Reverse Engineering Report Of Minesweeper

(CS 265-01 Cryptography and Computer Security)

Contributed By
Jayant Shelke
Mustafa Badshah

Mustafa Badshah – mustafa.badshah@sjsu.edu Jayant Shelke – jayant.shelke@sjsu.edu

Contents

1. Introduction

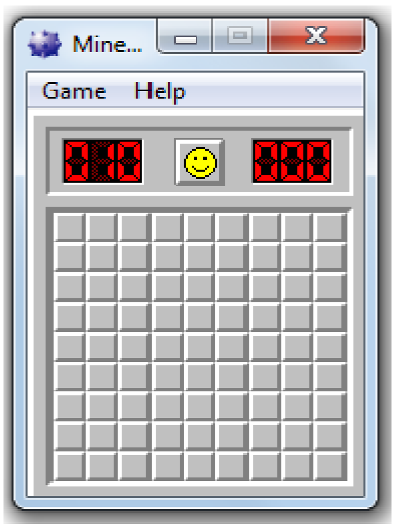
Software reverse engineering is taking apart an object or software to understand how it works. It involves reversing a programs binary machine code back into its source code. Reverse engineering is done mainly for the following reasons:

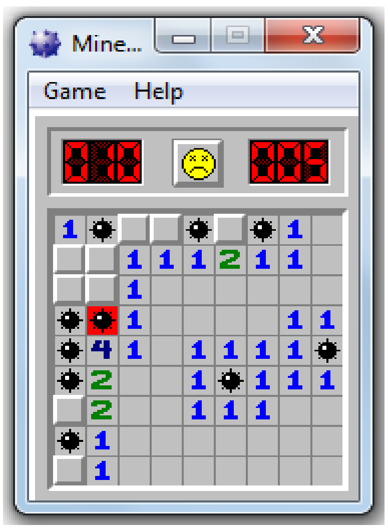
- To retrieve source code of a program for which the source code is lost.
- To understand how a program performs certain operations.
- To improve performance of a program.
- To fix a bug or an error.
- To identify malicious content in a program.
- To copy or duplicate a programs functionality.
- To overcome some security aspect.

Gaming software is notoriously affected by issues such as piracy, payment bypassing or just plain cheating. Industry analysts Gartner estimate that the worldwide video gaming market was worth an estimated $93 billion in 2013. However, according to another study, only 20 per cent of games realise a profit. One of the main reasons why some 80 per cent of games don't generate a profit is down to piracy and cheating which is due to reverse engineering and patching of games and releasing their keygens and cracks.[2]

2. Reverse Engineering of Minesweeper

Microsoft's minesweeper is a single player computer game created by Curt Johnson. The goal of the game is to uncover all the squares that do not contain mines without being "blown up" by clicking on a square with a mine underneath. The location of the mines is discovered by a process of logic. Clicking on the square will reveal what is hidden underneath the chosen square or squares. Some squares are blank but some contain numbers (1 to 8), each number being the number of mines adjacent to the uncovered square. To help avoid hitting a mine, the location of a suspected mine can be marked by flagging it with the right mouse button. The game is won once all blank squares have been uncovered without hitting a mine, any remaining mines not identified by flags being automatically flagged by the computer.[1]





There are multiple areas in this game which can be targeted to cheat our way through the game:

1. The Timer: To give best time statistics at winning the game.
2. The Statistics Board: To change the owners of the best time.
3. Mine Field: To display the mines to the player.
4. Mine Count: Reduce mine count in the mine fields.

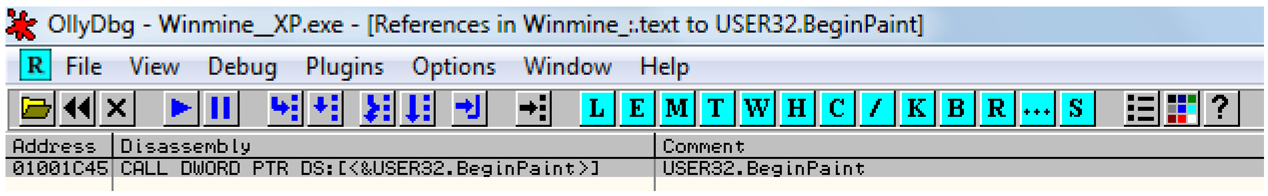
Here we target the mine field and display to the user all the mines at the start of the game. All that the user needs to do click on the remaining tiles which do not contain the mines and win the game.

