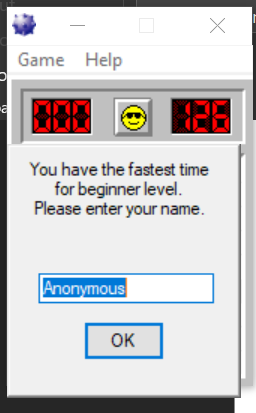
Lab Session 0x05

*You are given, winmine.exe. Crack it statically as follows:*

Rules of the game (because the last time I played this game was 2008 and back then I didn’t understand an ounce of English, so I was just clicking at random):

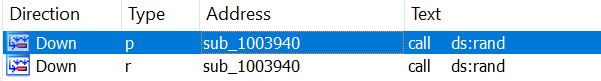
* Windows Minesweeper always makes the first click safe. You open squares with the left mouse button and put flags on mines with the right mouse button. Pressing the right mouse button again changes your flag into a questionmark.
* If you flag all of the mines touching a number, chording on the number opens the remaining squares. Chording is when you press both mouse buttons at the same time.
* The three difficulty levels are Beginner (9x9 with 10 mines), Intermediate (16x16 with 40 mines) and Expert (30 columns x16 lines with 99 mines). The game ends when all safe squares have been opened. A counter shows the number of mines without flags, and a clock shows your time in seconds. Minesweeper saves your best time for each difficulty level.

Pentru prima dată în 20 de ani, am reușit să bat jocul ăsta 😱😱😱 (trebuie să păstrez amintirea pentru posteritate – Delete pentru temă):

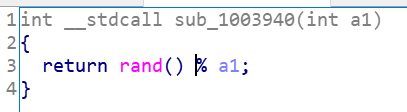


* *when you open winmine.exe the flags should already be present where the bombs are; (3p)*

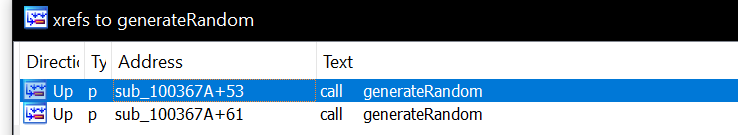
So, the map is definately a matrix. And, considering the bombs are always changing, we must assume that the algorithm makes use of the *rand()* function. After we open IDA (**32-bit!!!**), we click on *Imports* (to see all the functions that are being used by the program), double-click on *rand()*, which will move us to the *.idata* section, click on the name, so that is highlighted and press *X*, in order to see the references:



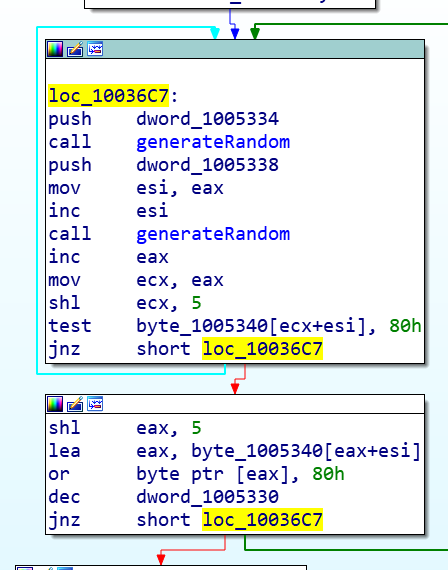
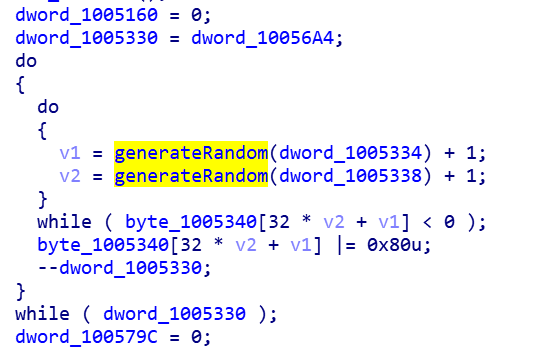
We can notice it is used inside only one function, ***sub\_1003940()***, that looks just like this:

