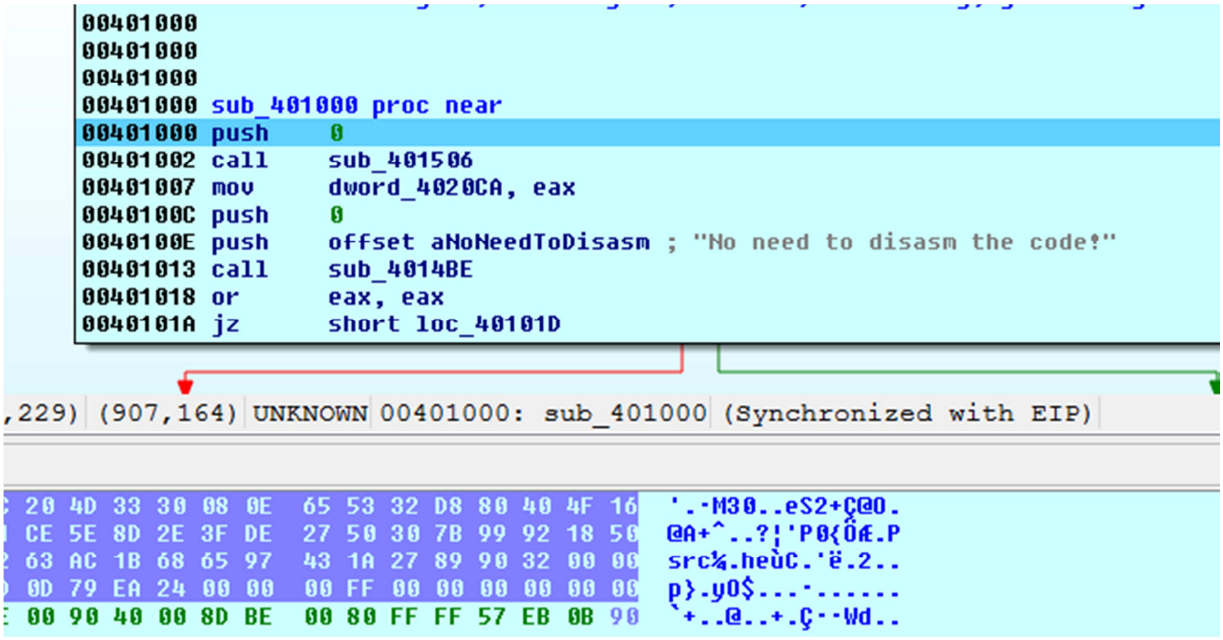
**REVERSING WITH IDA PRO FROM SCRATCH**
**PART 15**

In the previous part, we saw a pair of many methods to detect and get the OEP in a packed file. Now, we'll continue with the two missing steps: The dumped and the IAT rebuilding explained here.

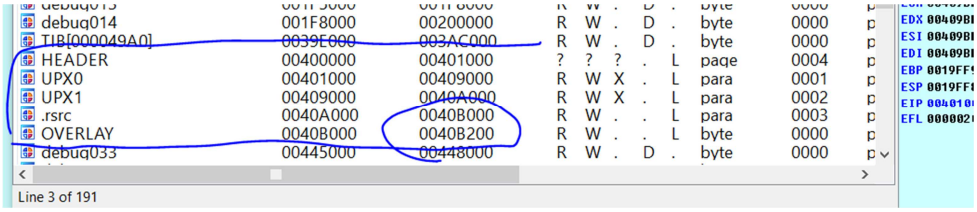


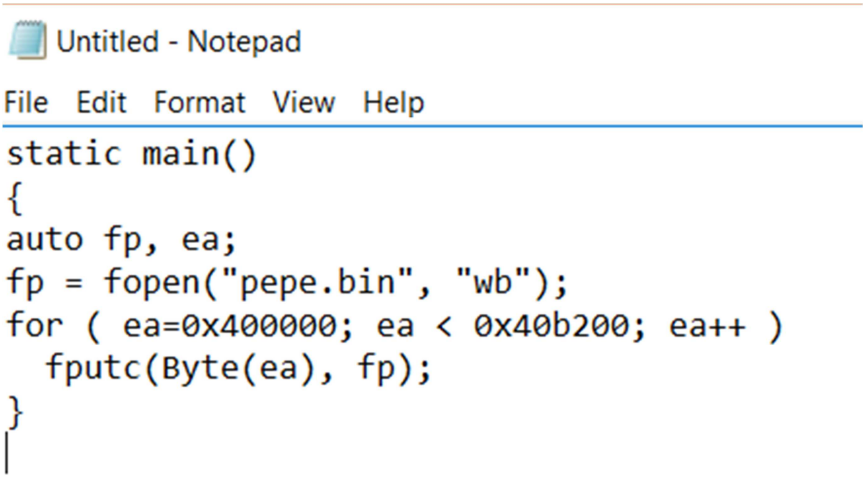

We go to the OEP and Reanalyze the program to have everything ready to Dump.

Let's use an IDC script not Python.

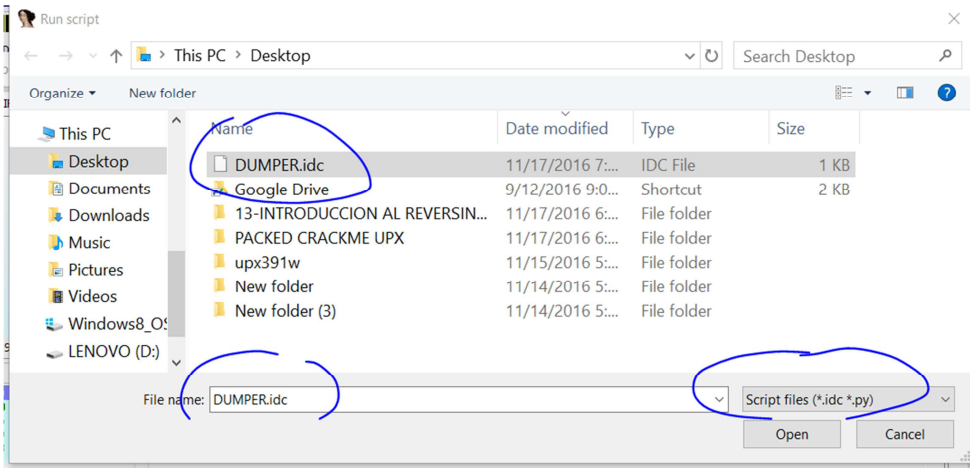
```
static main()
{
auto fp, ea;
fp = fopen("pepe.bin", "wb");
for ( ea=0x400000; ea < 0x40b200; ea++ )
  fputc(Byte(ea), fp);
}
```

In the script, we enter the ImageBase 0x400000 and the greater address we see in IDA Segments tab of the last section to be dumped, in this case, the Overlay section that ends in 0x40b200.