Steps for Exposing the Mine Field:

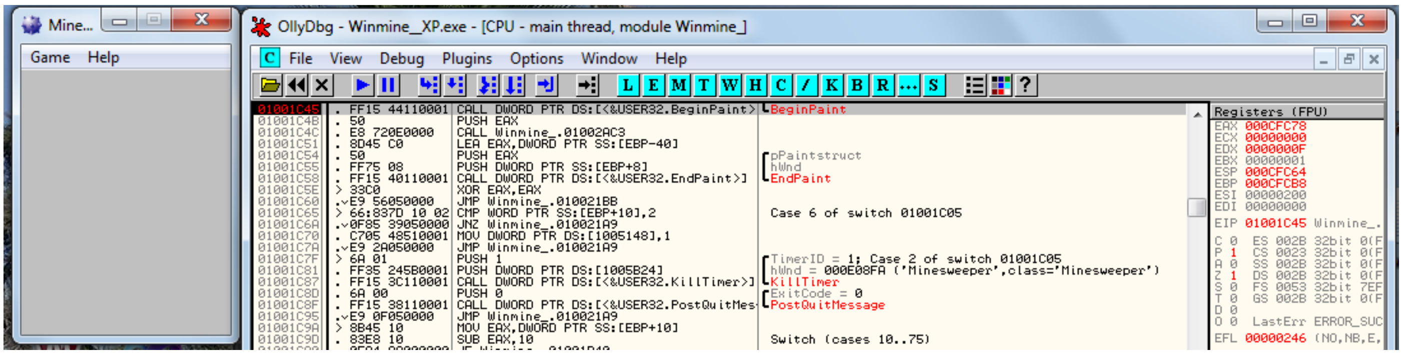
We use OllyDbg to disassemble the Minesweeper executable and reassemble the modified code.

2.1 Debugging and Disassembly: Locating the Mine Field:

To locate the minefield needs is a little knowledge of the WinAPI. When an application wants to draw and use graphics, it must use the **User32.dll** along with Windows GDI (GDI32.dll) both which export functions to enable such tasks. One of these functions is **BeginPaint** which sets up a particular window for painting. We know that minesweeper has to draw something so we search for the function in the import lists and find references to BeginPaint. We set up the debugger to stop when the game starts to lay out its graphical components.


OllyDbg - Winmine__XP.exe - [References in Winmine_:.text to USER32.BeginPaint]
File View Debug Plugins Options Window Help

```
Address   Disassembly                                   Comment
01001C45  CALL DWORD PTR DS:[<&USER32.BeginPaint>]      USER32.BeginPaint
```


Mine...
Game Help
OllyDbg - Winmine__XP.exe - [CPU - main thread, module Winmine_]
File View Debug Plugins Options Window Help

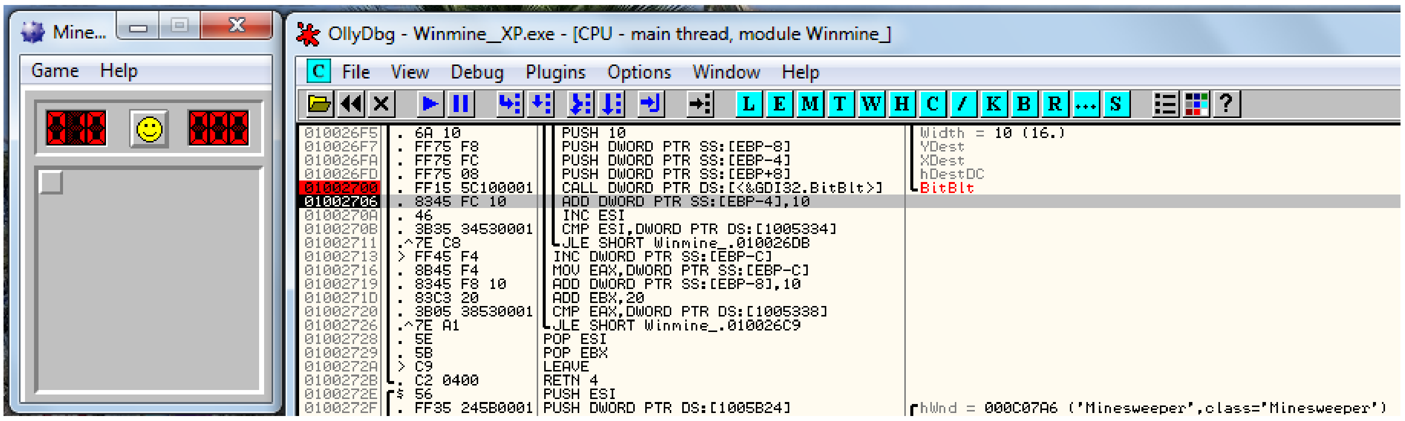
```
01001C45  . FF15 44110001  CALL DWORD PTR DS:[<&USER32.BeginPaint>  └BeginPaint
01001C4B  . 50             PUSH EAX
01001C4C  . E8 720E0000    CALL Winmine_.01002AC3
01001C51  . 8D45 C0        LEA EAX,DWORD PTR SS:[EBP-40]
01001C54  . 50             PUSH EAX                                 ┌pPaintstruct
01001C55  . FF75 08        PUSH DWORD PTR SS:[EBP+8]                │hWnd
01001C58  . FF15 40110001  CALL DWORD PTR DS:[<&USER32.EndPaint>]   └EndPaint
01001C5E  > 33C0           XOR EAX,EAX
01001C60  .vE9 56050000    JMP Winmine_.010021BB
01001C65  > 66:837D 10 02  CMP WORD PTR SS:[EBP+10],2                Case 6 of switch 01001C05
01001C6A  .v0F85 39050000  JNZ Winmine_.010021A9
01001C70  . C705 48510001  MOV DWORD PTR DS:[1005148],1
01001C7A  .vE9 2A050000    JMP Winmine_.010021A9
01001C7F  > 6A 01          PUSH 1                                   ┌TimerID = 1; Case 2 of switch 01001C05
01001C81  . FF35 245B0001  PUSH DWORD PTR DS:[1005B24]              │hWnd = 000E08FA ('Minesweeper',class='Minesweeper')
01001C87  . FF15 3C110001  CALL DWORD PTR DS:[<&USER32.KillTimer>]  └KillTimer
01001C8D  . 6A 00          PUSH 0                                   ┌ExitCode = 0
01001C8F  . FF15 38110001  CALL DWORD PTR DS:[<&USER32.PostQuitMes  └PostQuitMessage
01001C95  .vE9 0F050000    JMP Winmine_.010021A9
01001C9A  > 8B45 10        MOV EAX,DWORD PTR SS:[EBP+10]
01001C9D  . 83E8 10        SUB EAX,10                                Switch (cases 10..75)
```

```
Registers (FPU)
EAX 000CFC78
ECX 00000000
EDX 0000000F
EBX 00000001
ESP 000CFC64
EBP 000CFCB8
ESI 00000200
EDI 00000000

EIP 01001C45 Winmine_.

C 0  ES 002B 32bit 0(F
P 1  CS 0023 32bit 0(F
A 0  SS 002B 32bit 0(F
Z 1  DS 002B 32bit 0(F
S 0  FS 0053 32bit 7EF
T 0  GS 002B 32bit 0(F
D 0
O 0  LastErr ERROR_SUC
EFL 00000246 (NO,NB,E,
```

