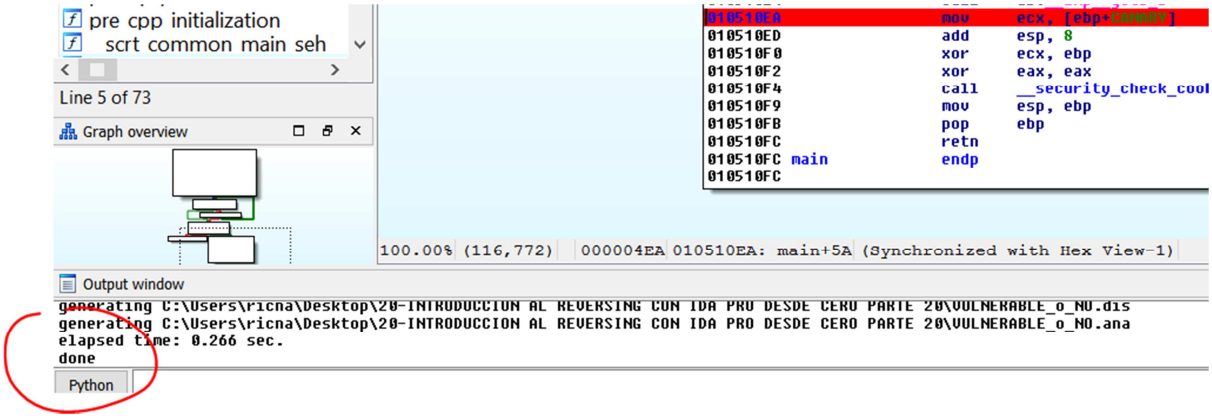
It is necessary to take the information of each idb to be compared.



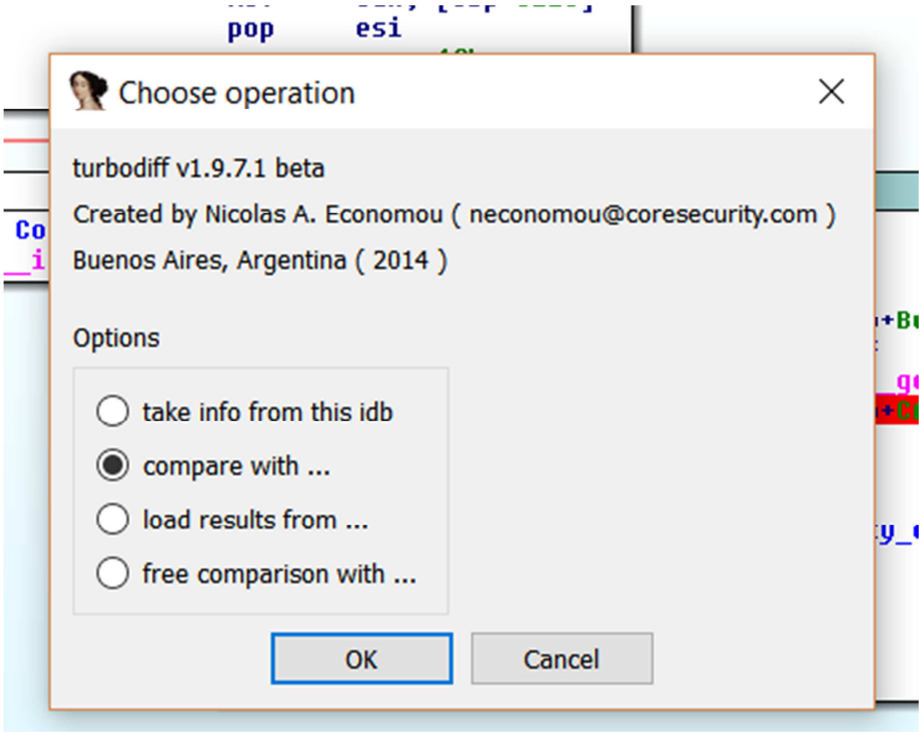
So I do that in this first one.

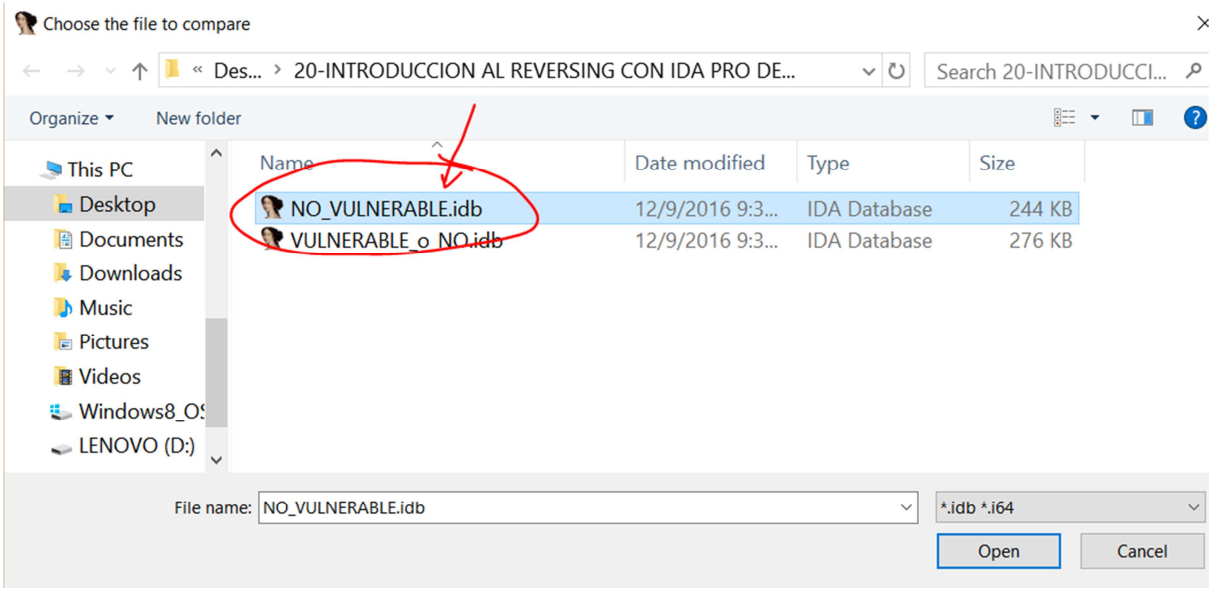


Then I open the vulnerable and do the same.