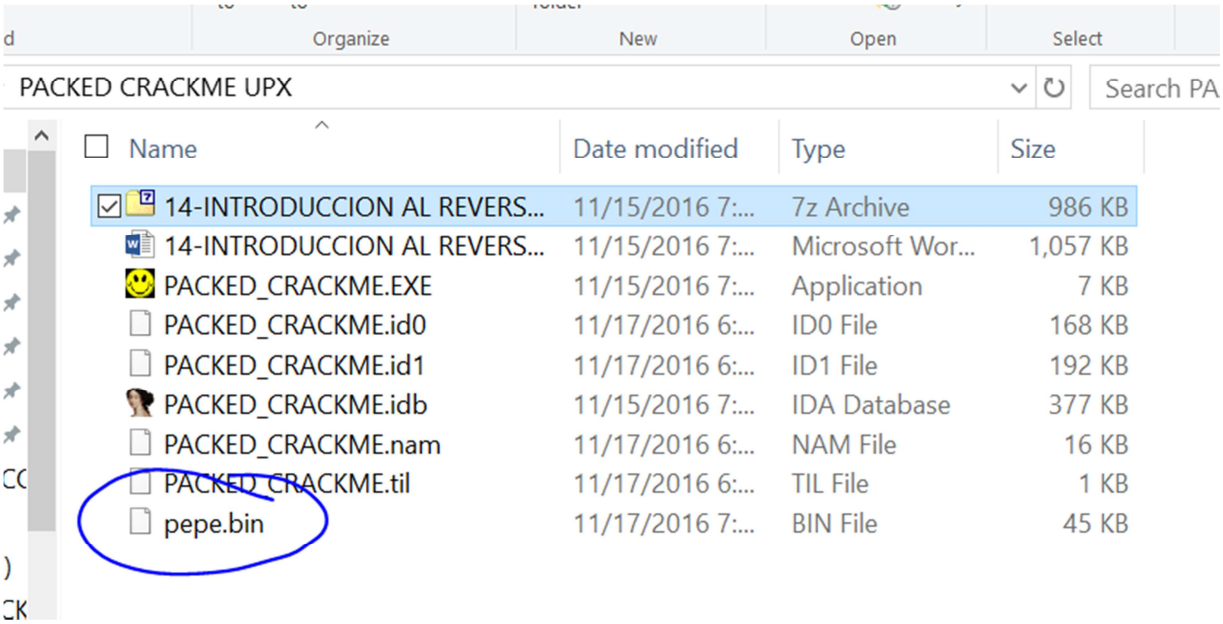


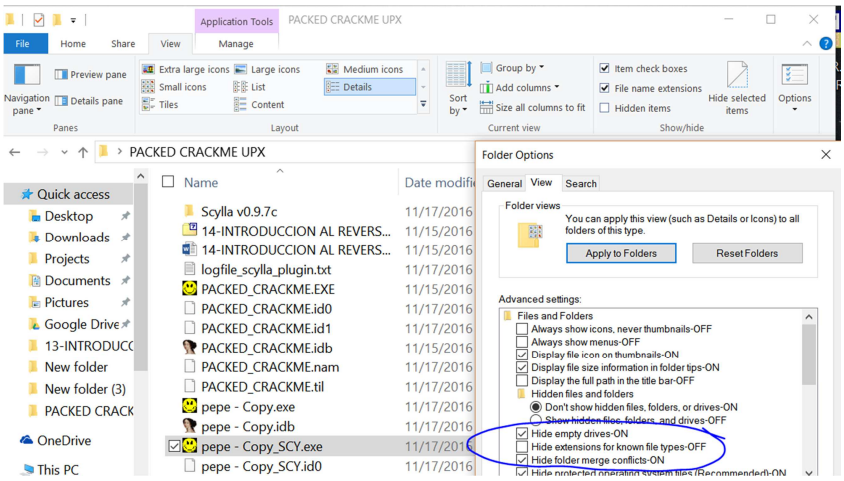

Let’s save it as DUMPER.idc

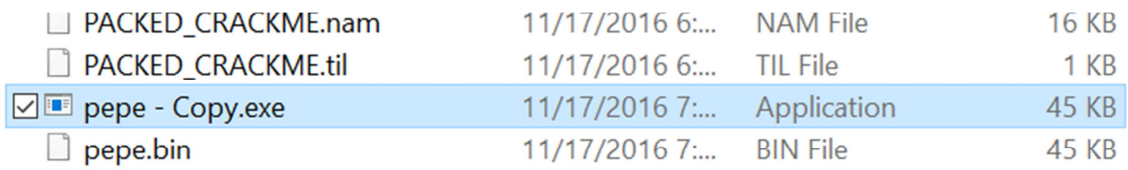



And run it in FILE-SCRIPT FILE. It accepts Python and IDC scripts. So, there’s no problem.



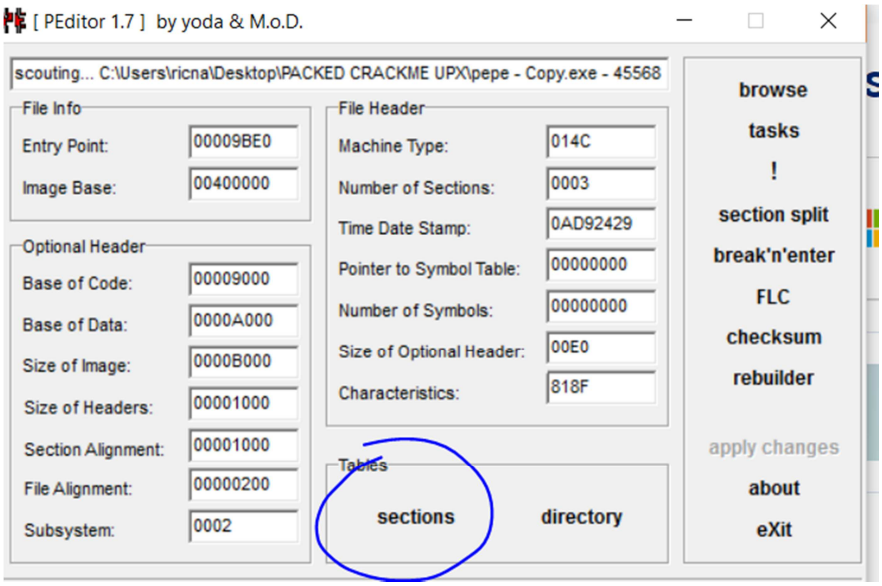

I make a backup and rename it with the **.exe** extension. (If you don't see the extensions, modify that option in Folder Options in Windows)



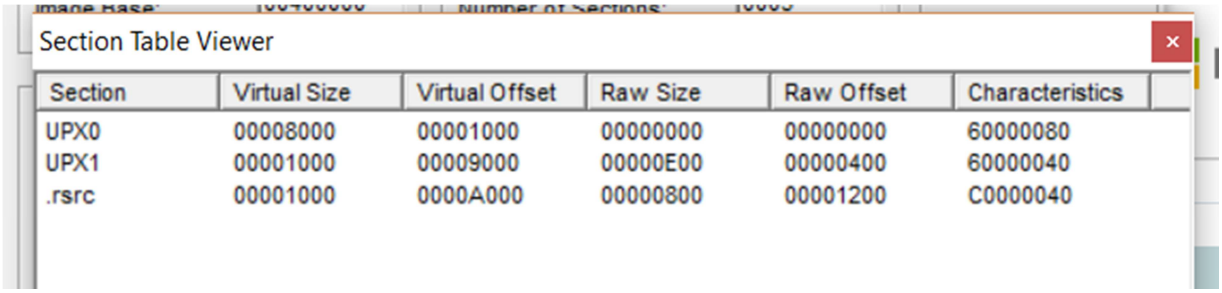



We see it doesn’t have any icon. So, some steps are missing.

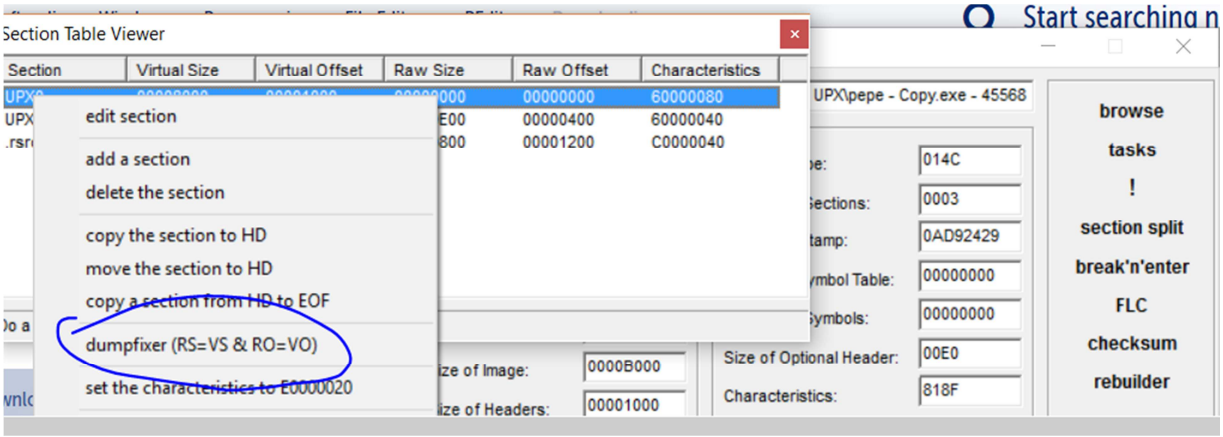
Let’s decompress PE Editor 1.7 and run it.