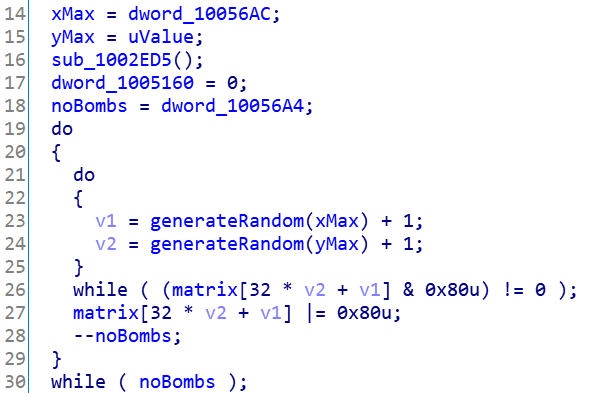


I renamed the function (easier to find it) – ***generateRandom()***. Our purpose is trying to find where the map is being created and/or initialized. So, let’s see the references to this function:



It is inside the function ***sub\_100367A()*** (rename: ***placeBombs()***) (***loc\_10036C7*** is inside it) and only here, so it is safe to assume that here it’s where the mine placement is happening.

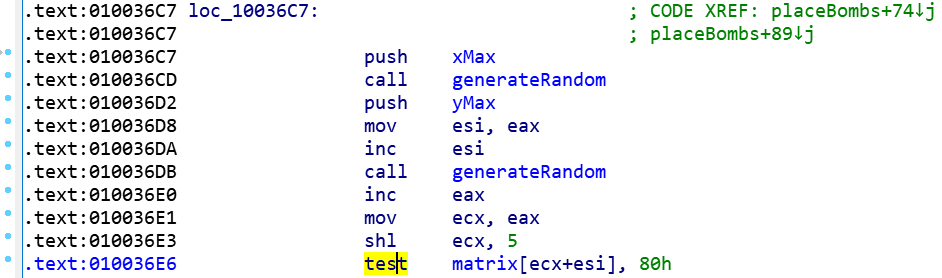
After renaming and retyping some values we get this:

As we can notice, now it is far easier to understand what is happening here. Depending on the level of difficulty, ***xMax*** (*line*) & ***yMax*** (*column*) are set, and a random value is generated, a value that cannot be greater than ***xMax*** / ***yMax***. The inner loop generates random numbers until the cell inside the matrix (*v1* and *v2* are coordinates inside the matrix) is not set (aka, does not have a bomb). If the algorithm reaches a cell without a bomb, it sets the 8-th bit from right to left and decreases the number of bombs remaining. So, we found our minefield; double click on ***matrix*** and we find the address: ***01005340***.

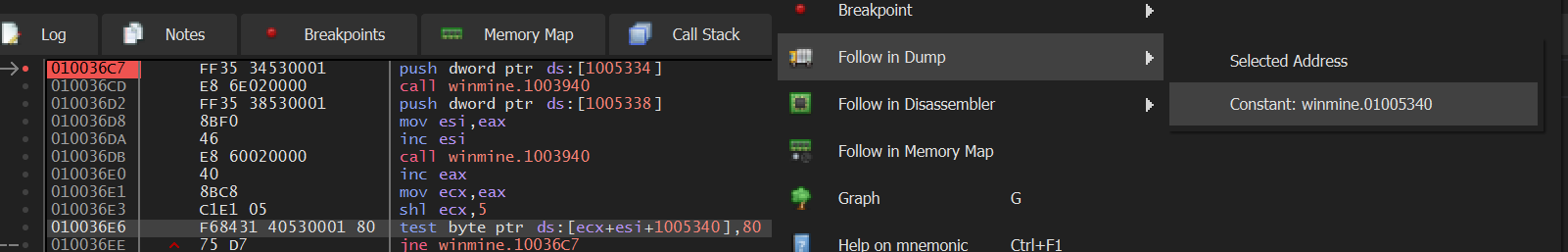


**Why 32 thou?** The maximum number of columns is 30 + 2 columns that delimit the actual matrix (Ex: beginner mode → column 0 is delimitation, columns 1-9 the actual matrix, column 10 is delimitation, column 11-32 empty because the matrix is 9x9). We will see in an example below.

