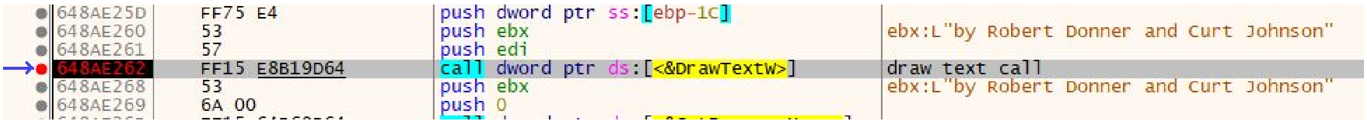
Lab 5 - Cracking Minesweeper

Edit "About Minesweeper" window

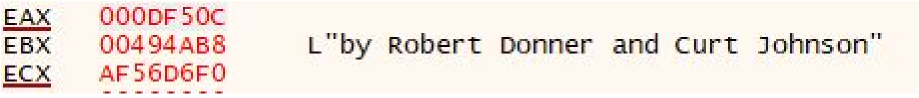
For this task, I will edit the "About Minesweeper" window to display the text `"by Stefan"` instead of `"by Robert Donner and Curt Johnson"`.

I started by debugging the executable in `x32dbg`. I tried to search references for the string to be replaced, but I couldn't find any. Thus, I started to look after api calls, and I found the function DrawTextW being used. I set a breakpoint at its address and observed that the `ebx` register holds the addresses of the strings displayed in the "About" window.



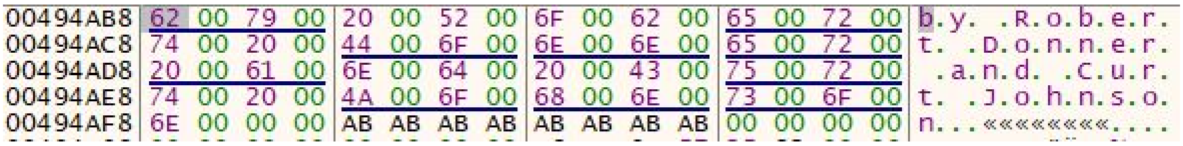
```
648AE25D   FF75 E4         push dword ptr ss:[ebp-1C]
648AE260   53              push ebx                       ebx:L"by Robert Donner and Curt Johnson"
648AE261   57              push edi
648AE262   FF15 E8B19D64   call dword ptr ds:[<&DrawTextW>]   draw text call
648AE268   53              push ebx                       ebx:L"by Robert Donner and Curt Johnson"
648AE269   6A 00           push 0
```

Here is our string:



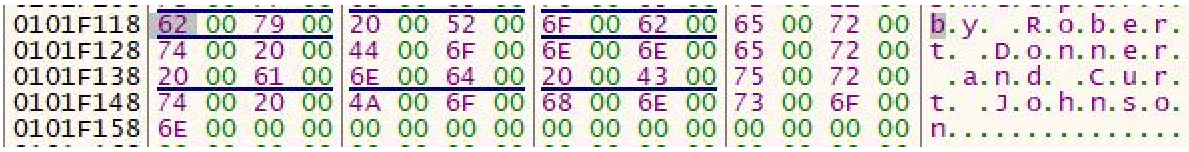
```
EAX   000DF50C
EBX   00494AB8     L"by Robert Donner and Curt Johnson"
ECX   AF56D6F0
```

I follwed in dump the address in `ebx` and found the string. But that's not the location where the string is stored, so I couldn't patch the program.