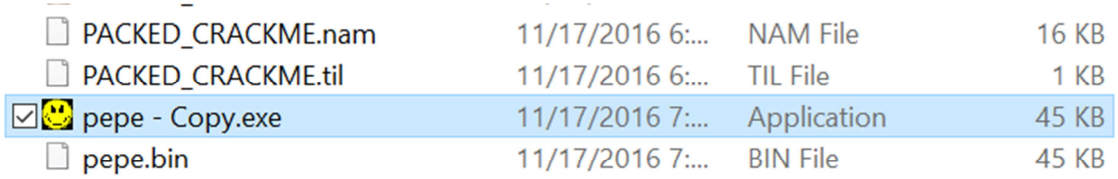



In SECTIONS.




Right click in each section and select DumpFixer.






We are closer. At least, we see the icon, but it doesn't run yet because we need to fix the IAT.

**What is the IAT?**

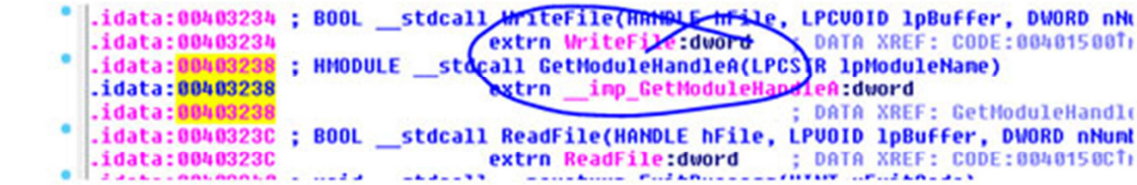
The IAT or IMPORT ADDRESS TABLE is a table located in the executable and it is used when the program runs. It saves the imported function addresses there used to run the program in any machine.

If the IAT is correct and we give another person the executable, the IAT will be filled with the correspondent values of that machine independently of the OS and it will run normally.

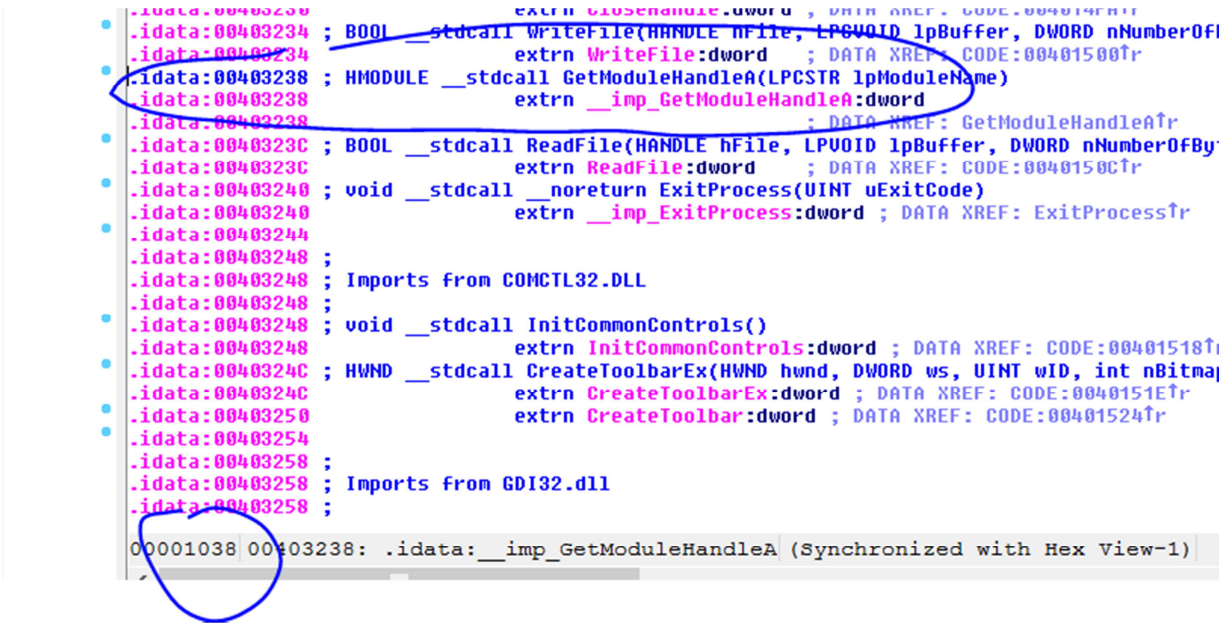
The IAT will be in a determined position in each executable and it will have fix positions for each function to fill it.

Let's see the difference between the top image that belongs to the packed file when it was at the OEP before unpacking and the original one.


The content of 0x403028 is an offset (off_) or the GetmoduleHandleA API address and in the original exe, the same address is in the idata section and it has the address of the same API..

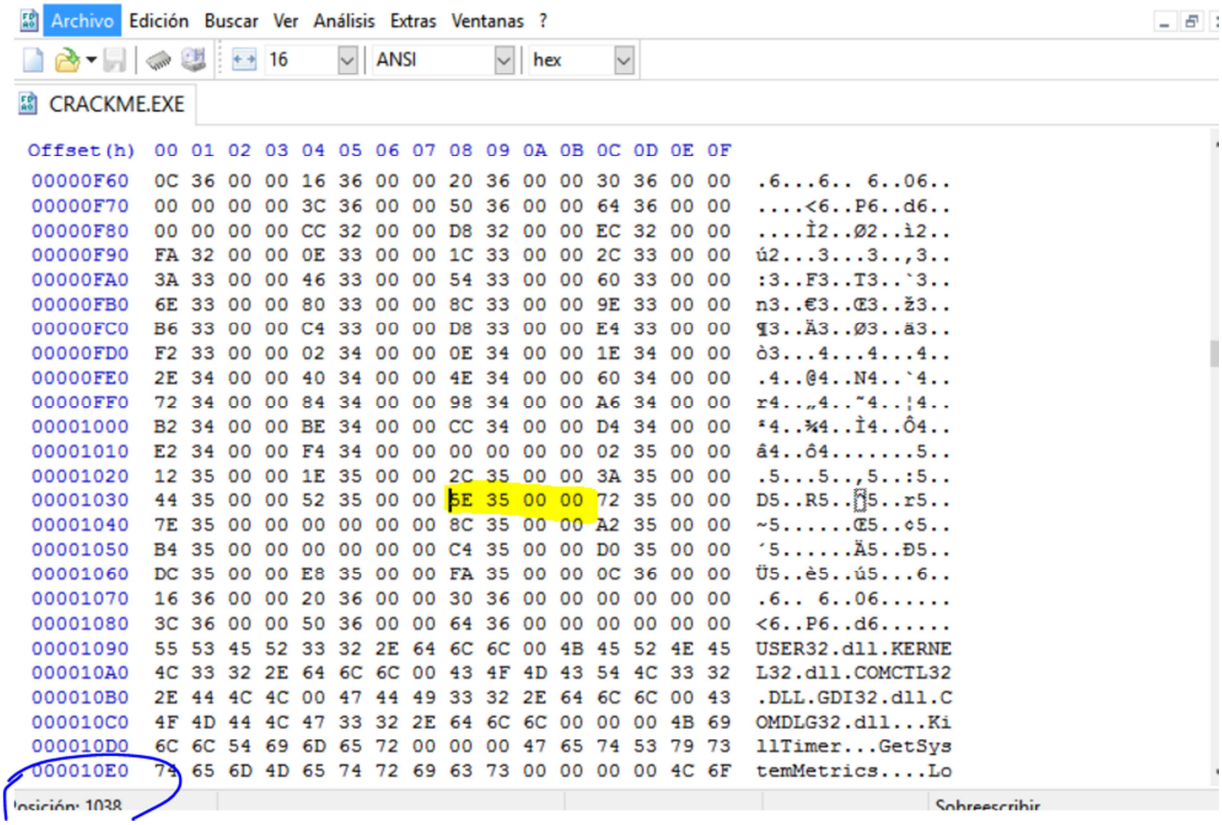


Both show the 0x403238 address and it seems they have the same content. Let's open the original program too.