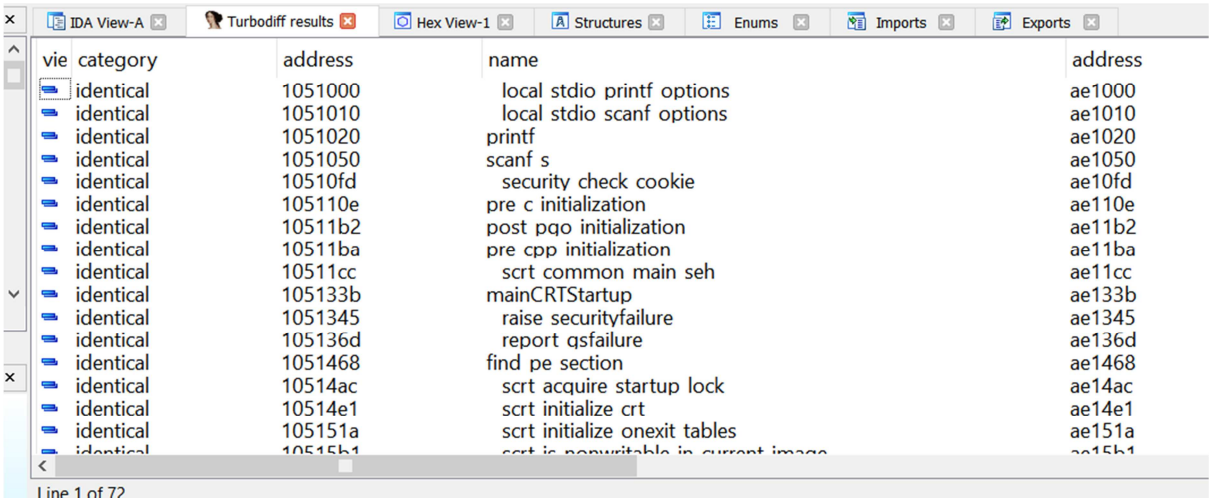


Then from the vulnerable I call the plugin again.

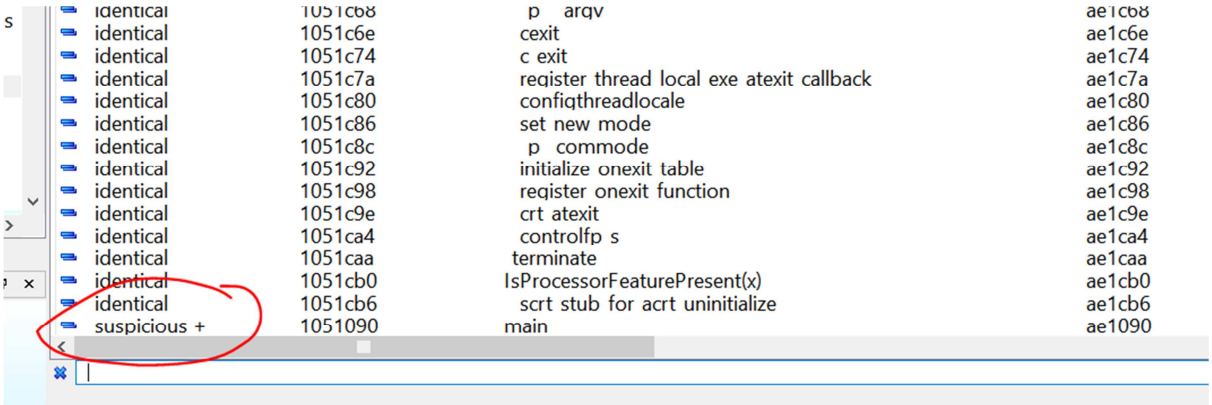




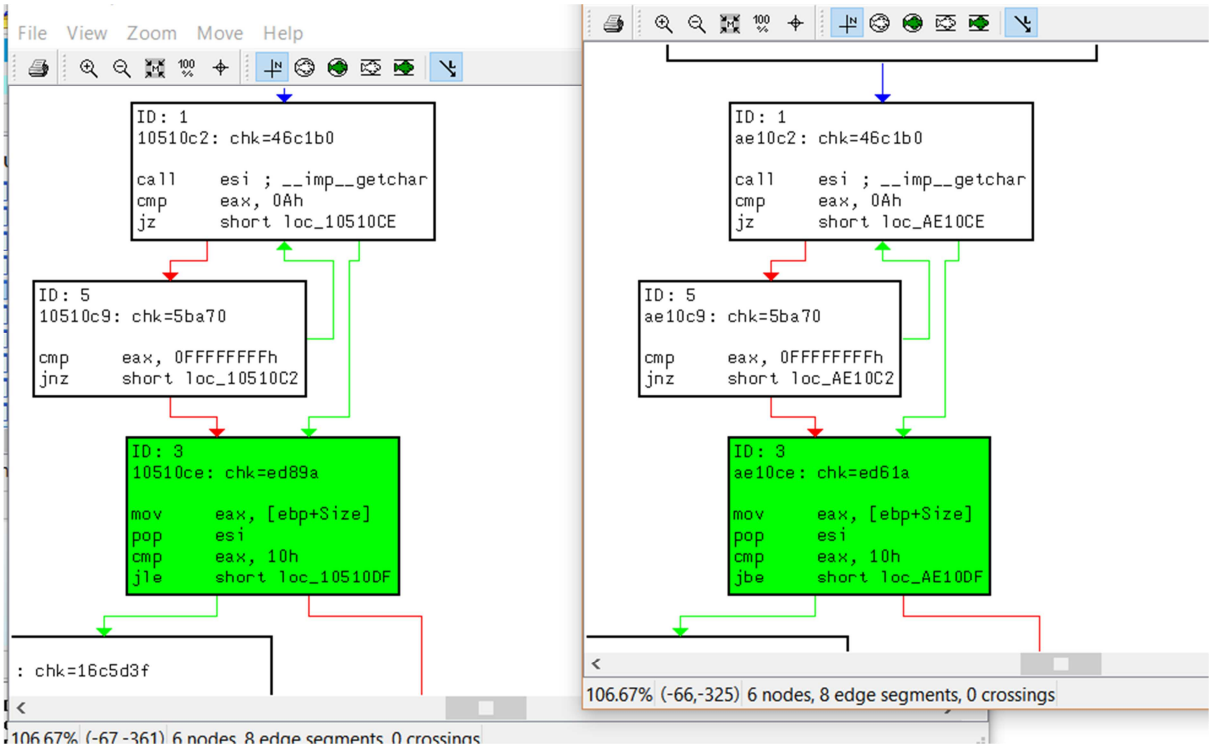
I look for the non-vulnerable version and accept the options that it brings by default.



There I can press CTRL + F and look for **changed** or **suspicious** to show me the changed ones.



There it is. Double click.