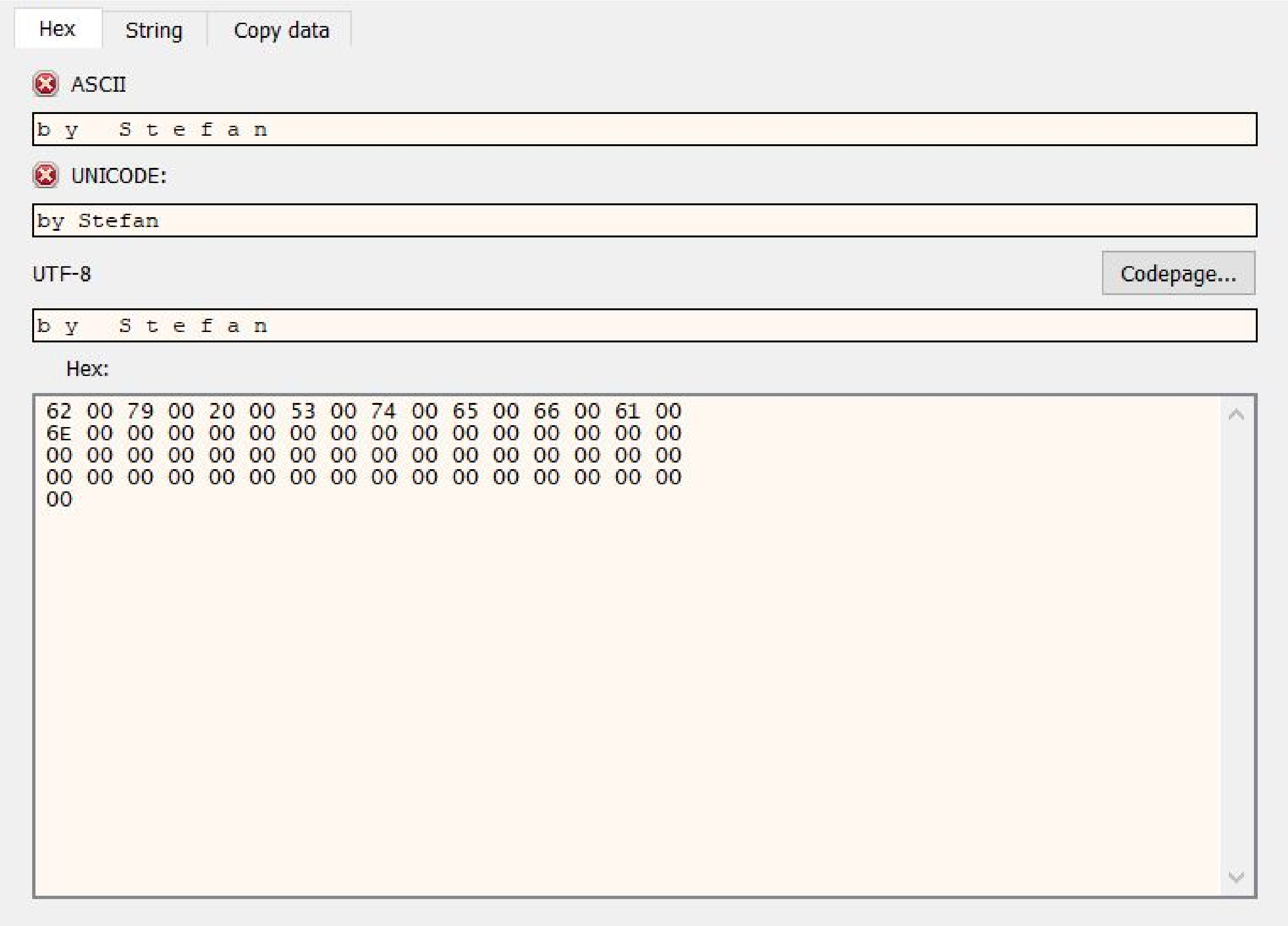
```
00494AB8  62 00 79 00 20 00 52 00 6F 00 62 00 65 00 72 00  b.y. .R.o.b.e.r.
00494AC8  74 00 20 00 44 00 6F 00 6E 00 6E 00 65 00 72 00  t. .D.o.n.n.e.r.
00494AD8  20 00 61 00 6E 00 64 00 20 00 43 00 75 00 72 00   .a.n.d. .C.u.r.
00494AE8  74 00 20 00 4A 00 6F 00 68 00 6E 00 73 00 6F 00  t. .J.o.h.n.s.o.
00494AF8  6E 00 00 00 AB AB AB AB AB AB AB AB 00 00 00 00  n...««««««««....
```

However, I observed that each letter is represented on two bytes, so I tried once again to search for the string reference, this time using the pattern. I was able to find our string stored at the address `0x0101F118`, along with other strings.



```
0101F118  62 00 79 00 20 00 52 00 6F 00 62 00 65 00 72 00  b.y. .R.o.b.e.r.
0101F128  74 00 20 00 44 00 6F 00 6E 00 6E 00 65 00 72 00  t. .D.o.n.n.e.r.
0101F138  20 00 61 00 6E 00 64 00 20 00 43 00 75 00 72 00   .a.n.d. .C.u.r.
0101F148  74 00 20 00 4A 00 6F 00 68 00 6E 00 73 00 6F 00  t. .J.o.h.n.s.o.
0101F158  6E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  n...............
```

Now, I simply edited the binary:



ASCII

b y   S t e f a n

UNICODE:

by Stefan

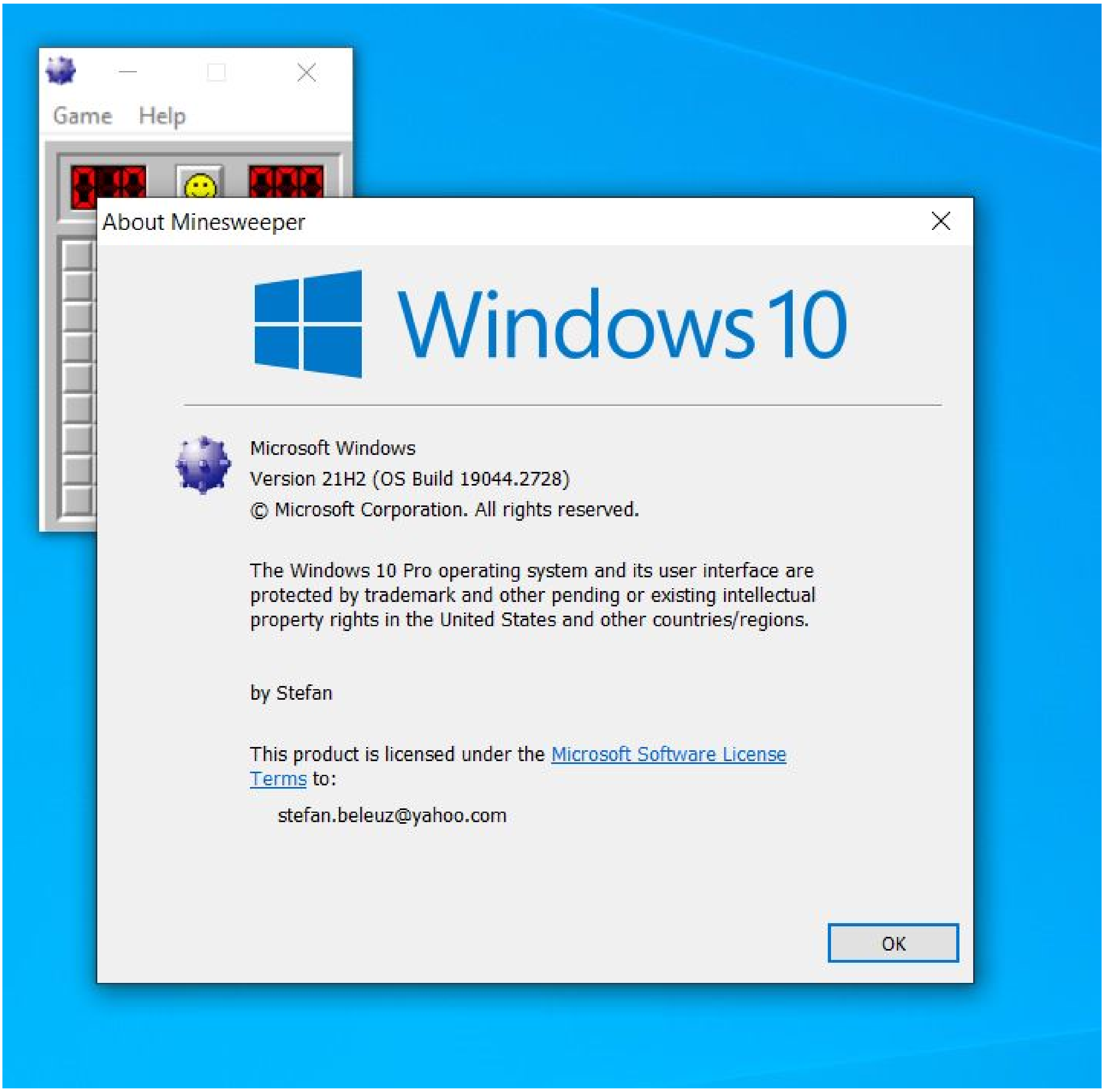
UTF-8

b y   S t e f a n

Hex:

```
62 00 79 00 20 00 53 00 74 00 65 00 66 00 61 00
6E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
```

Here is the program patched with my name as the creator of the game:



About Minesweeper

Windows 10

Microsoft Windows
Version 21H2 (OS Build 19044.2728)
© Microsoft Corporation. All rights reserved.

