

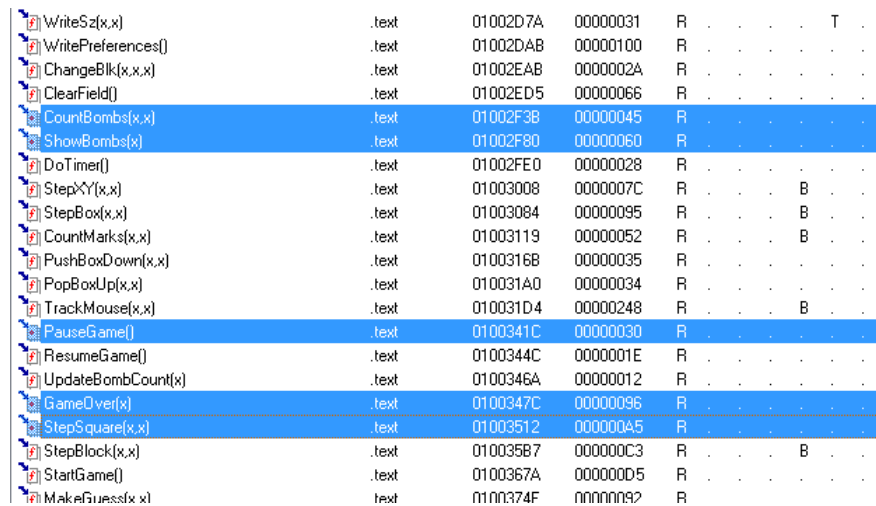
Writeup

Main C2 Program

The main program creates a process into winmine.exe and records a handle to it. On command we inject one of 5 dlls, each of them containing the code relevant to the task within their DllMain functions. In order to be able to hit each option more than once we also clean up by freeing the library we created in the remote thread as well.

Analysis and Report

As soon as we open up IDA, we clearly see some nice functions, namely ShowBombs and PauseTimer. These do as expected, either showing the locations of the bombs or pausing the timer if it is currently running. We also see a GameOver function which takes a boolean as an argument, however using this is cheating, even by our standards :)



WriteSz(x,x)	.text	01002D7A	00000031	R	T	.
WritePreferences()	.text	01002DAB	00000100	R
ChangeBlk(x,x,x)	.text	01002EAB	0000002A	R
ClearField()	.text	01002ED5	00000066	R
CountBombs(x,x)	.text	01002F38	00000045	R
ShowBombs(x)	.text	01002F80	00000060	R
DoTimer()	.text	01002FE0	00000028	R
StepXY(x,x)	.text	01003008	0000007C	R	.	.	.	B	.	.
StepBox(x,x)	.text	01003084	00000095	R	.	.	.	B	.	.
CountMarks(x,x)	.text	01003119	00000052	R	.	.	.	B	.	.
PushBoxDown(x,x)	.text	01003168	00000035	R
PopBoxUp(x,x)	.text	010031A0	00000034	R
TrackMouse(x,x)	.text	010031D4	00000248	R	.	.	.	B	.	.
PauseGame()	.text	0100341C	00000030	R
ResumeGame()	.text	0100344C	0000001E	R
UpdateBombCount(x)	.text	0100346A	00000012	R
GameOver(x)	.text	0100347C	00000096	R
StepSquare(x,x)	.text	01003512	000000A5	R
StepBlock(x,x)	.text	010035B7	000000C3	R	.	.	.	B	.	.
StartGame()	.text	0100367A	000000D5	R
MakeGuess(x,x)	.text	0100374F	00000092	R

Figure 1: important functions in IDA

Although the previous two functions take care of mines_visible and free_timer, we still know nothing about get_layout, auto_win, etc. However, we see in the ShowBombs function, and many others that there exists a common memory address 0x01005360 which stores information on the board. Dumping its values at various points in the program shows us that it takes on a few common traits:

- The board is a fixed size. Variables are used at runtime to keep track of what memory is being used or not. We can thus get the value of tile (i,j) with $*(0x01005360 + j + 0x20 * i)$.

