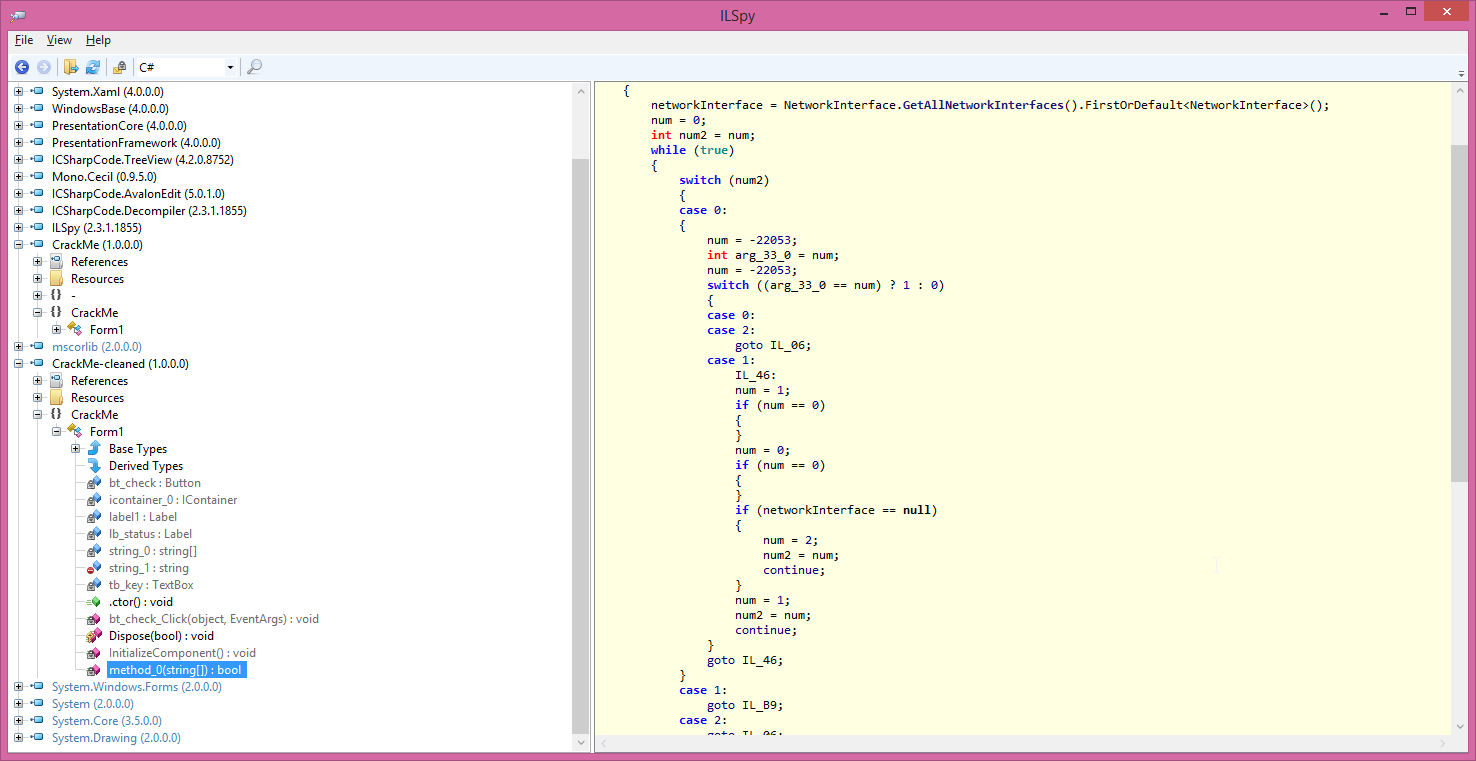
The task was done together with Maksim Kharaneka.