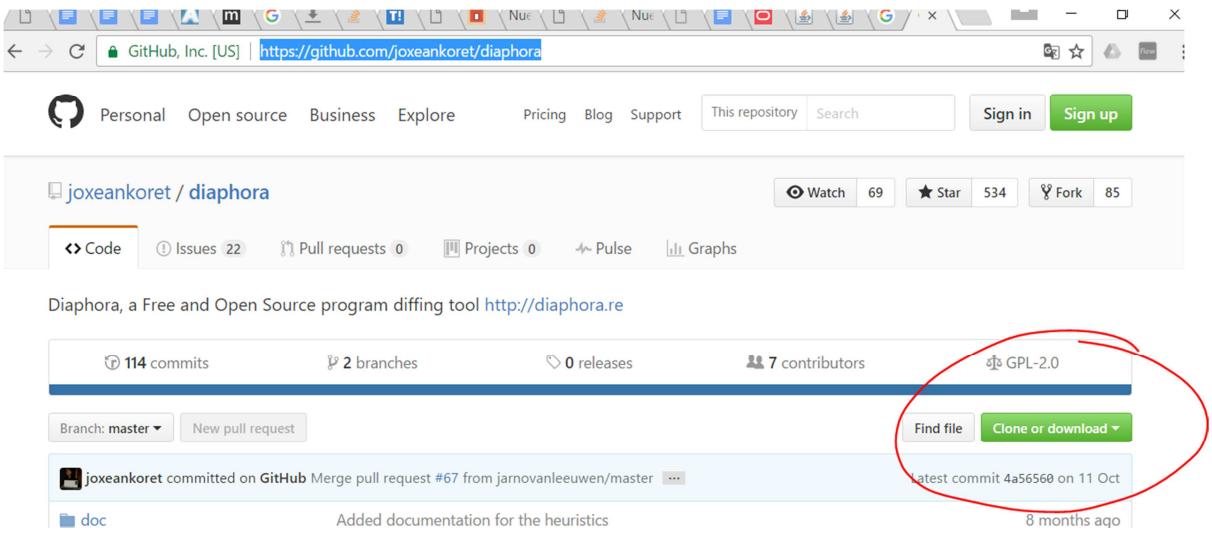


There, you will see the changed ones. There is also a color code according to the exchange rate, green for the blocks with minimum changes, yellow for the very changed blocks and red for the added blocks. Obviously, the graphics are pictures and they are not interactive, but it is a very fast Differ. It really is the fastest. It is very noticeable in large executables and does not show too many stupid changes like Bindiff, assuming many as not important which in big works is appreciated.If one does not like the shape of the graphics can use both differs at a time and then see the results in the graph of the Bindiff.

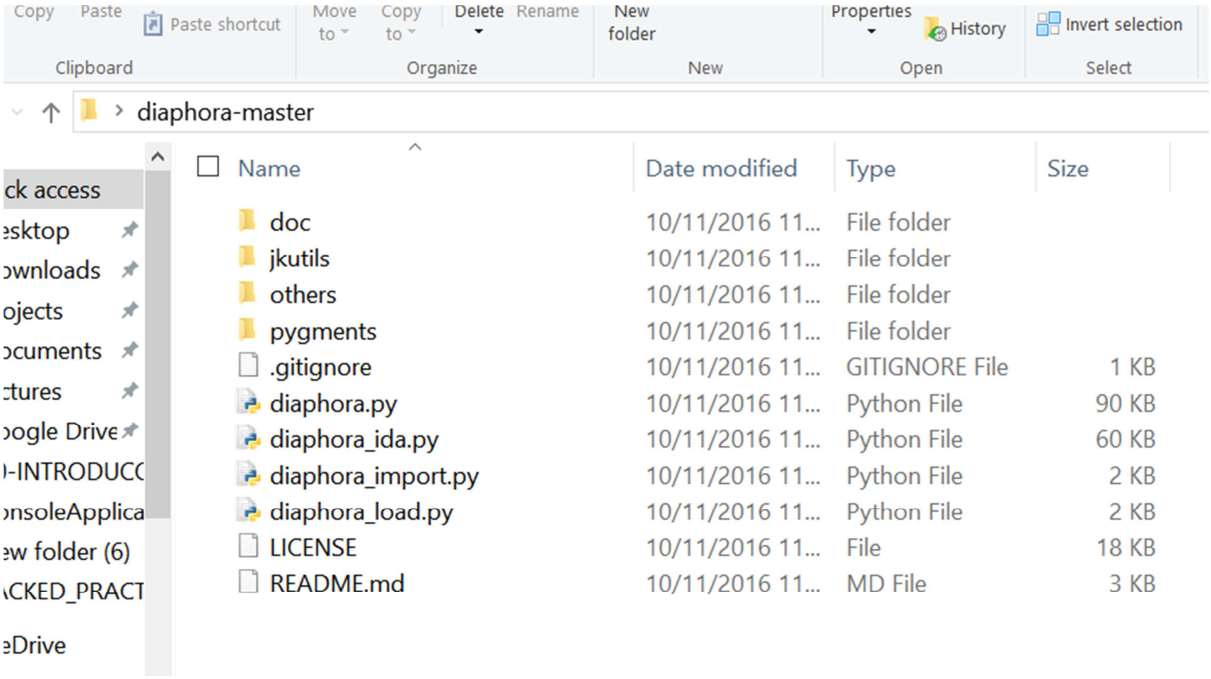
DIAPHORA

The Diaphora is a plugin made in python by Joxean Koret.