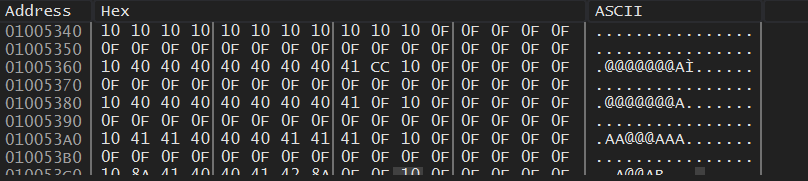
Load the program in x32dbg, then press F9. This will take you to the main function usually. We saw from above that the loop starts at the address ***10036C7*** and we place a breakpoint here.



At the address ***10036E6*** is where we check if the 8th LSB is set or not. ***Right click*** at that address in debugger, ***Follow in Dump, Constant: winmine.01005340***.

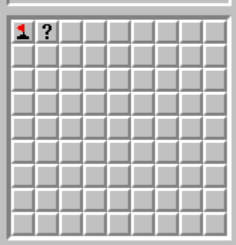
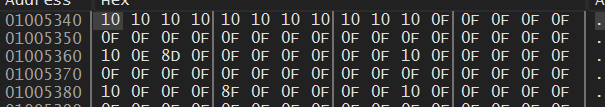


Let’s play a little in order to notice how the matrix behaves:

So: ***border - 10***, ***empty cell – 40***, ***1 (cell) – 41***, ***clicked bomb – CC***, ***unopened cell / cells beyond the border - 0F***, ***unopened bomb (after finish) – 8A***, ***2 (cell) – 42***.

Let’s see the symbols for flags and question mark:

The second cell has a bomb, that is why the hex is 8D, otherwise the hex would’ve been 0D. So: ***flag – 0E***, ***question mark – 0D***, ***bomb (in game) – 8F***.

Now, all we have to do is, instead of setting 8F to a bomb, we need to set it as 8E:

* ***0F: 0b00001111***,
* ***8F: 0b10001111***,
* ***8E: 0b10001110***

The assembly instruction that sets a bomb is this:



We go in the debugger at the coresponding address:



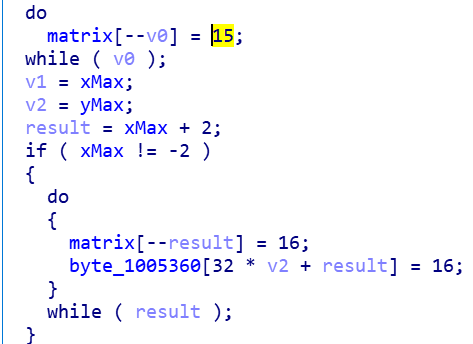
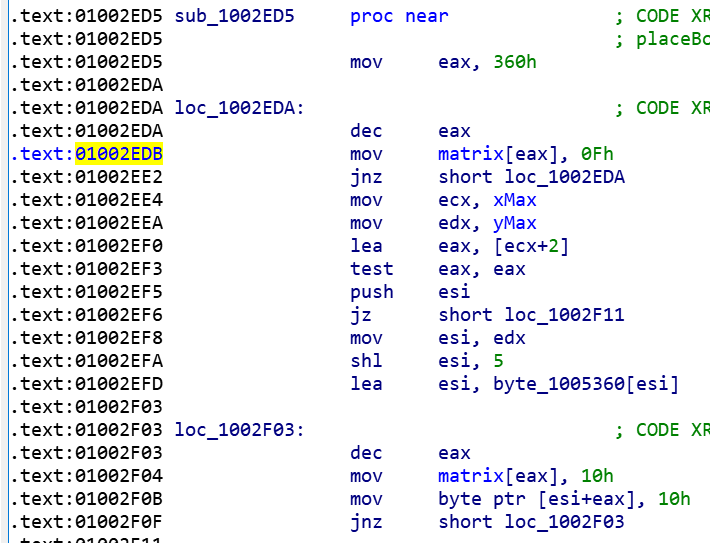
***Right click*** on the line, ***Binary***, ***Edit*** and we need to XOR with 81:



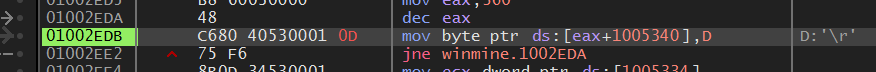
And now, we save the patched version: ***Right click → Patches → Patch File → winmine\_crack.exe***.