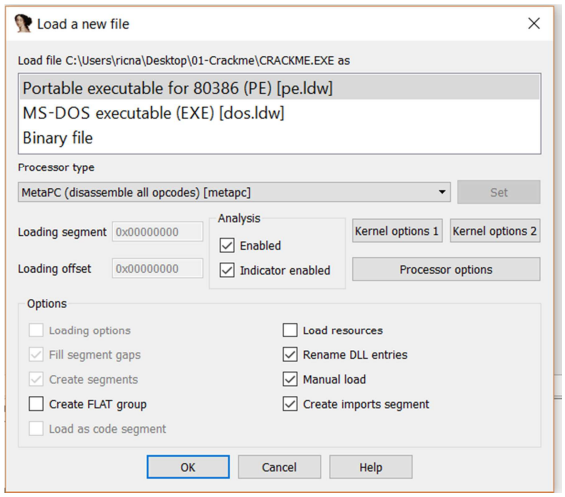

There, we see the 0x1038 File Offset to able to search in the HXD of the original one.

Let's open the original one in HXD.

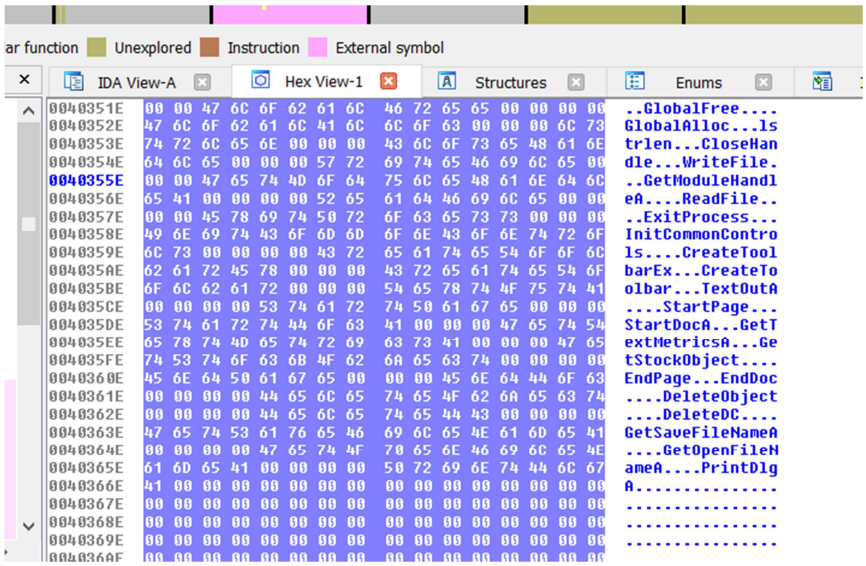


The content of the 0x1038 offset is 0x355E.

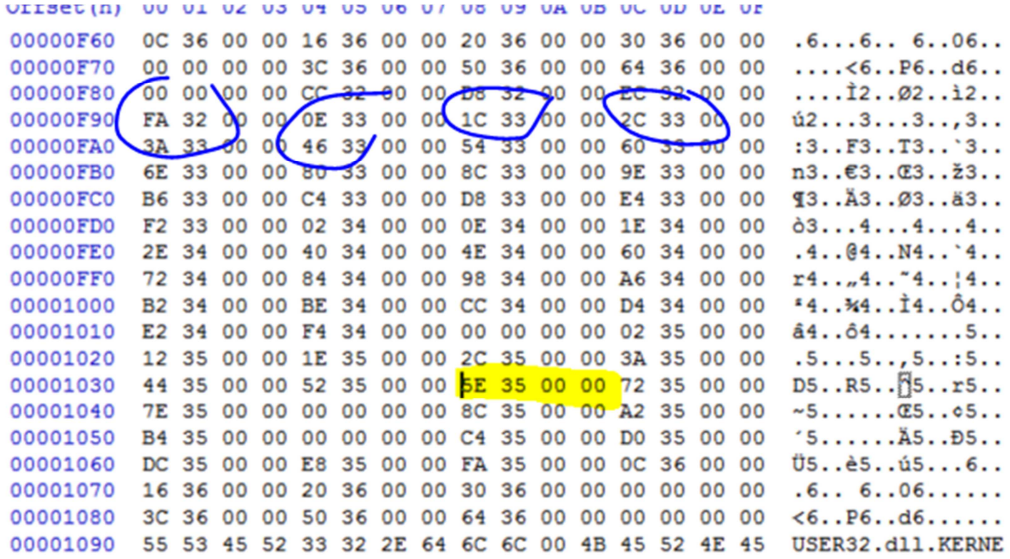
If I add 0x355E to the 0x400000 ImageBase, we have 0x40355E. What is in there? To find it out, we should load the original one with Manual Load to load all the executable sections.




Accept all the sections and when it runs, go to 0x40355E and on the right, we see the API name: GetModuleHandleA.




So, when the system runs, if we see the below image, it works on those little boxes.




And it add each other the ImageBase content and finds the correspondent function string. And from that string, it solves the address in our machine in runtime. As in this case, it finds the GetModuleHandleA address and modifies **5E 35** with the API address.

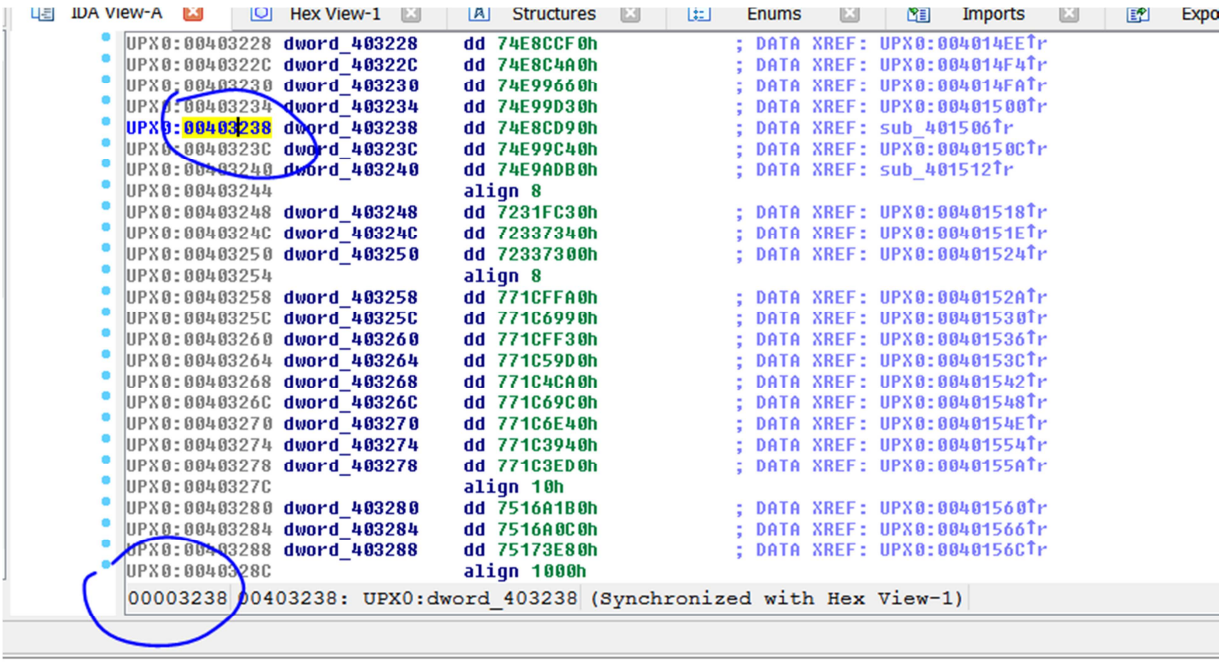
That’s why an executable works on any machine because it will always find the name of the correspondent API and solve it to run. So, the address that it always has will be valid independently of the machine. But the address of the GetModuleHandleA little box will be the same in all machines and executables without randomization. Just the content will change.

That’s why I use:

CALL [0x403238]

It will always work because 0x403238 is the IAT address for GetModuleHandleA, what changes is the content that the system will save modifying the 5E 35 original value that pointed to the string if we add it the base.