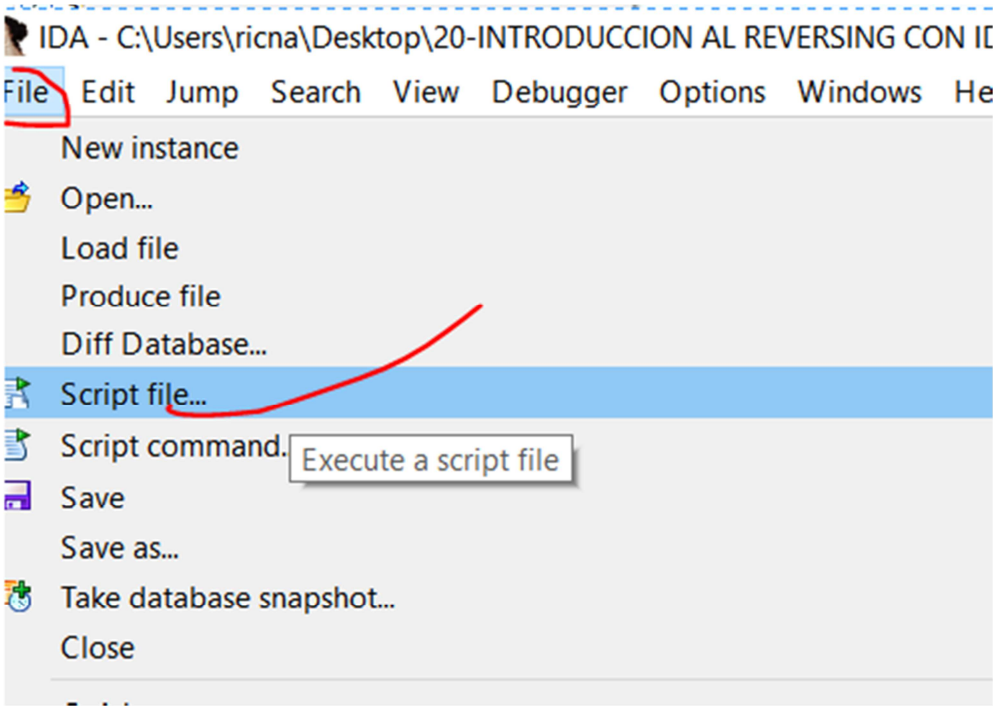
https://github.com/joxeankoret/diaphora



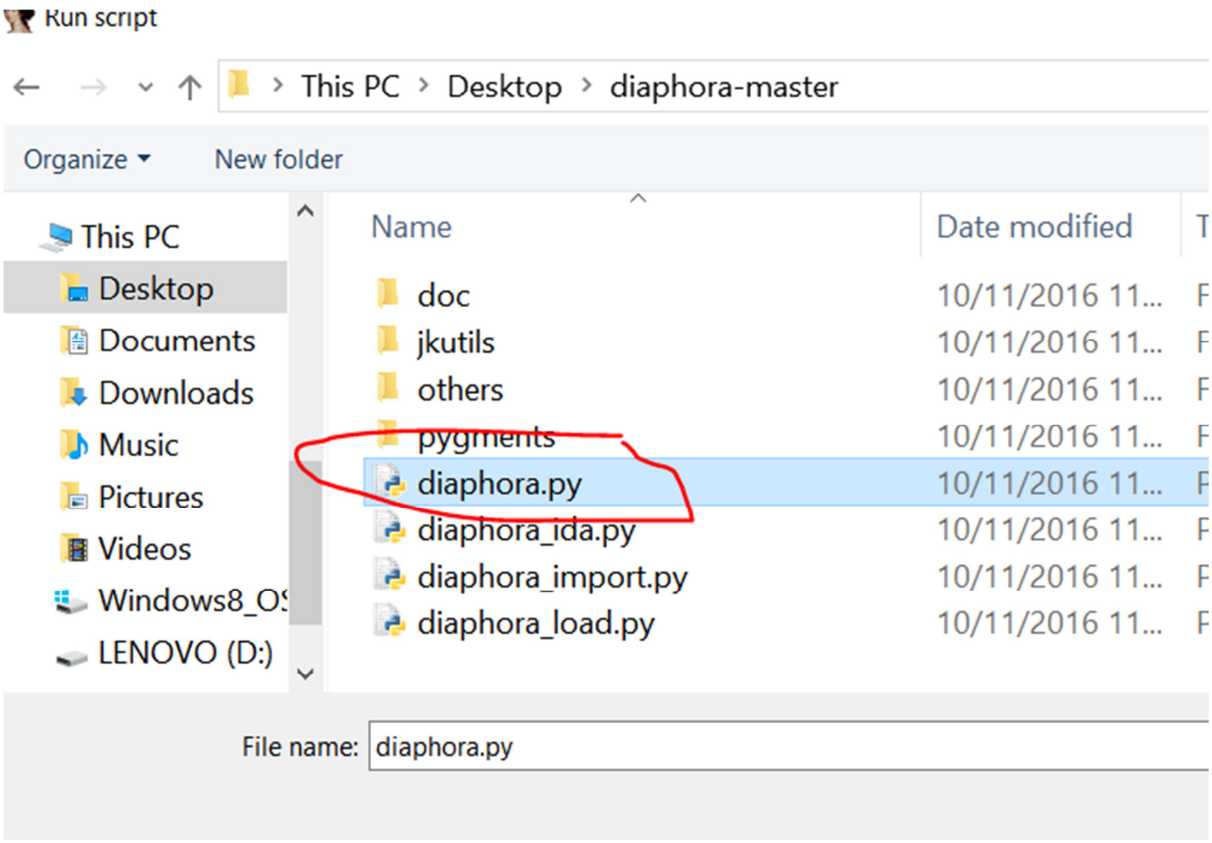
It is not necessary to install it. I can decompress it anywhere and it is only necessary to have Python installed on the machine which if we have IDA we will have it.