* *when you open winmine.exe put the question mark on positions that are blank; (3p)*

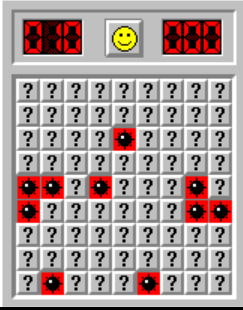
We know that that ***? → 0D: 0b00001101*** and ***empty cell → 0F: 0b00001111***. We xref the matrix and in ***sub\_1002ED5()*** we can notice here is the place where the matrix is being initialized.

We look in the debugger, at the address ***0x01002EDB*** and replace ***0F*** with ***0D***:

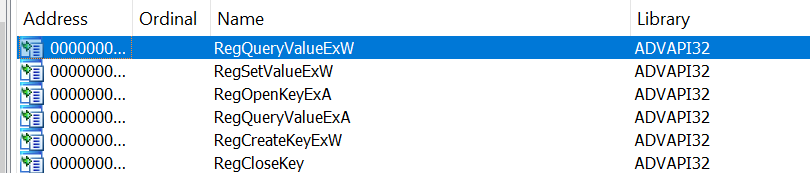


And in ***winmine\_Fix.exe*** we have the final version of the modified game:

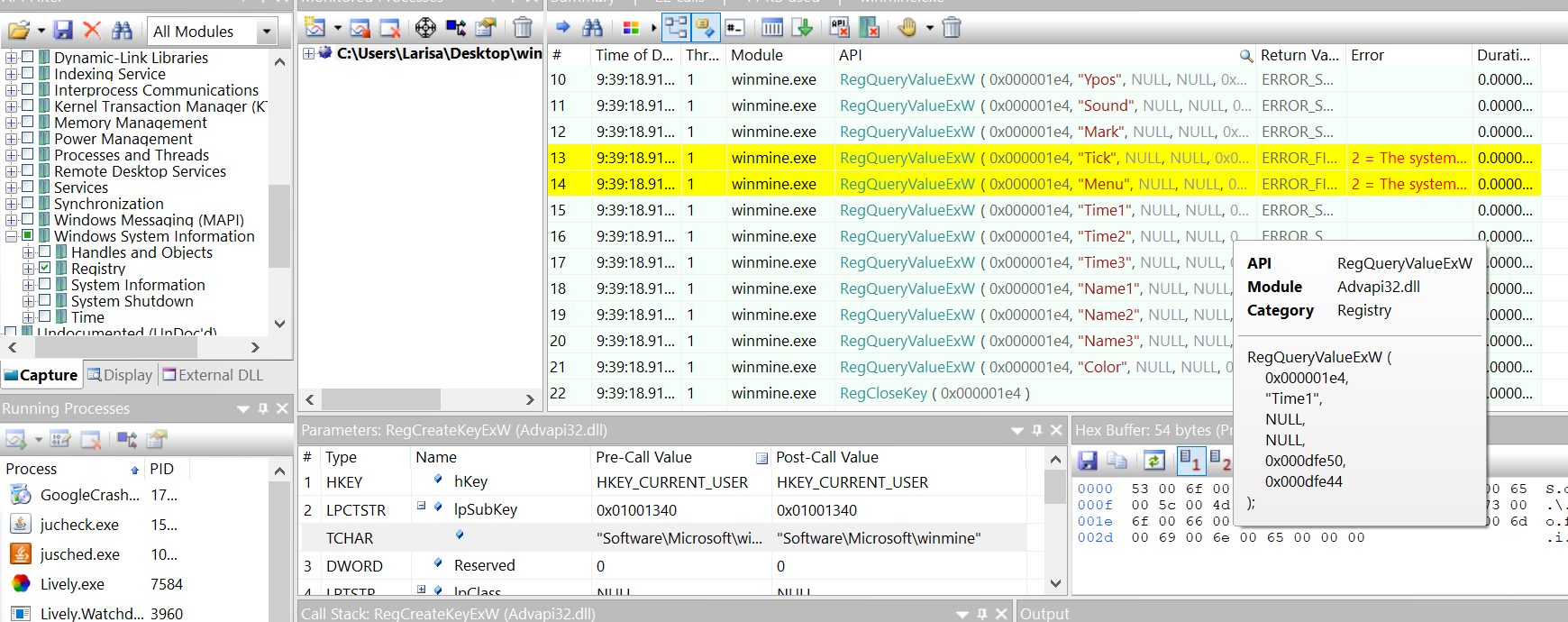
 