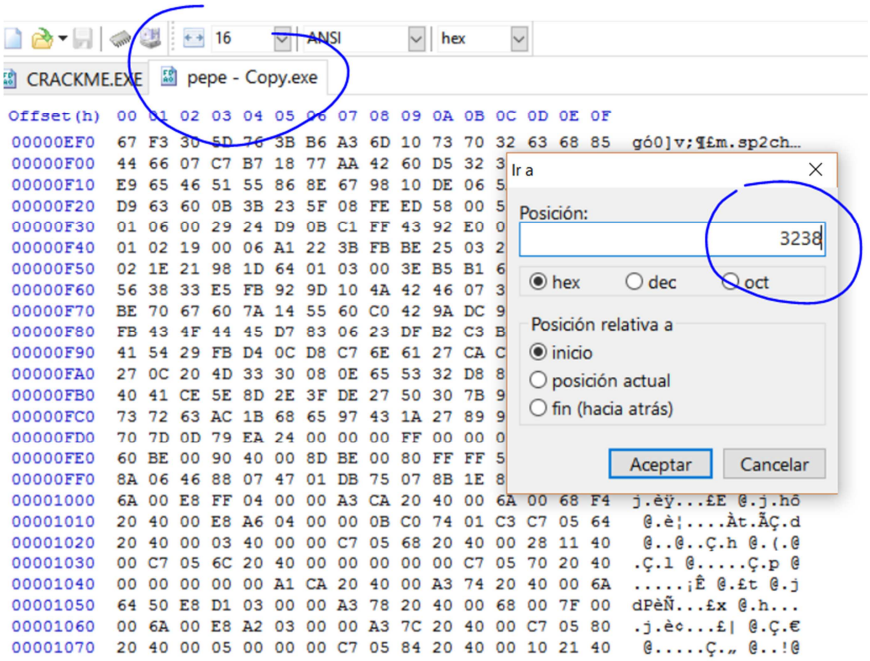
Without closing the two IDAs and with the packed and original files at the OEP, in a third IDA, load the Dumped we have just done: **pepe - Copy.exe** and in the dump, go to 0x403238 in this.



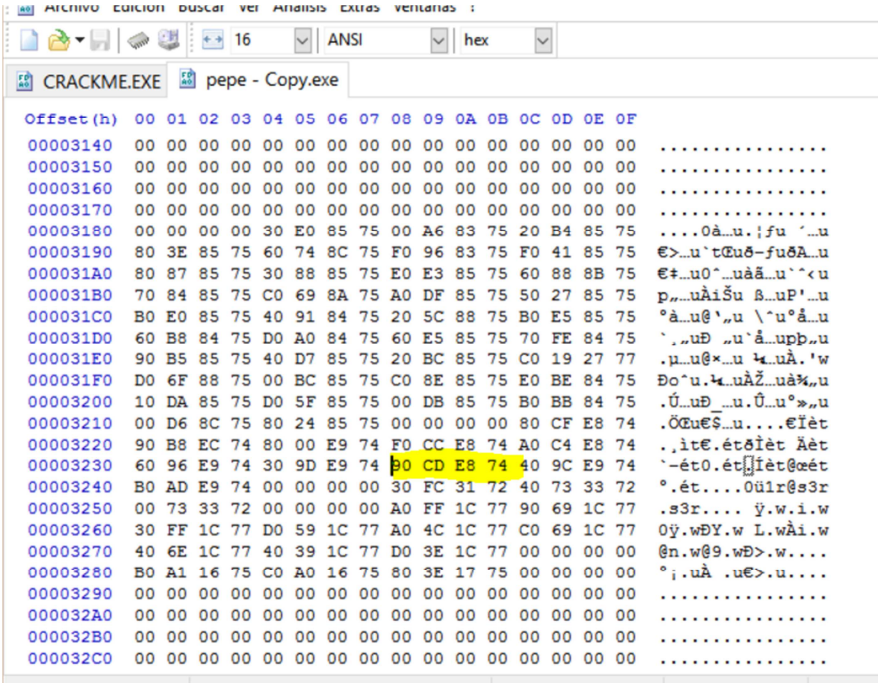

been propagated

Here, file offset is 0x3238 and it doesn't match the original because of the DumpFixer that made RawSize and VirtualSize the same changing the offsets and the GMHA API address.

Let's open it in HXD and go to 0x3238.

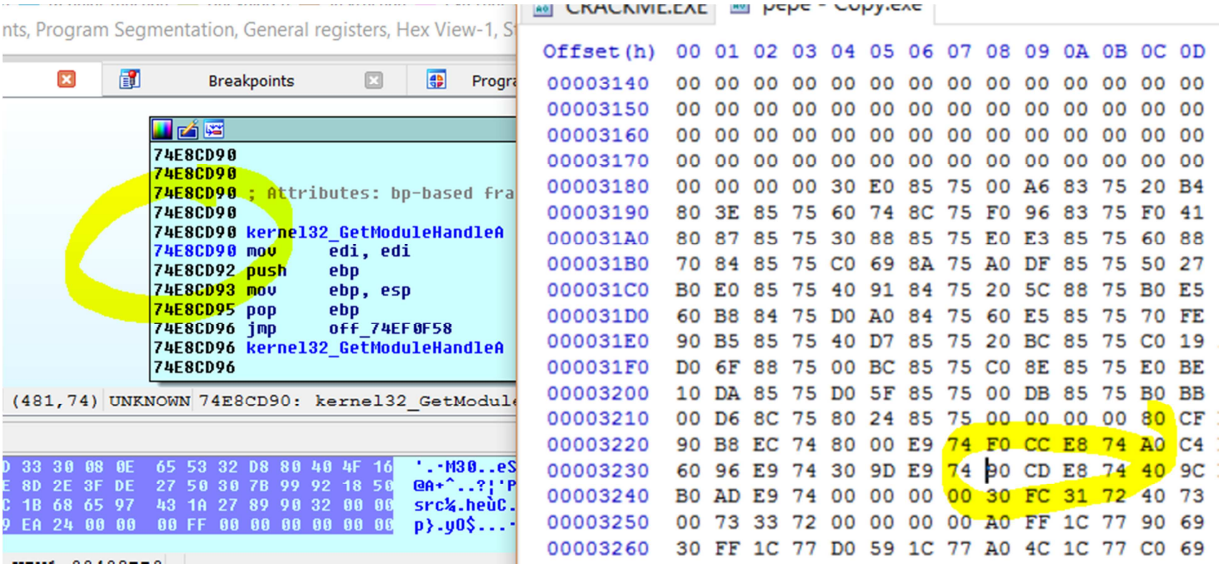



There, we see.




We have an API address there and not an offset to add it the ImageBase to find the API name.

When the system ran it solved the API address and saved it there and when dumping, the packed file GetModuleHandleA API address stays there.



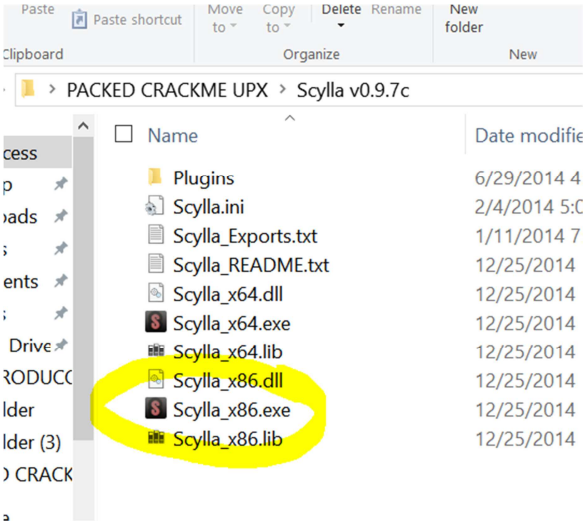

And what's the problem?

When the system runs, it looks for the value in the API address to add it to the ImageBase and find a string to solve it, but it is broken because when dumping, the final address was saved and the program will crash not being able to fill the IAT correctly.