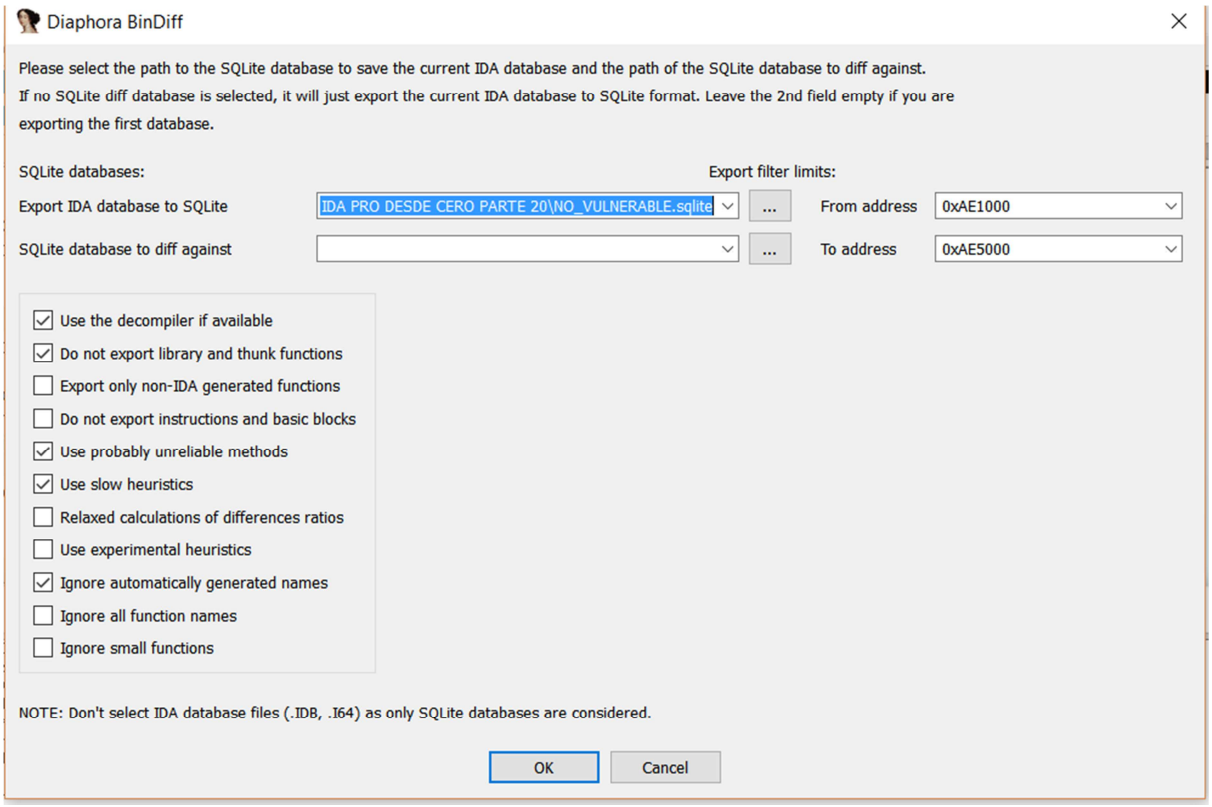


So we will go as usual. First the patch or not vulnerable in IDA.

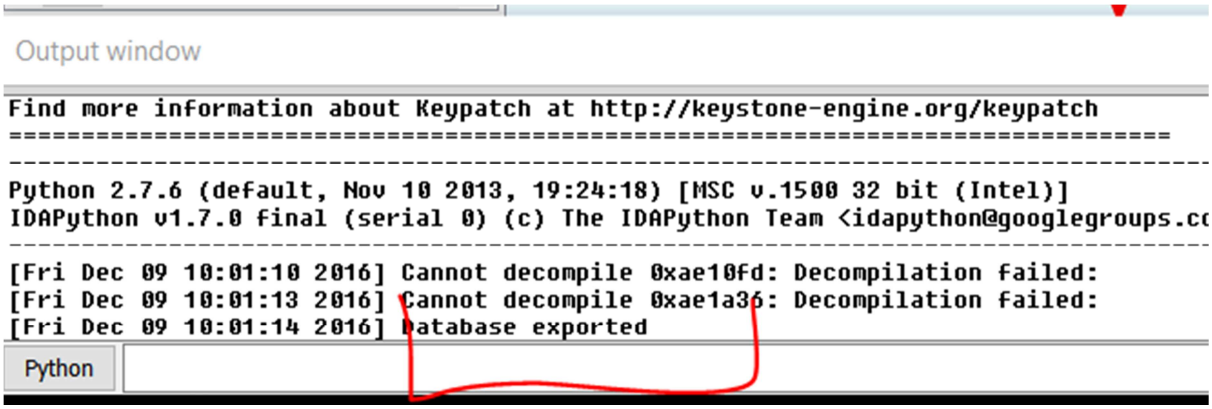


In FILE-SCRIPT FILE open the search engine and go where we decompressed the Diaphora and look for **diaphora.py**.

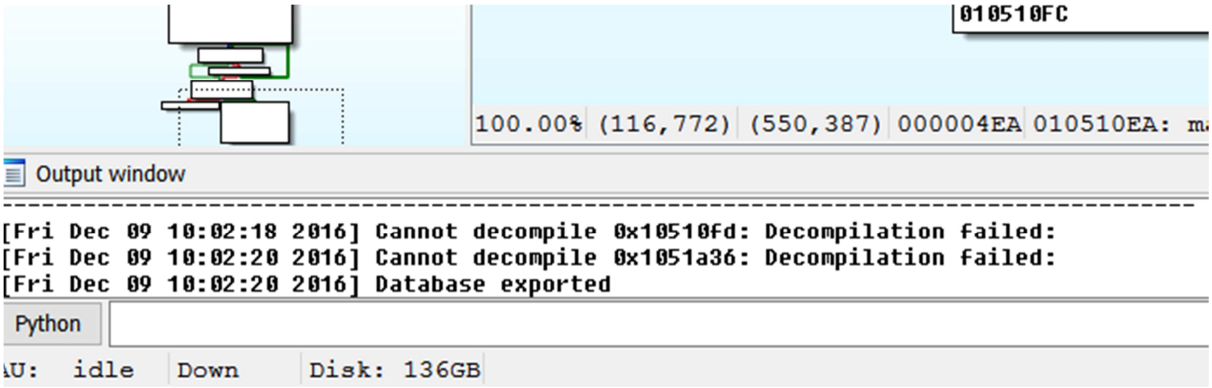




As it comes, we press OK to export the database to SQL.



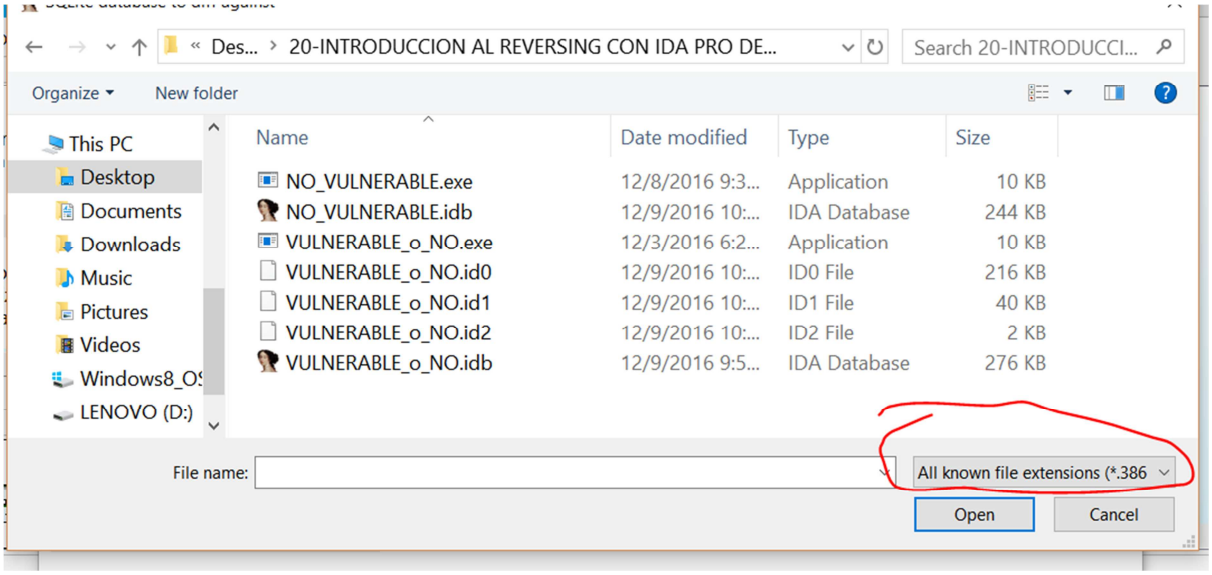
When it finishes we open the vulnerable in IDA and we do the same we open diaphora.py and without changing anything we export the database.