* *edit the “Fastest Mine Sweepers” to show your name for all levels of difficulty and set the number of seconds to the minimum; (2p)*

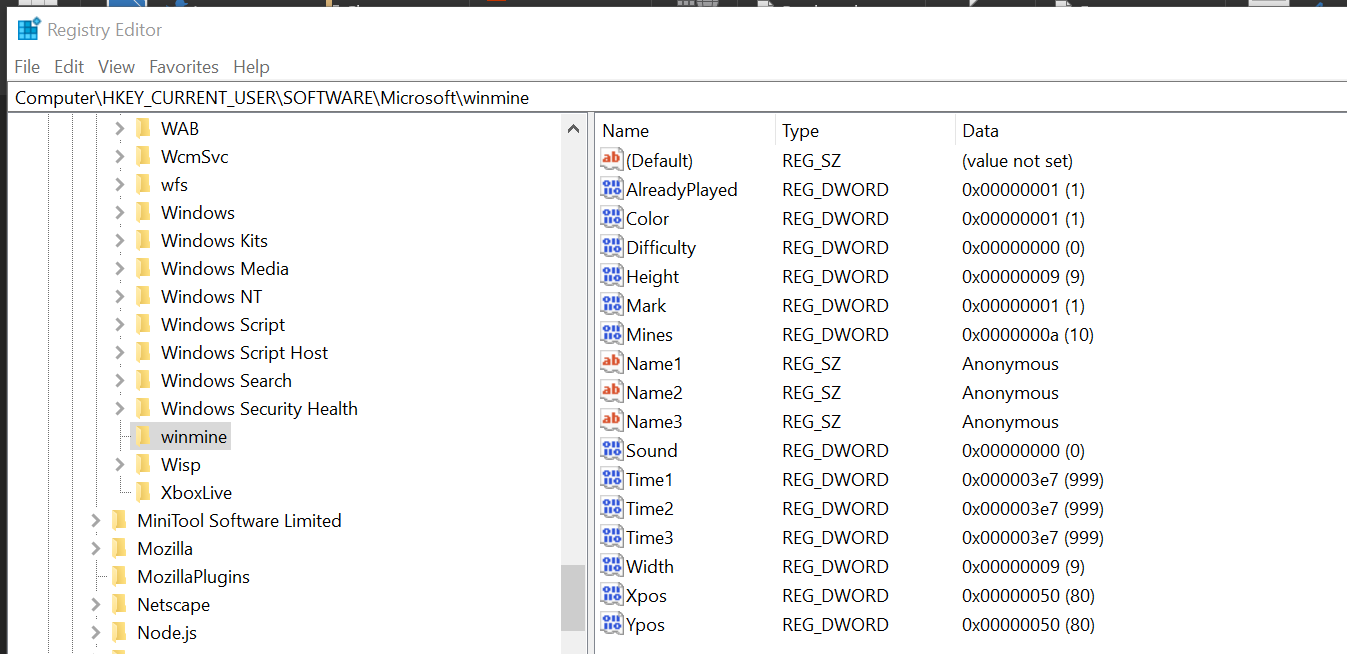
If we look at the ***Imports*** tab in IDA, we can notice ***ADVAPI32*** library, that provides Windows API functions related to Windows registry, services, applications and user accounts ([source](https://www.file.net/process/advapi32.dll.html)). It makes sense considering that highscores and times are saved locally, for each user individually (and also, in the *About Minesweeper* window, we can see that the PC name is being used).