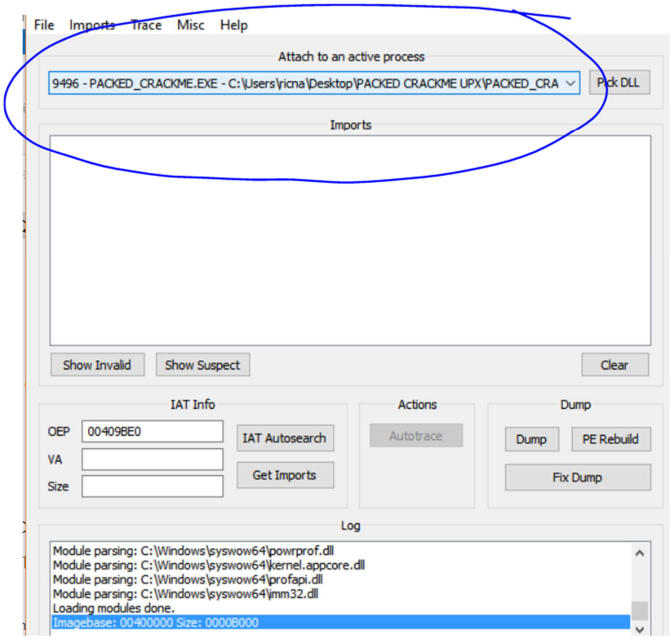
The idea is to fix the IAT and restore all those offsets, which pointed to the strings, with the API names. To do this, we need a tool called Scylla.

https://tuts4yoututs4you.com/download.php?view.3503

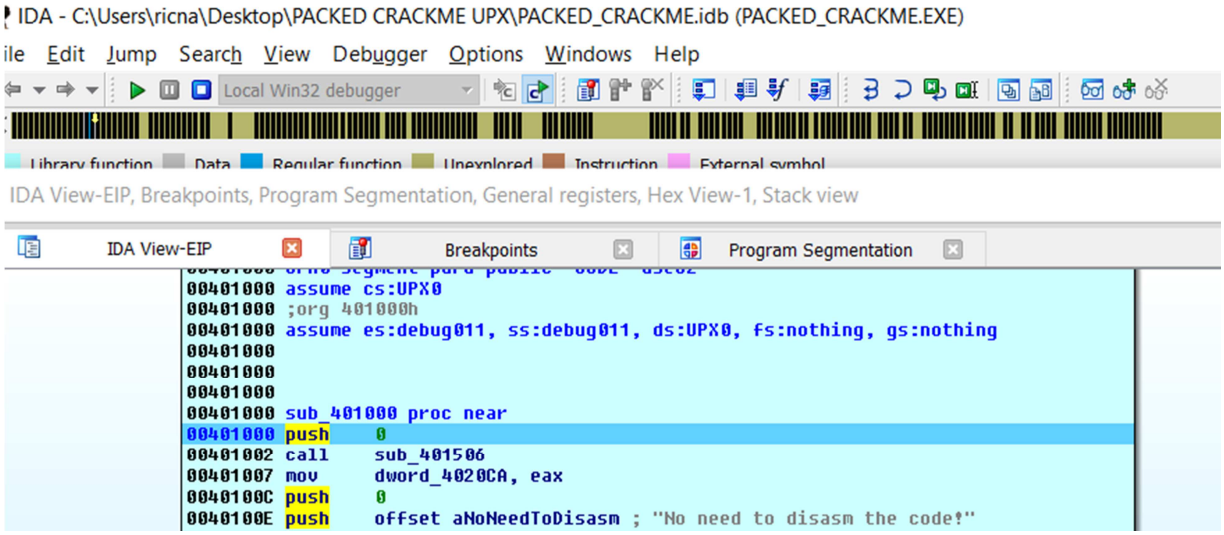
rar password = **tuts4you**



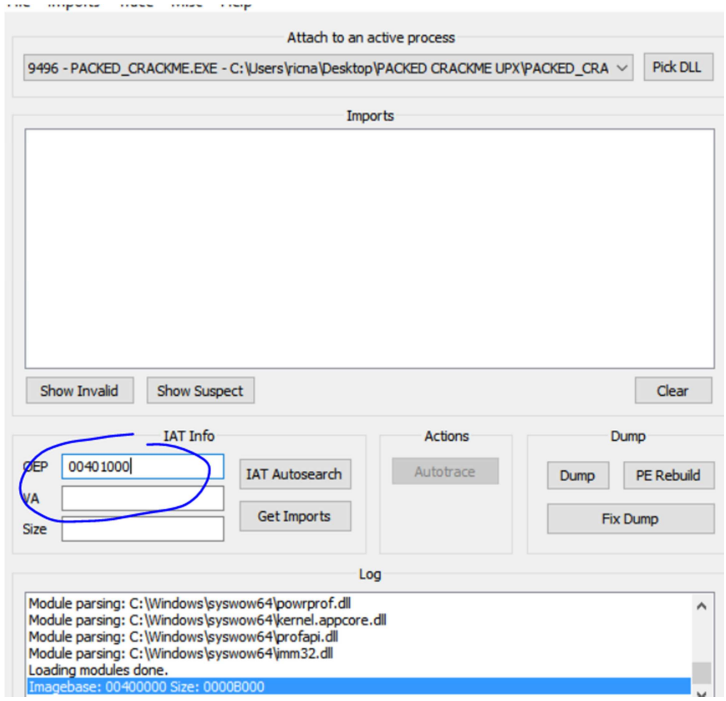

Run it.



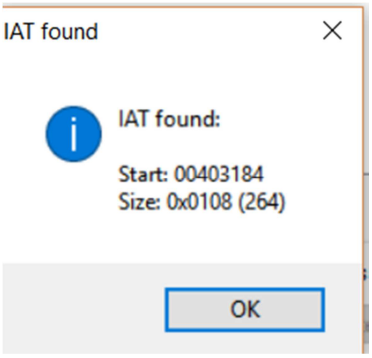
In ATTACH TO AN ACTIVE PROCESS, select the packed file process that is at the OEP in IDA yet.




Change the OEP to 401000.

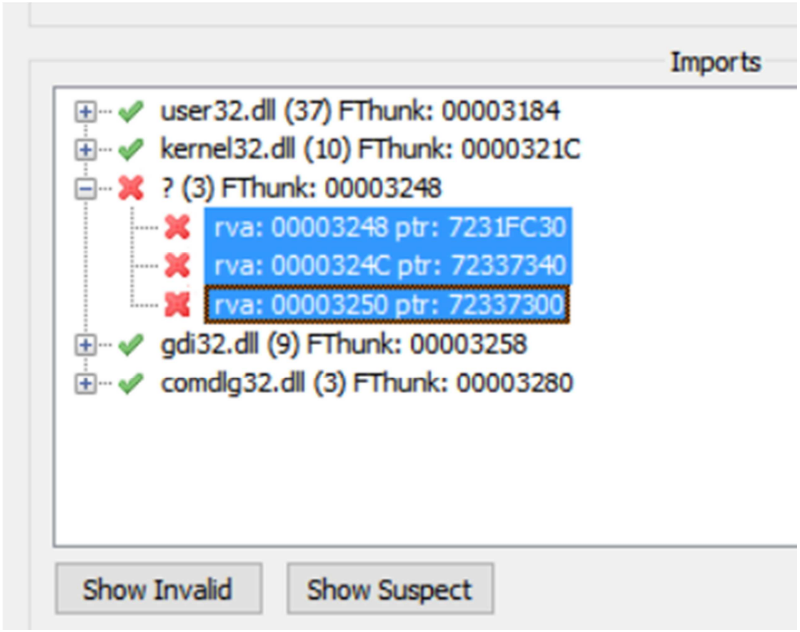



Press IAT AUTOSEARCH.

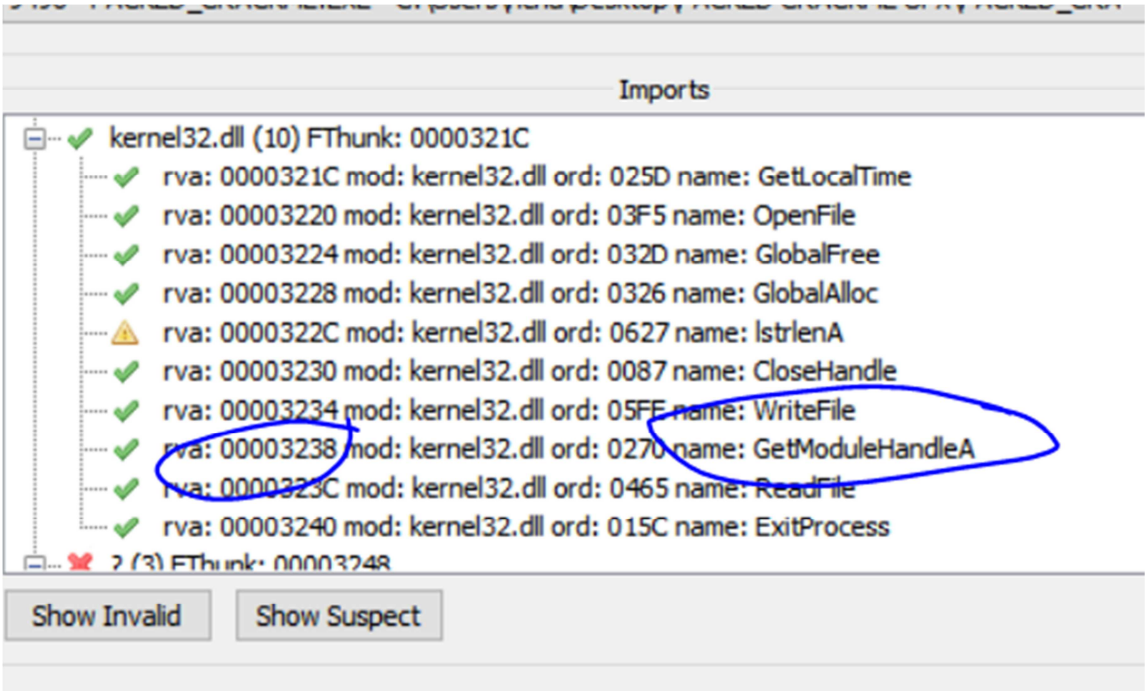



There, it says that the IAT starts at 00x403184 and its size is 0x108.