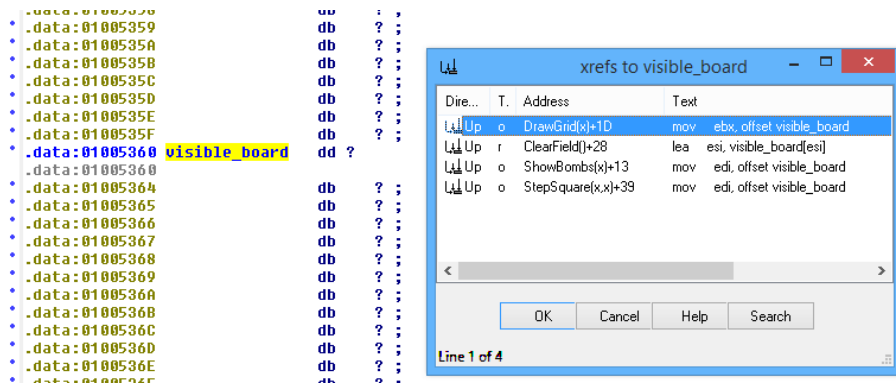


Figure 2: nothing much to see here. . .

- We can get the dimensions of the board from some variables in memory, and get the bomb count at each tile from the function CountBombs.
- We see that all bombs, visible or invisible or flagged or whatever all have 0x80 set, in other words it is a flag for a tile being a bomb tile.
- Also, all numbers that have been revealed are flagged with 0x40
- Less important, flags have bitmask 0xFE and question marks have bitmask 0xFD
- Tiles are all masked with 0x1F, so that these bits are not shown. . . as a result the values taken up by the tiles can be between 0 and 16, after being anded with 0x1F. (see below)

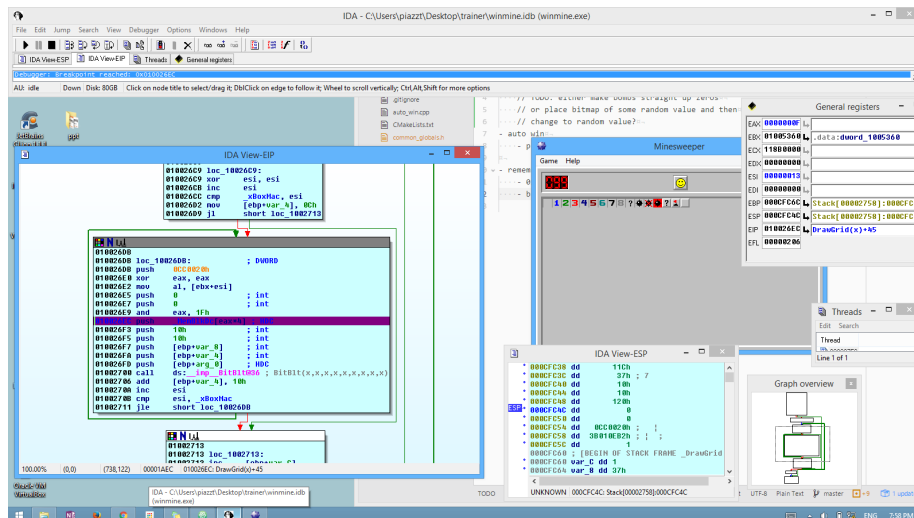


Figure 3: playing with the debugger

Finally, we later see the function StepSquare which allows us to complete both GetLayout and AutoWin - we can simply iterate over the board and perform operations based on the bits that are set on each tile.

We handle disable_mines specially because there is no simple function that gives us this functionality (why would there be?). We instead patch some bytes at runtime.