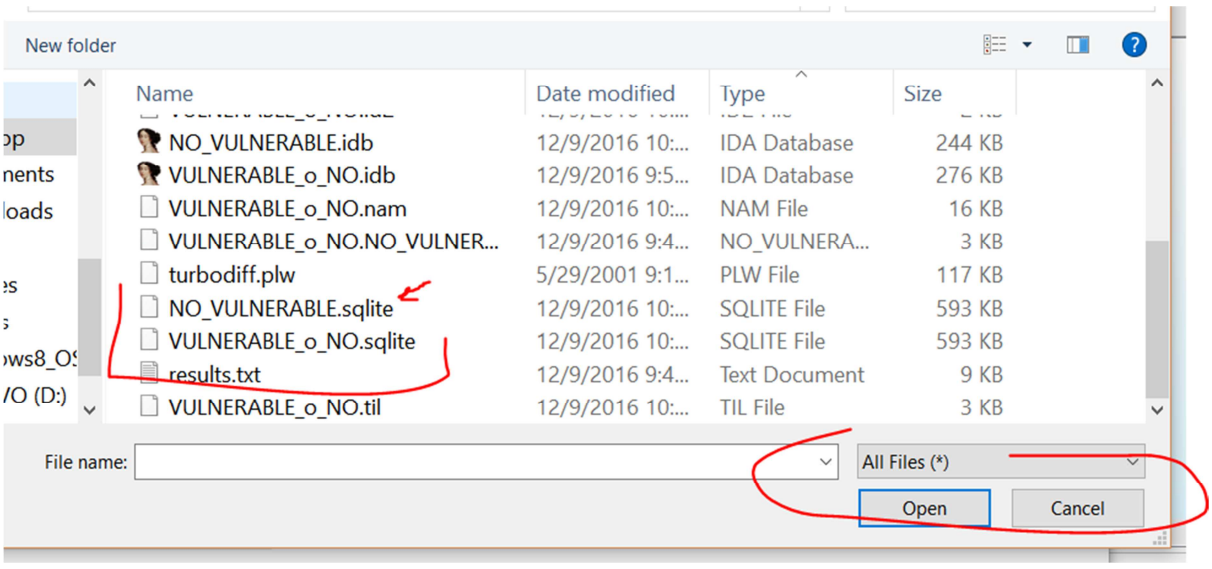
Once we did the same in both, we reopen the diaphora.py in the vulnerable version but this time.

In the second place we look for the SQL database of the patching that it exported before.

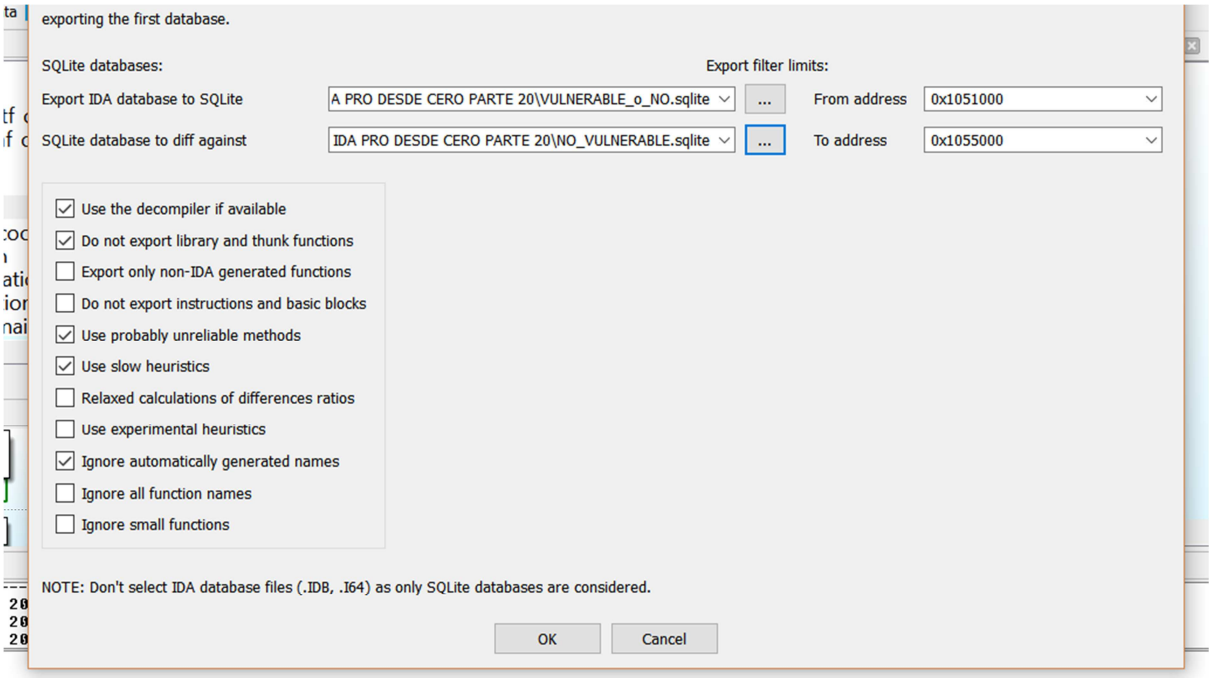
We see that when we go to the folder, there seems to be nothing



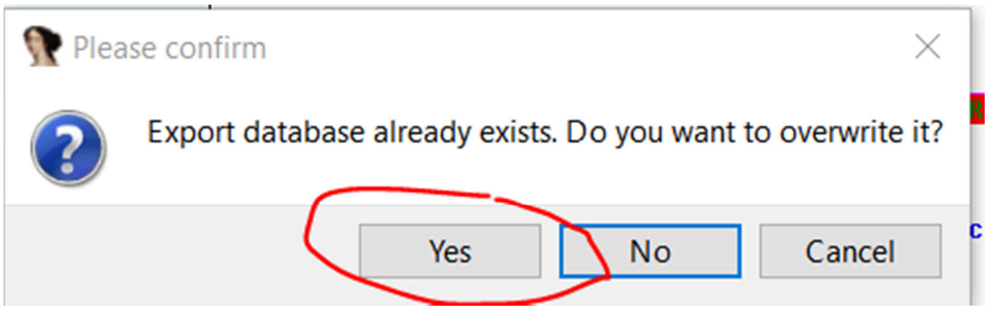
But it's because of the file filter. We change it to show everything.