The Windows 10 Pro operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

by Stefan

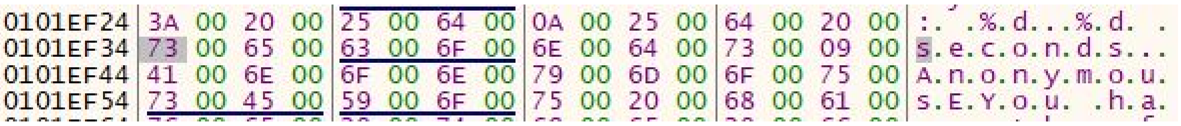
This product is licensed under the Microsoft Software License Terms to:

stefan.beleuz@yahoo.com

Edit "Fastest Mine Sweepers" window

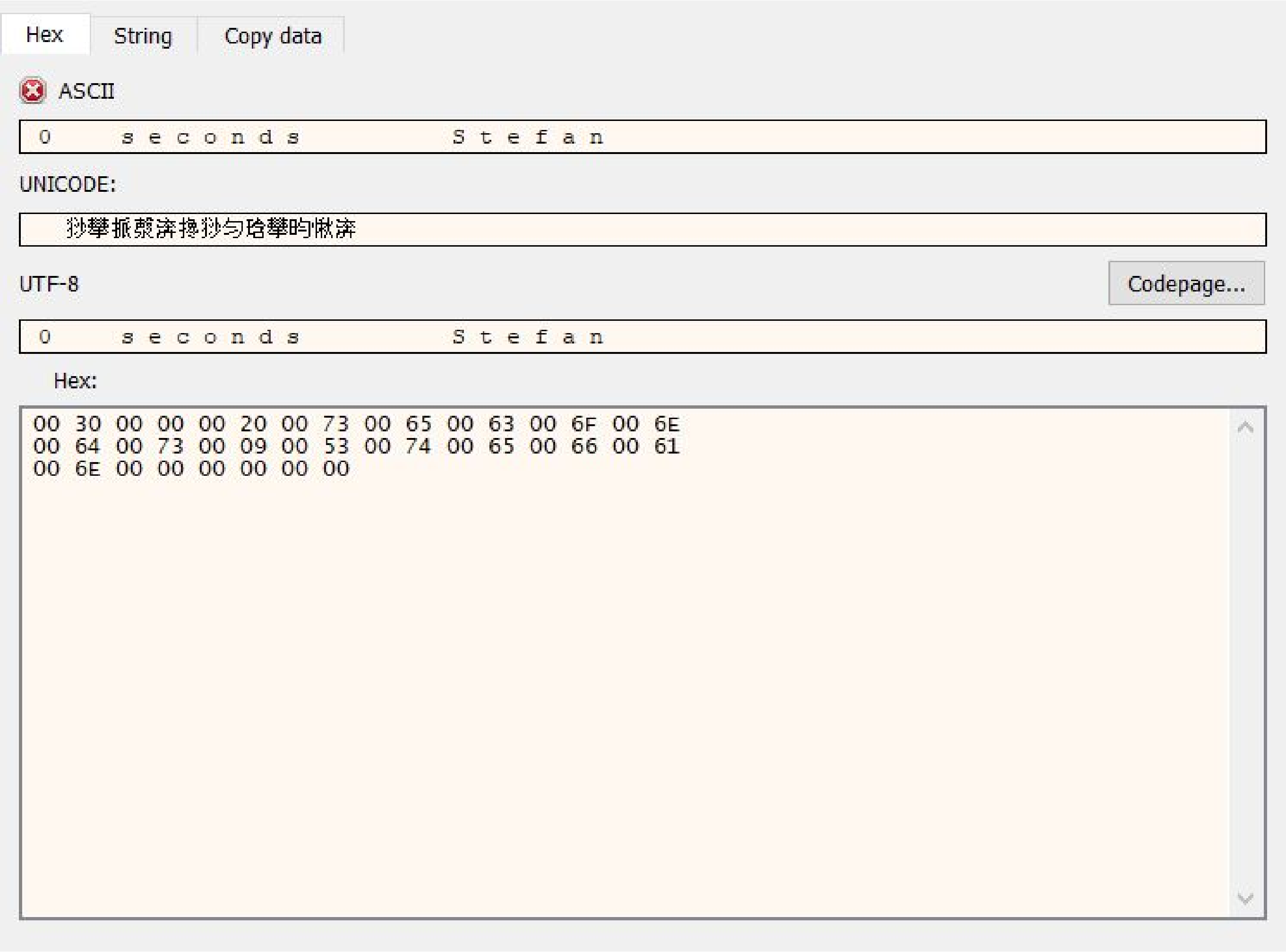
For this task, I will edit the "Fastest Mine Sweepers" window to show my name for all levels of difficulty and the number of seconds set to 0.