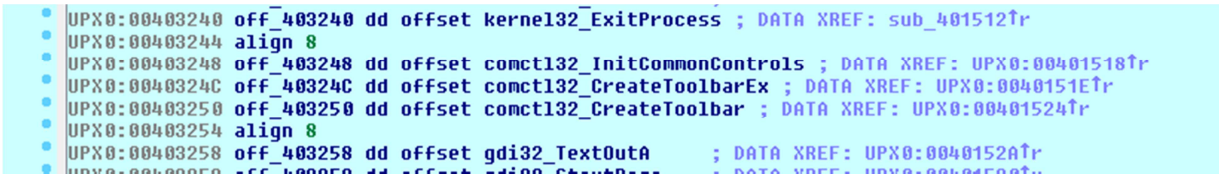
Let’s press GET IMPORTS.



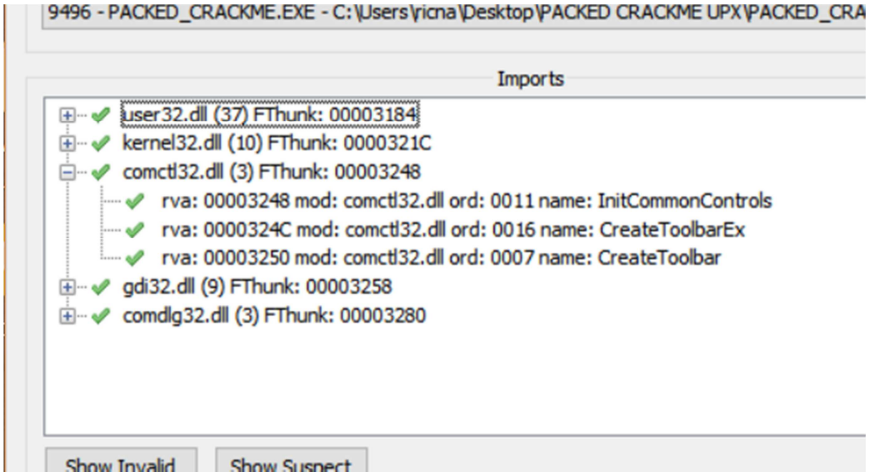

It solved everything except 3 APIs shown when pressing SHOW INVALID.



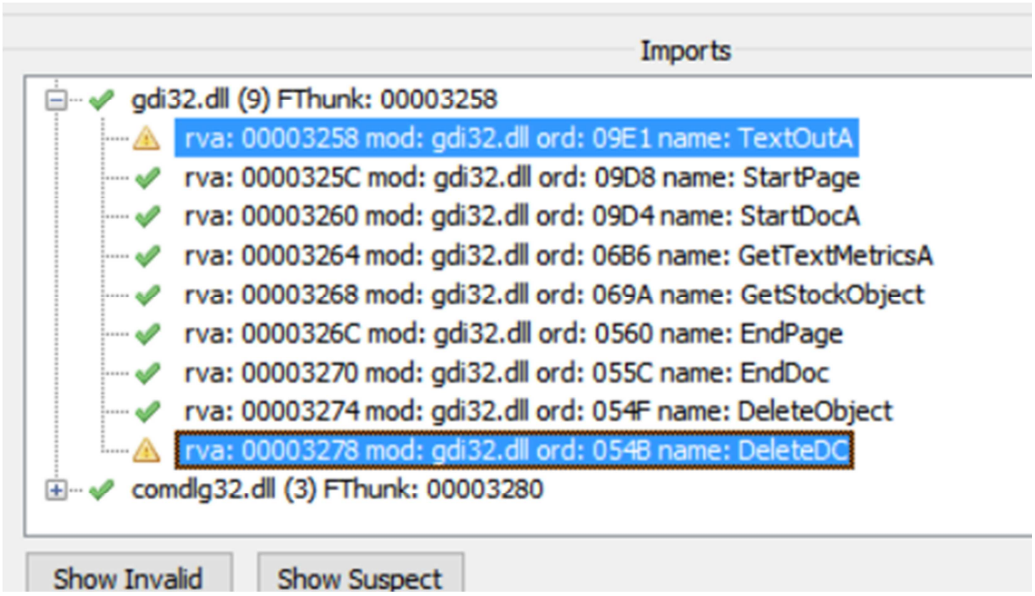

In the packed file, we see that the 3238 offset belonged to GetModuleHandleA and that is well solved. Let's see, in the packed file, the invalid API addresses from 0x403248 on to see what they are.



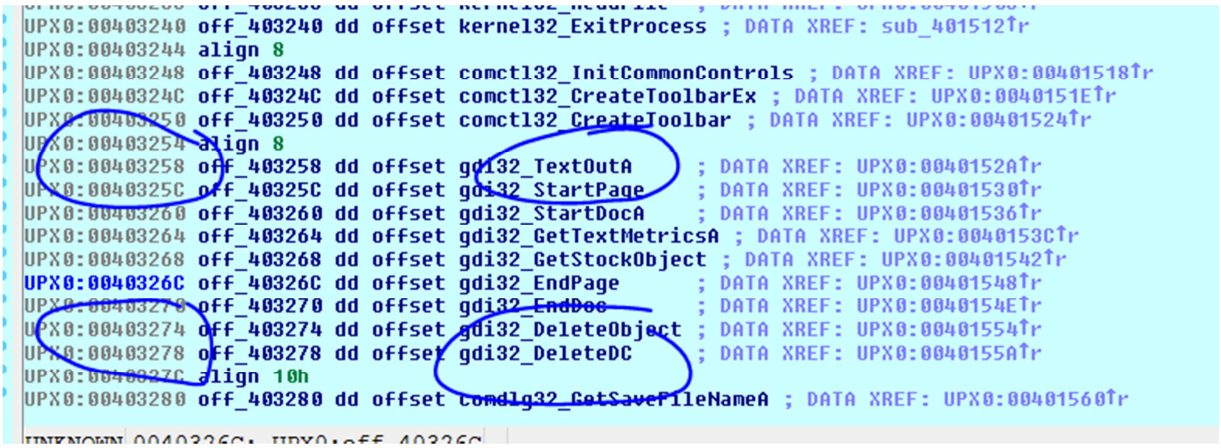

Those APIs belong to comctl32. If I right click on the invalid APIs in Scylla and select PLUGIN-PE COMPACT, it fixes them well.



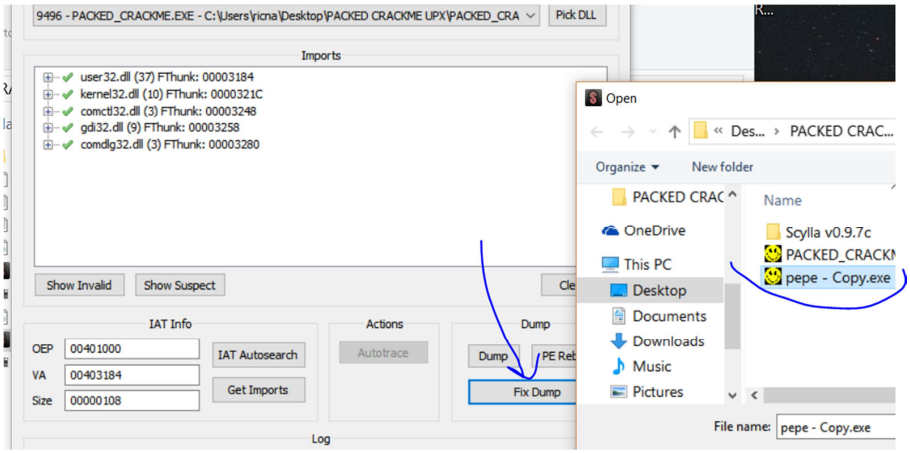

They are the API's we need.