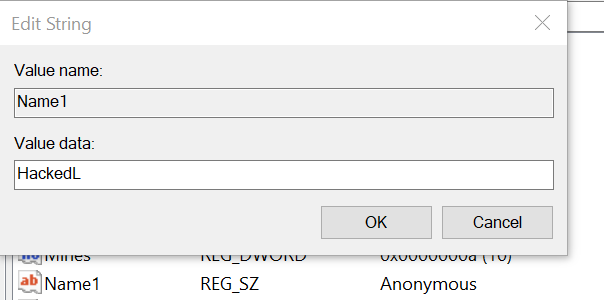
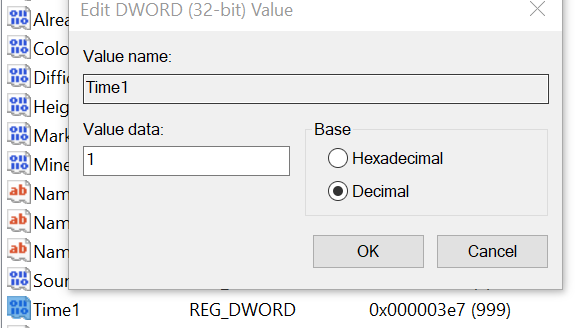
So, we use **API Monitor** and check ***System Services/Windows System Information/Registry*** from the left panel, in order to see the registry keys that are being accessed:



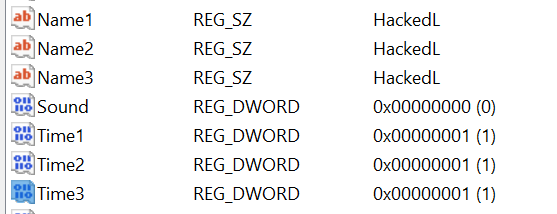
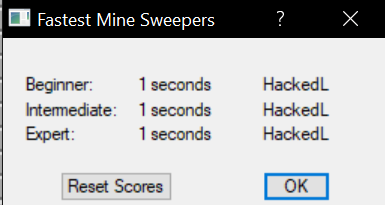
Our deduction is correct (we see 3 names and 3 times for winning the 3 levels of difficulty). Let’s open the folder and change the data (**Windows Key + R** → **regedit** → ***HKEY\_CURRENT\_USER \Software\Microsoft\winmine***):



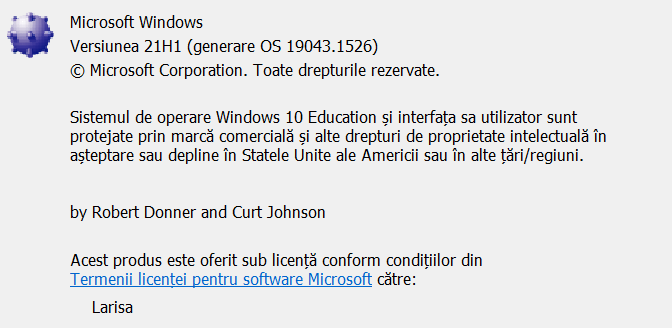
So, we will set the name to my own and the time to one second:

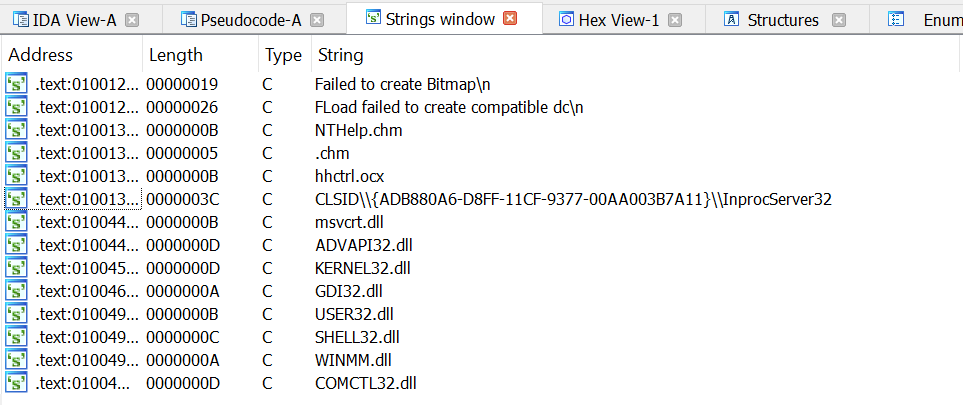
We get:

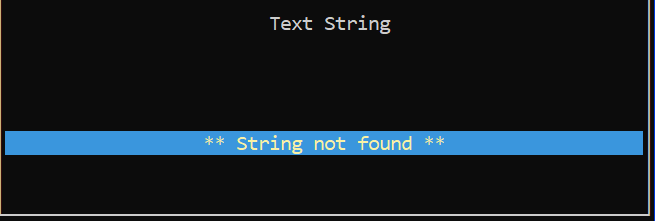
* *edit the “About MineSweeper” window to show your name as the creator of the game. (2p)*



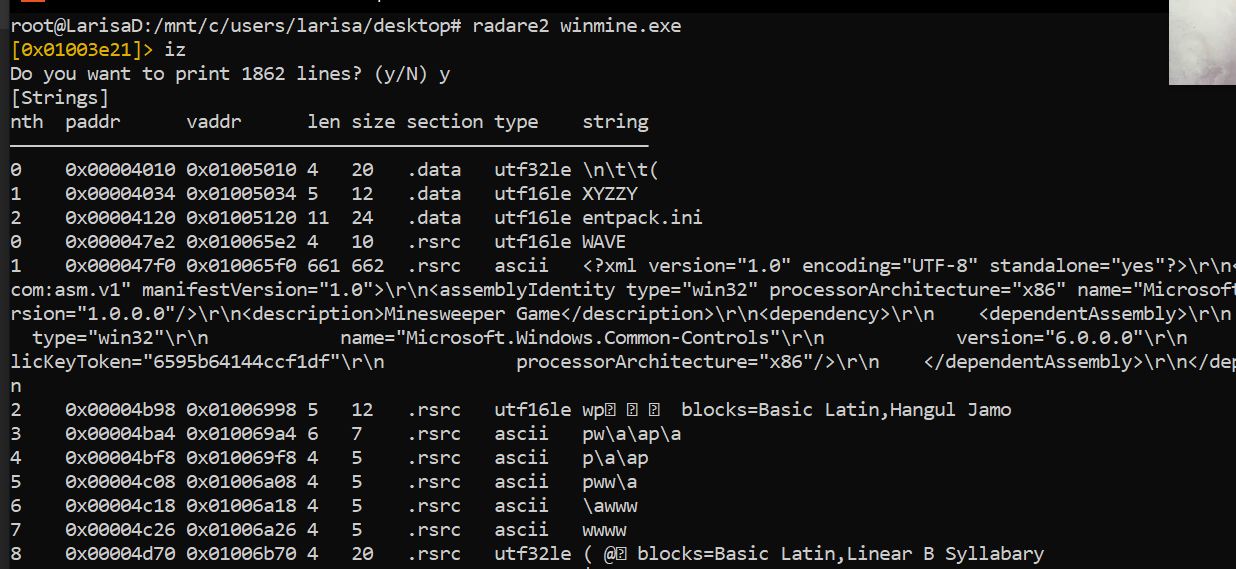
Ok, let’s try to look at the strings found in IDA (***SHIFT + F12***):

Not very usefull.