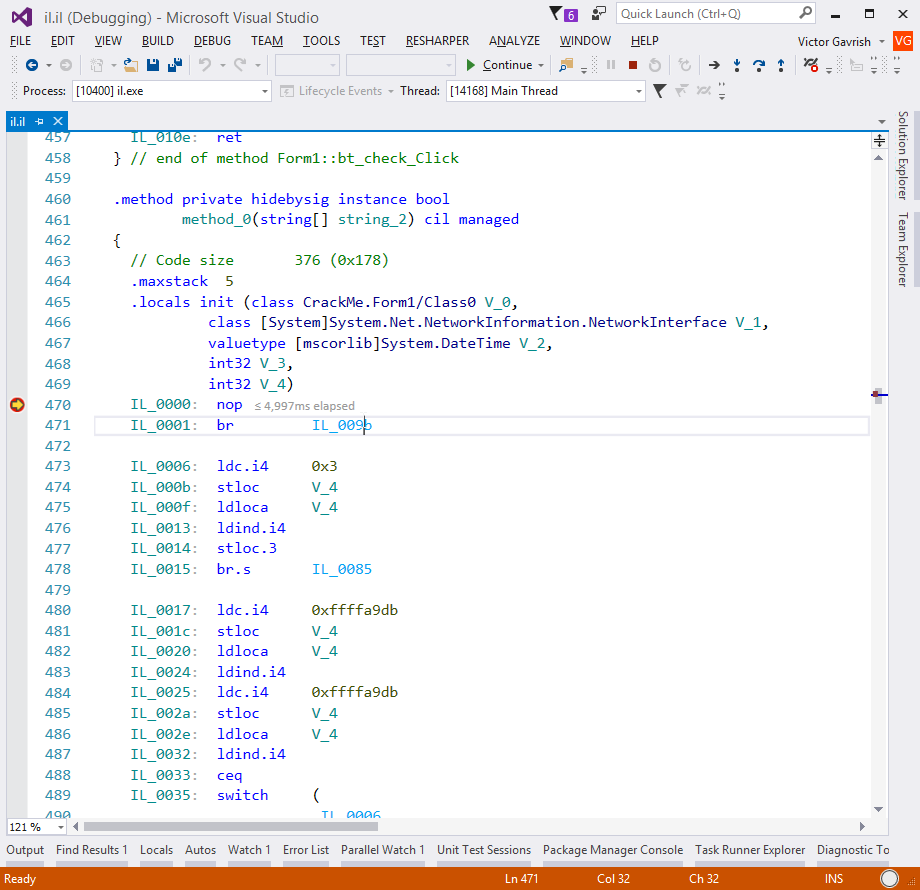
For a start, we used ILSpy in order to get the C# code. But it was obfuscated, so we used de4dot to deobfuscate it. We got some more or less legible IL code from this, though there were still unnecessary objects and loops and gotostatements.



Then we discovered that compiler-generated code and methods didn’t get disassembled, so we turned on that option in the decompiler.

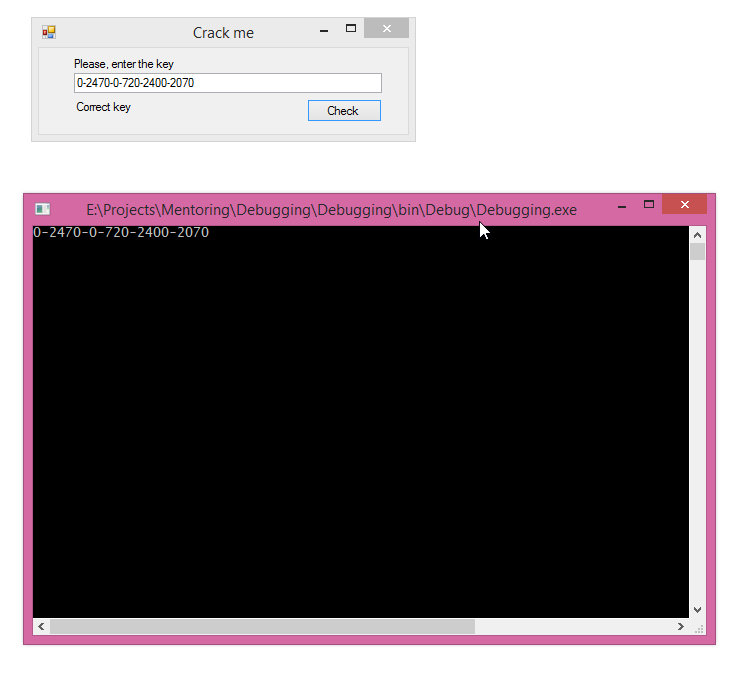
Then we used ildasm to decompile the exe into il code, and then compiled it again with ilasm /DEBUG in order to get a .pdb and be able to debug the IL code.

This allowed us to find the place where the actual check was being made.

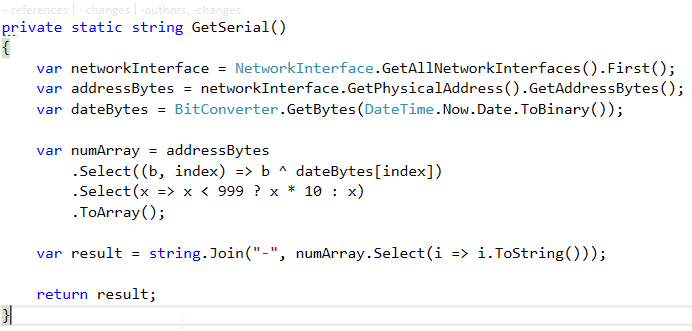


We started to look very carefully at what this code did in the decompiled sources. After deleting all the garbage forks, all the garbage objects and all the garbage goto statements, we were able to get code that generates an array of numbers.

After looking at it this way and that, we finally understood that all the rest of the code did was compare this generated number with the number the user entered. So all we needed to do is run this code on the local machine, and we’d find what “serial number” the code would respond to. Success!



After cleaning up the code some more, this is all it does in the end:



If this was real code, I imagine that a real hash algorithm would have been used (a cryptographically strong one), and not a simple XOR of two arrays, and obfuscation would not be relied upon as the only security measure. As we have learned, it is not all that difficult to understand what obfuscated code does, with a little effort.