Writing the Trainer

The trainer was written using CMake, which automates builds for visual c++/msbuild, as well as MinGW (which currently is not supported). The unimportant driver is trainer.cpp which heavily uses inject.cpp. trainer.cpp opens up a console and winmine.exe and asks the user for a command, and will inject the correct dll depending on the command.

```
C:\Users\piazzt\Desktop\trainer>cd build
C:\Users\piazzt\Desktop\trainer\build>cmake .. -G"Unix Makefiles"
-- The C compiler identification is GNU 4.9.1
-- The CXX compiler identification is GNU 4.9.1
-- Check for working C compiler: c:/MinGW/bin/gcc.exe
-- Check for working C compiler: c:/MinGW/bin/gcc.exe -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - failed
-- Check for working CXX compiler: c:/MinGW/bin/c++.exe
-- Check for working CXX compiler: c:/MinGW/bin/c++.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - failed
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/piazzt/Desktop/trainer/build

C:\Users\piazzt\Desktop\trainer\build>make
Scanning dependencies of target auto_win
[ 14%] Building CXX object CMakeFiles/auto_win.dir/auto_win.cpp.obj
Linking CXX shared library libauto_win.dll
[ 14%] "Built target auto_win"
Scanning dependencies of target disable_mines
[ 28%] Building CXX object CMakeFiles/disable_mines.dir/disable_mines.cpp.obj
Linking CXX shared library libdisable_mines.dll
[ 28%] "Built target disable_mines"
Scanning dependencies of target freeze_timer
[ 42%] Building CXX object CMakeFiles/freeze_timer.dir/freeze_timer.cpp.obj
Linking CXX shared library libfreeze_timer.dll
[ 42%] "Built target freeze_timer"
Scanning dependencies of target get_layout
[ 57%] Building CXX object CMakeFiles/get_layout.dir/get_layout.cpp.obj
Linking CXX shared library libget_layout.dll
[ 57%] "Built target get_layout"
Scanning dependencies of target mines_visible
[ 71%] Building CXX object CMakeFiles/mines_visible.dir/mines_visible.cpp.obj
Linking CXX shared library libmines_visible.dll
[ 71%] "Built target mines_visible"
Scanning dependencies of target trainer
[ 85%] Building CXX object CMakeFiles/trainer.dir/trainer.cpp.obj
[100%] Building CXX object CMakeFiles/trainer.dir/inject.cpp.obj
Linking CXX executable trainer.exe
[100%] "Built target trainer"

C:\Users\piazzt\Desktop\trainer\build>
```

Figure 4: pretty output! - Cmake

Each dll is written in the same format with a DllMain. These are all compiled

in 32bit because it must match the bitwidth of the target process, winmine. Although intensive work should not be done in DllMain, for example calling MessageBox, since all the other modules have been loaded already we do not really need to worry about doing annoying stuff while holding the global loader lock.

Each dll essentially works the same. They all take some hardcoded values as pointers to variables or functions in the program and take advantage of them in some way.

Features

`get_layout`

- We iterate over the board (located at address 0x01005360) and call CountBombs function in winmine.exe which gets all adjacent bombs.
- Because winmine.exe determines all of its important tiles based on flags, while also not determining any numbers until the user has clicked on a good tile, we need to mask the return value with 0x80, the indicator for the existence of a bomb.

`mines_visible`

- We make use of the built in ShowBombs function, which takes 0xa in as an argument (is it the bitmap?). This displays all of the bombs.
- Funny enough, if you click on a bomb after ShowBombs gets called, we see that we do not lose the game. This is because MineSweeper is not able to penalize someone on the very first click, so it moves the bomb elsewhere.

`freeze_timer`

- We again call a built-in function to PauseGame(), though this only works after the timer has started initially, i.e. the game was started.

`disable_mines`

- This was the most involved system. We VirtualProtect all memory relevant to us with very XRW permissions temporarily, and dump some code in, completely overwriting what was there. This allows us to skip the call to GameOver(0), which inevitably exits the game.

auto_win

- We use a scheme similar to `get_layout` except if we are not on a bomb tile we step on it with `StepSquare`.