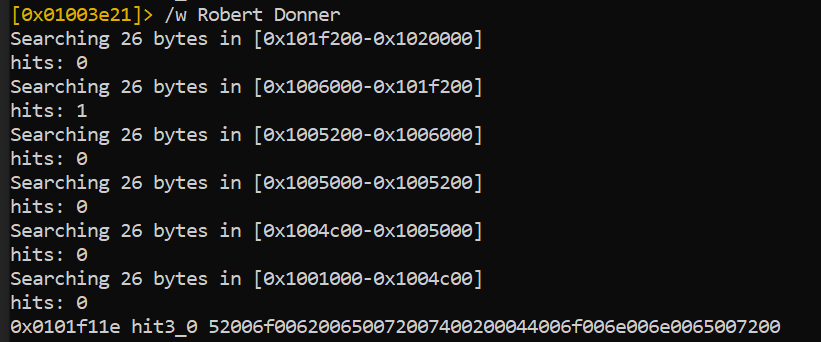
Let’s try with hexeditor (we will see below why it doesn’t work when we search for „Robert”, for example):



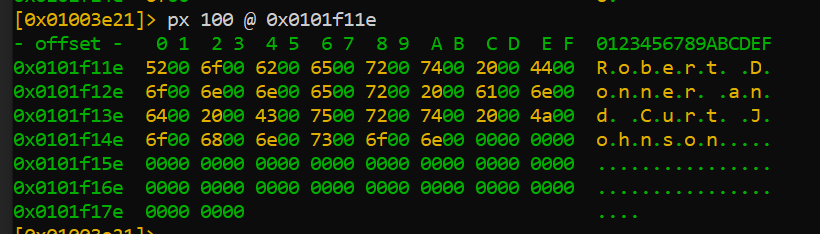
Let’s try with ***radare2***: ***radare2 winmine.exe; iz;*** (iz – find strings)



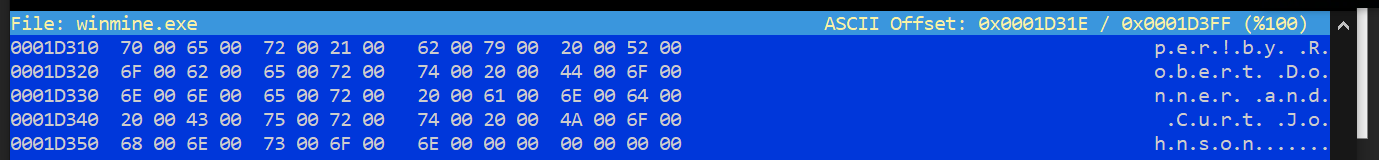
We can notice that we have multiple strings, a lot more than IDA found. Let’s search using the name of the authors: ***/w Robert Donner***

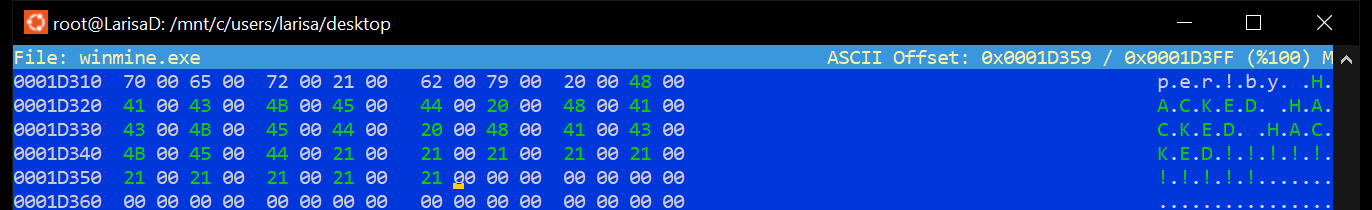


The first parameter is the start address and the last one is actually the hex value. Let’s print 100 bits starting at that address: ***px 100 @ 0x0101f11e***

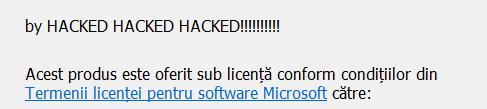


And this is why we couldn’t find it using hexeditor. We search the hex string using hexeditor and change the string:





Voila:



* *Make a script that finds dinamically a Minesweeper window and applies on it the patches from the laboratory. (6p)*

I wrote a .py file (***patch.py***) to do this. The game must be run first, then the patch. Inside the patch, we modify the registry keys and ensure that the bombs will get flags (we replace the original game with the modified file):