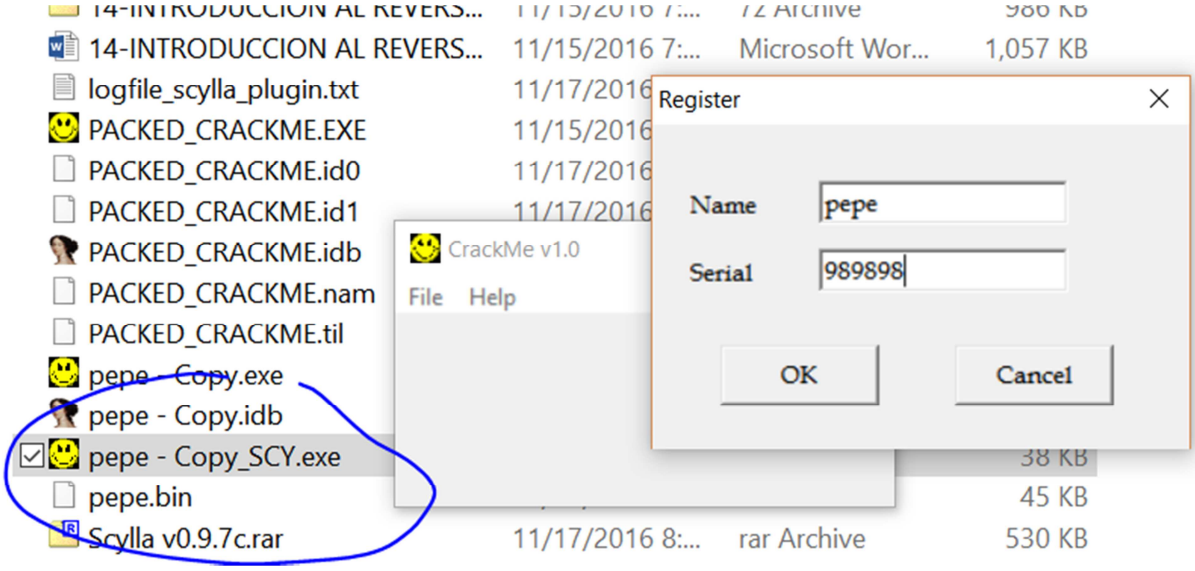

If we press SHOW SUSPECT, we see if those two suspects are correct going to 0x403258 and 0x403278.



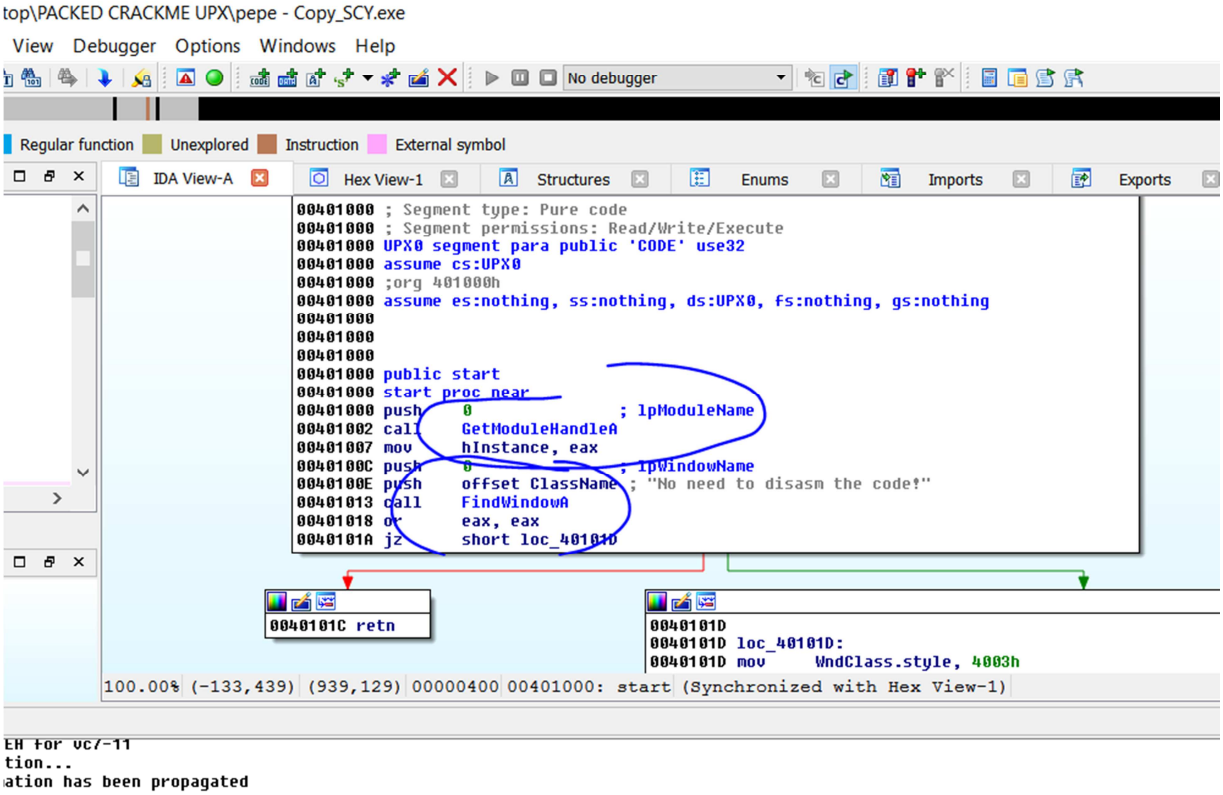

Yes, they are. Now, press FIX DUMP.

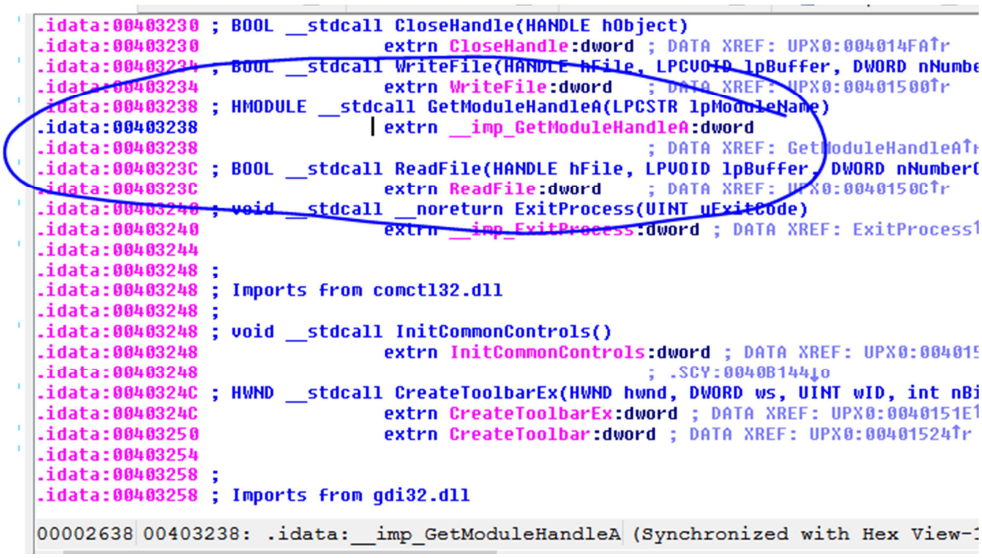


It saves it as pepe-Copy_SCY.exe and it works.




When loading the unpacked file in IDA, we see that it runs from the 0x401000 OEP and the API names look like the original file and they are well fixed.






The IAT part looks original too.

We have just unpacked our first and easy packer. We'll see more complex packers later.

**Ricardo Narvaja**

**Translated by: @IvinsonCLS**