I applied a simmilar strategy to the previous task. I searched for the pattern `seconds`, again with each letter represented on two bytes, and I found it at address `0x0101EF34`, along with the string `Anonymous`.



```
0101EF24 3A 00 20 00 25 00 64 00 0A 00 25 00 64 00 20 00 :. .%.d...%.d. .
0101EF34 73 00 65 00 63 00 6F 00 6E 00 64 00 73 00 09 00 s.e.c.o.n.d.s...
0101EF44 41 00 6E 00 6F 00 6E 00 79 00 6D 00 6F 00 75 00 A.n.o.n.y.m.o.u.
0101EF54 73 00 45 00 59 00 6F 00 75 00 20 00 68 00 61 00 s.E.Y.o.u. .h.a.
```

I edited the binary, changing the `%d` pattern to `0` and the `Anonymous` string to my name:



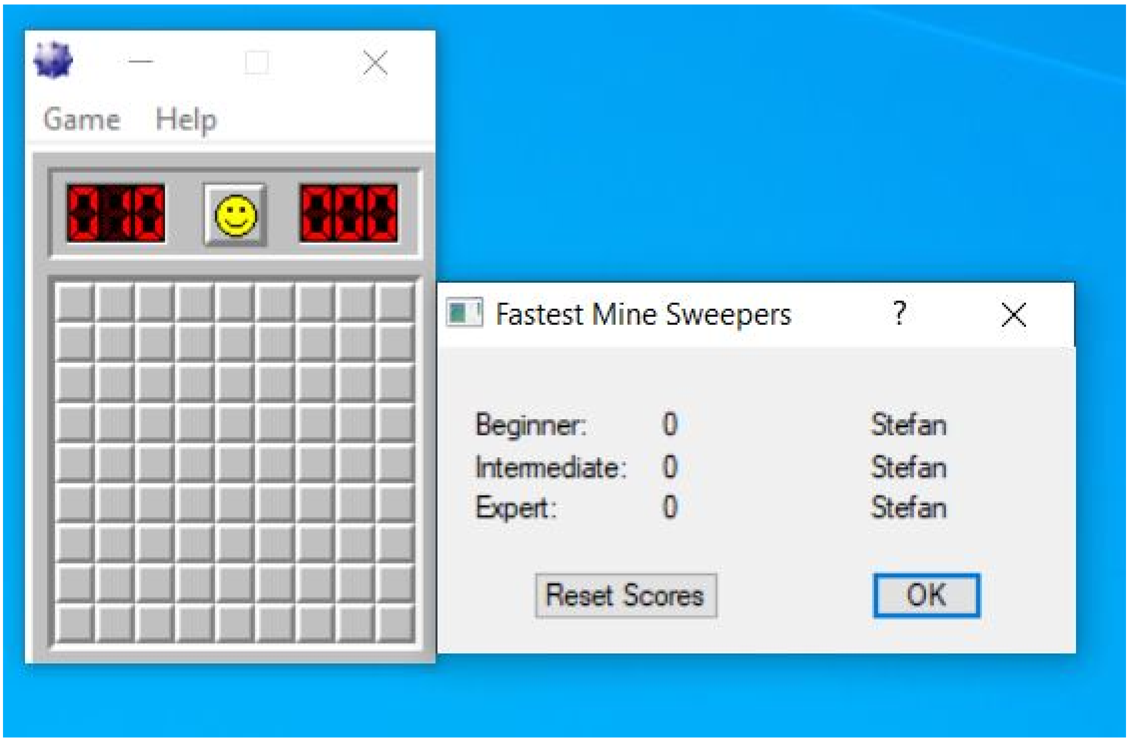
```
0    s e c o n d s          S t e f a n
```

```
    挘攀挀漀渀搀猀ऀ匀琀攀昀愀渀
```

```
0    s e c o n d s          S t e f a n
```

```
00 30 00 00 00 20 00 73 00 65 00 63 00 6F 00 6E
00 64 00 73 00 09 00 53 00 74 00 65 00 66 00 61
00 6E 00 00 00 00 00 00
```

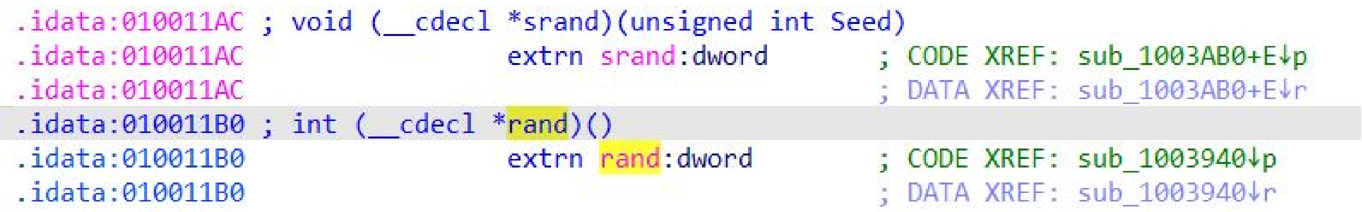
Here is the program patched with my name for all levels of difficulty and the number of seconds set to 0:



Beginner:	0	Stefan
Intermediate:	0	Stefan
Expert:	0	Stefan

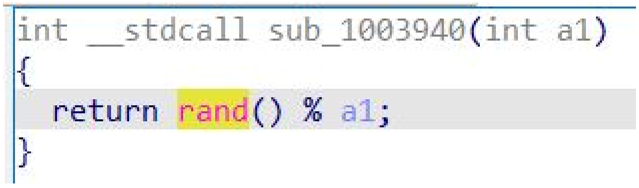
Flag the bombs

We know that bombs are randomly placed on the grid, so I started by searching for the `rand()` function in the `.idata` segment.



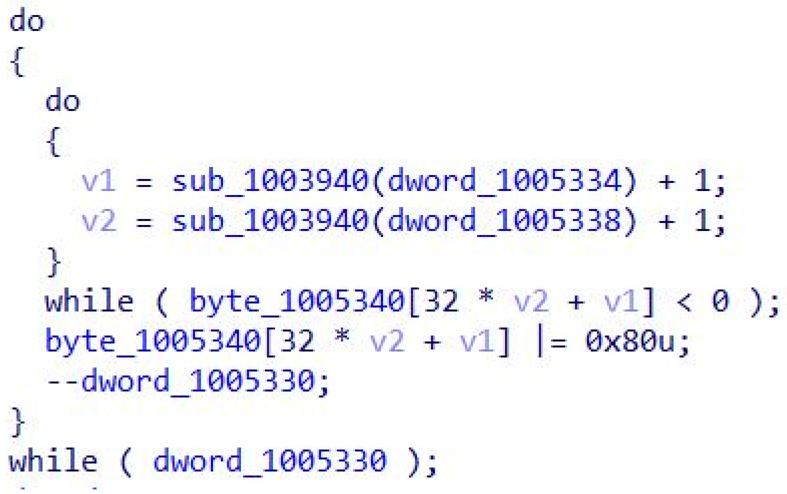
```
.idata:010011AC ; void (__cdecl *srand)(unsigned int Seed)
.idata:010011AC                 extrn srand:dword       ; CODE XREF: sub_1003AB0+E↓p
.idata:010011AC                                         ; DATA XREF: sub_1003AB0+E↓r
.idata:010011B0 ; int (__cdecl *rand)()
.idata:010011B0                 extrn rand:dword        ; CODE XREF: sub_1003940↓p
.idata:010011B0                                         ; DATA XREF: sub_1003940↓r
```

I found it being referenced in the function `sub_1003940`:



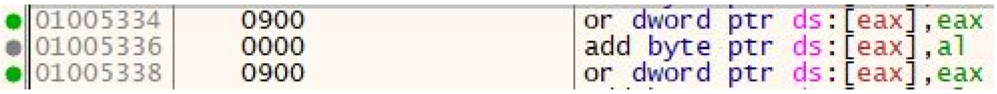
```
int __stdcall sub_1003940(int a1)
{
  return rand() % a1;
}
```

Next, this method is called from `sub_100367A` function. Here I found this interesting snippet:



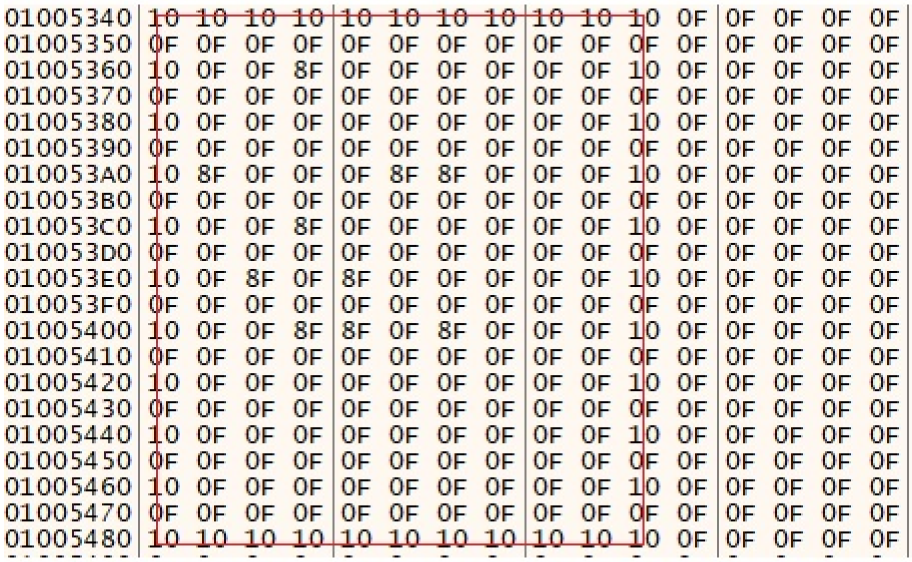
```
do
{
  do
  {
    v1 = sub_1003940(dword_1005334) + 1;
    v2 = sub_1003940(dword_1005338) + 1;
  }
  while ( byte_1005340[32 * v2 + v1] < 0 );
  byte_1005340[32 * v2 + v1] |= 0x80u;
  --dword_1005330;
}
while ( dword_1005330 );
```

Now, I run the program in the debugger and followed in dump the addresses `0x01005334` and `0x01005338`, which both store the value `9`. I found the parameters for the function that calls the `rand()`. These are the width and the height of the grid. So, `v1` and `v2` variables in the previous snippet, store the coordinats of the bombs.



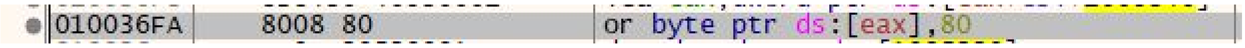
```
01005334    0900    or dword ptr ds:[eax],eax
01005336    0000    add byte ptr ds:[eax],al
01005338    0900    or dword ptr ds:[eax],eax
```

At address `0x01005340`, the grid is stored as a matrix. The blank cells are represented by `F` and the bombs with `8F`. The margins are stored as `10` bytes. If we place a flag on a bomb position at runtime, the value changes to `8E`.



```
01005340  10 10 10 10  10 10 10 10  10 10 10 0F  0F 0F 0F 0F
01005350  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005360  10 0F 0F 8F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005370  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005380  10 0F 0F 0F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005390  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
010053A0  10 8F 0F 0F  0F 8F 8F 0F  0F 0F 10 0F  0F 0F 0F 0F
010053B0  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
010053C0  10 0F 0F 8F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
010053D0  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
010053E0  10 0F 8F 0F  8F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
010053F0  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005400  10 0F 0F 8F  8F 0F 8F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005410  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005420  10 0F 0F 0F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005430  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005440  10 0F 0F 0F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005450  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005460  10 0F 0F 0F  0F 0F 0F 0F  0F 0F 10 0F  0F 0F 0F 0F
01005470  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F  0F 0F 0F 0F
01005480  10 10 10 10  10 10 10 10  10 10 10 0F  0F 0F 0F 0F
```

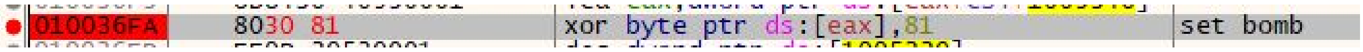
The line `byte_1005340[32 * v2 + v1] |= 0x80u` in the snippet sets the bomb. A bitwise `or` operation is performed on value `0x0F` with value `0x80`, and it obtains `0x8F`. We need an operation to get `0x8E`.



```
010036FA  8008 80  or byte ptr ds:[eax],80
```

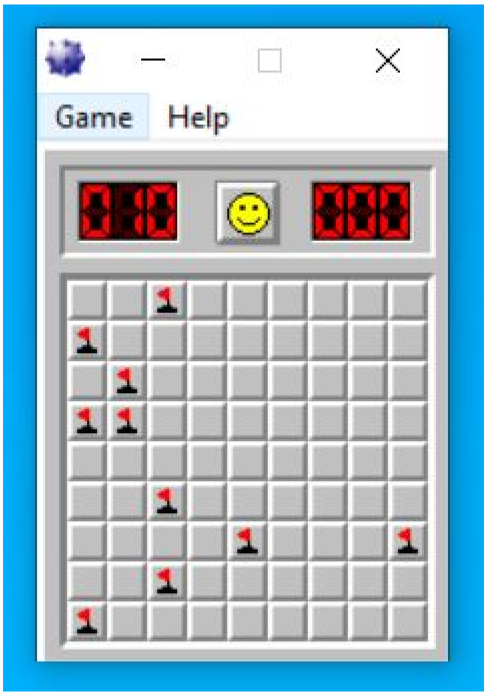
As we can't reverse the `or` operation, I used the `xor` operation which is non-destructive. So, we need to find which value xored with `F` results in `8E`:

```
0x0F ^ ? = 0x8E
0x8E ^ 0x0F = 0x81
```



```
010036FA  8030 81  xor byte ptr ds:[eax],81  set bomb
```