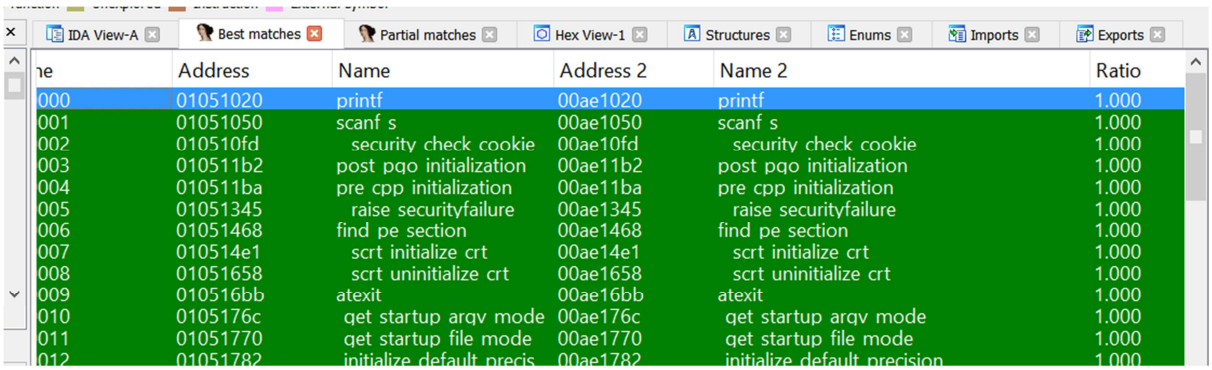
And we look for the non-vulnerable.



I press OK, just like this.

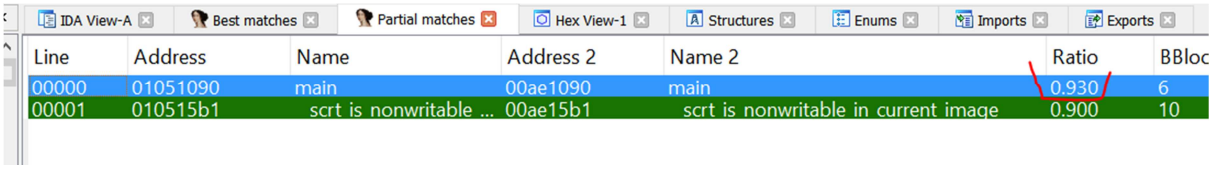


We see that there is a BEST MATCHES tab with which there is no doubt that they are the same.

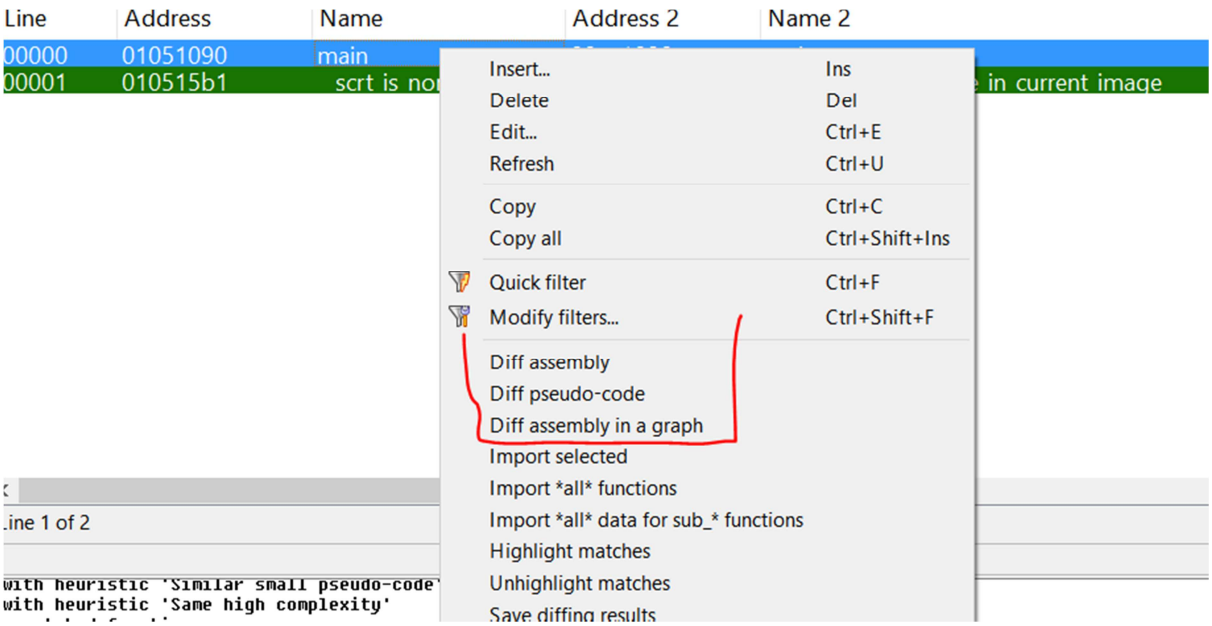


ne	Address	Name	Address 2	Name 2	Ratio
000	01051020	printf	00ae1020	printf	1.000
001	01051050	scanf_s	00ae1050	scanf_s	1.000
002	010510fd	security_check_cookie	00ae10fd	security_check_cookie	1.000
003	010511b2	post_pgo_initialization	00ae11b2	post_pgo_initialization	1.000
004	010511ba	pre_cpp_initialization	00ae11ba	pre_cpp_initialization	1.000
005	01051345	raise_securityfailure	00ae1345	raise_securityfailure	1.000
006	01051468	find_pe_section	00ae1468	find_pe_section	1.000
007	010514e1	scrt_initialize_crt	00ae14e1	scrt_initialize_crt	1.000
008	01051658	scrt_uninitialize_crt	00ae1658	scrt_uninitialize_crt	1.000
009	010516bb	atexit	00ae16bb	atexit	1.000
010	0105176c	get_startup_argv_mode	00ae176c	get_startup_argv_mode	1.000
011	01051770	get_startup_file_mode	00ae1770	get_startup_file_mode	1.000
012	01051782	initialize_default_precis	00ae1782	initialize_default_precision	1.000

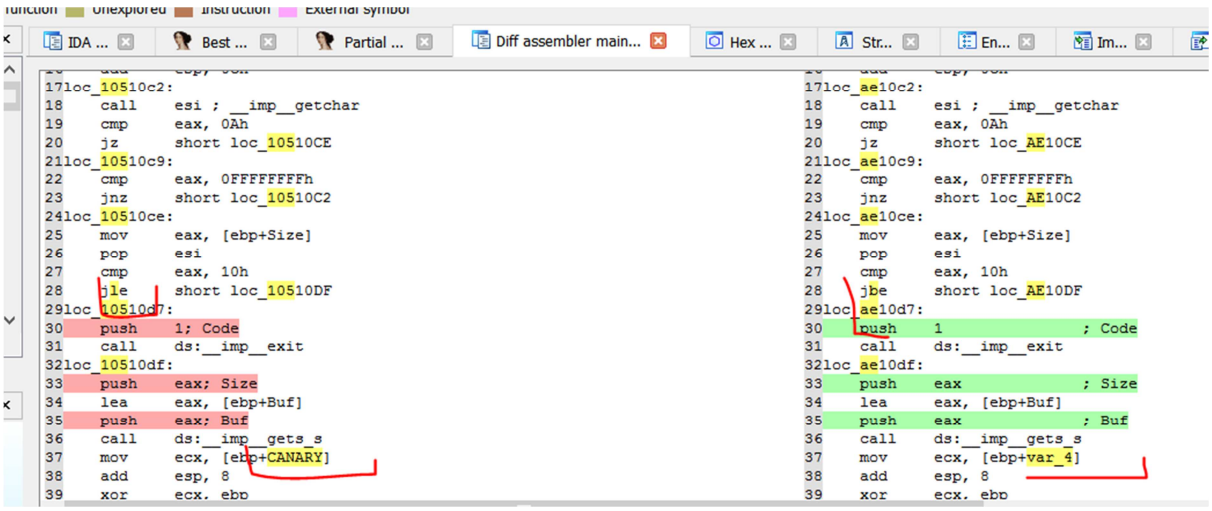
In the tab PARTIAL MATCHES we see the ones that possibly were changed.



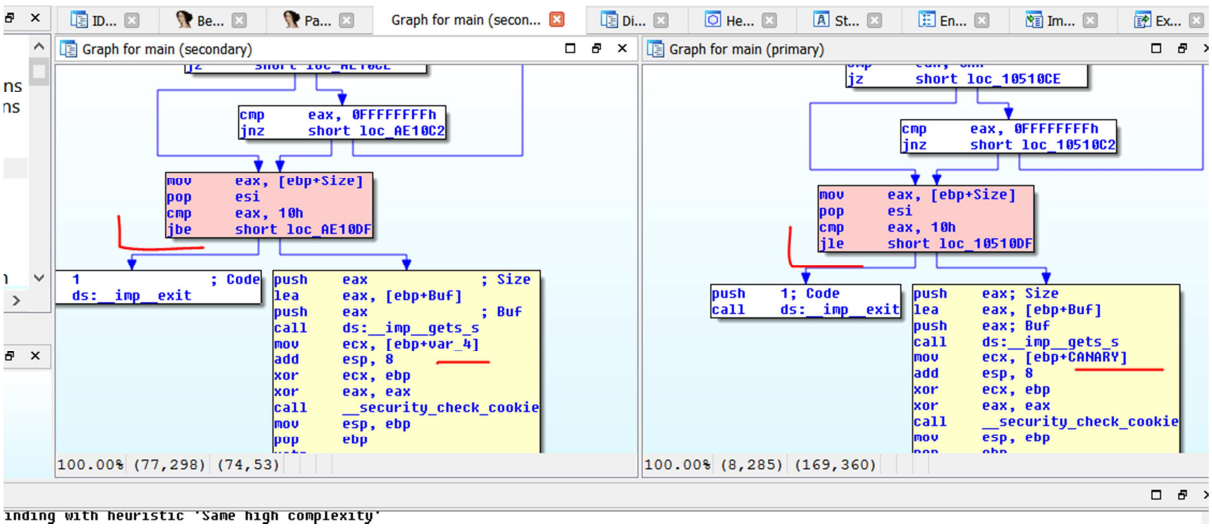
There, the version is changed. We see that it found two changes. One of the things that Diaphora has is that it is very precise. Sometimes, that's good, but sometimes when you have hundreds of functions, you want it to be a little more relaxed and not show as many nonsense as changes.



We see that the first DIFF ASSEMBLY has several options to graph.

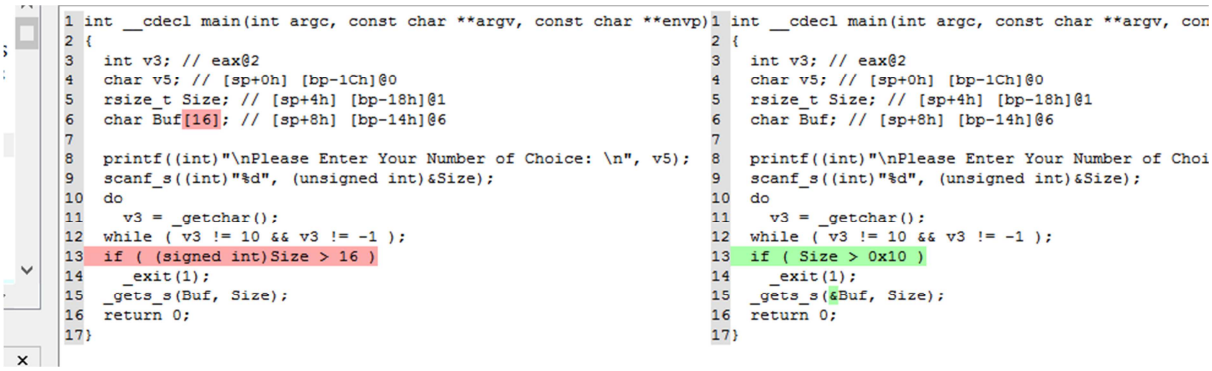


It's like very precise and detailed but when you see a hundred functions like this you want to die. Let's see the second option **DIFF ASSEMBLY IN A GRAPH**.



This graph is a little better, although it is not interactive, the important block changed is in red and in yellow those that have minor changes like the name of a variable.

The other **DIFF PSEUDO CODE** option uses the HEX RAYS plugin that comes with the included IDA that tries to build a source code from the executable.



We see that in the vulnerable that we had reversed by hand and determined that there was a buffer of 16 bytes, that variable Buf is detected as buffer, but in the other as we did not do the same work it does not detect it but as a variable char nothing more. It also shows that the variable is signed in the vulnerable and in the other it says nothing which is supposed to be unsigned. Another characteristic of Diaphora is that it is the slowest (it is programmed in Python and turbodiff in C) and in big executables the analysis and mach are very long.

I attached the IDA1.exe file. I would like to be analyzed and see if it is vulnerable and also if you can overflow the buffer and modify the flow of the

program to show us the msg of good guy. The exercise is openly discussed on both the Crackslatinos mailing list and our telegram group.

https://telegram.me/CLSExploits

Ricardo Narvaja

Translated by: @IvinsonCLS