Finally, patching the program with this new instruction, results in the game to display the flags where the bombs are:

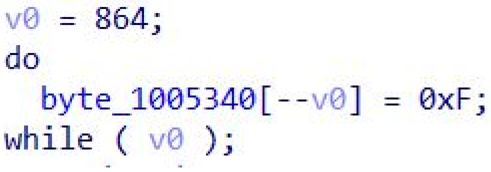


Question mark the blanks

We saw earlier that the matrix contais the value `8F` for the bombs and `F` for the blanks. Also, a question mark is represented as `8D` if it's placed on bomb cell, or as `D` otherwise.

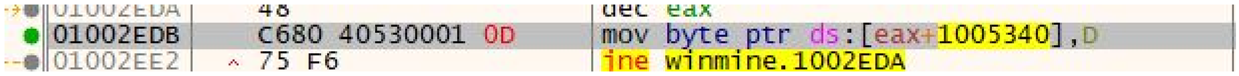
My idea was to replace all `F` values with `D` to display all grid cells as question marks and then to modify again the instruction that set the bombs to not be displayed as `?`.

We already know that the matrix is accessed using `byte_1005340` variable, so in IDA I searched for more references to it. I found the function `sub_1002ED5` which fills a large memory space with `0xF` bytes.



```
v0 = 864;
do
  byte_1005340[--v0] = 0xF;
while ( v0 );
```

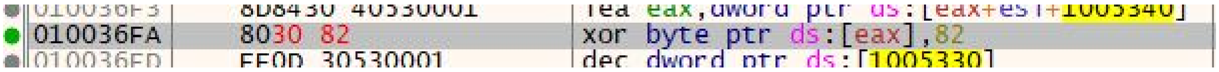
Using the debugger, I changed the byte value from `F` to `D`:



```
01002EDA | 48                 | dec eax
01002EDB | C680 40530001 0D   | mov byte ptr ds:[eax+1005340],D
01002EE2 | 75 F6              | jne winmine.1002EDA
```

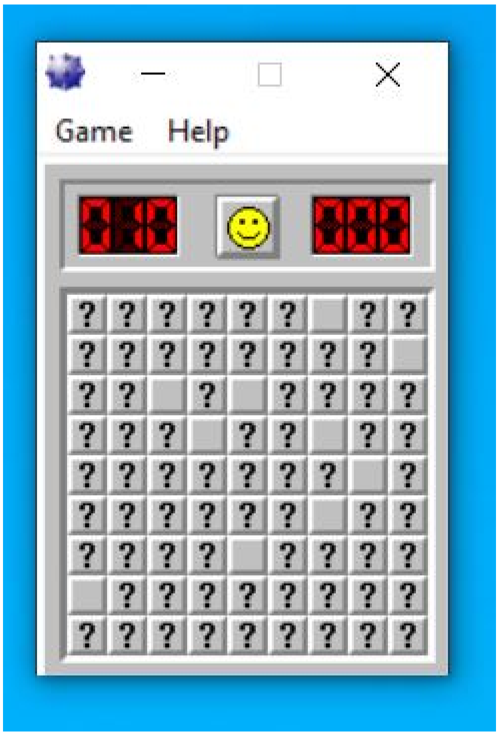
For the bombs to not be marked as `?`, I had to find an instruction that transforms the value `D` to `8F`. Similarly to the previous task, I xored `D` with `82` to obtain `8F` (the bomb):

```
0x0D ^ ? = 0x8F
0x8F ^ 0x0D = 0x82
```



```
010036F3 | 8D8430 40530001    | lea eax,dword ptr ds:[eax+esi+1005340]
010036FA | 8030 82            | xor byte ptr ds:[eax],82
010036FD | FF0D 30530001      | dec dword ptr ds:[1005330]
```

Finally, patching the program with these two new instructions, results in the game to put the question mark on positions that are blank (not bomb):


Game
Help
010
000

?	?	?	?	?	?		?	?
?	?	?	?	?	?	?	?	
?	?		?		?	?	?	?
?	?	?		?	?		?	?
?	?	?	?	?	?	?		?
?	?	?	?	?	?		?	?
?	?	?	?		?	?	?	?
	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?