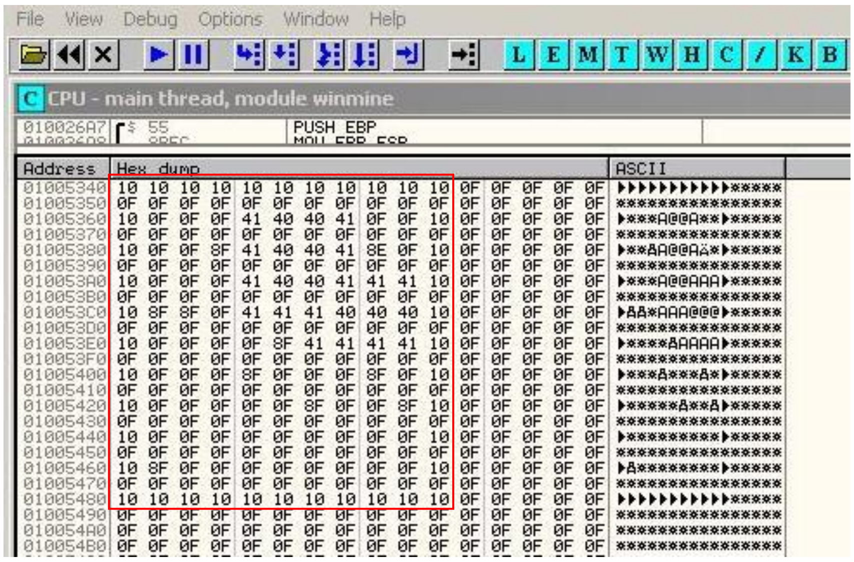
We follow a typical reverse engineering approach to find out procedures which draw out cells. We carefully step over each instruction until we start to see cells being drawn out and then step into the function which calls out this task. The **BitBlt** function is what draws out each cell of the mine field. When we analyze the set of instructions, we understand it is sort of a while loop where *esi* is a counter starting at 0 and *ebx* is the array beginning at **0x1005340**.


Mine...
Game Help
OllyDbg - Winmine__XP.exe - [CPU - main thread, module Winmine_]
File View Debug Plugins Options Window Help

```
010026F5  . 6A 10          PUSH 10                                   Width = 10 (16.)
010026F7  . FF75 F8        PUSH DWORD PTR SS:[EBP-8]                 YDest
010026FA  . FF75 FC        PUSH DWORD PTR SS:[EBP-4]                 XDest
010026FD  . FF75 08        PUSH DWORD PTR SS:[EBP+8]                 hDestDC
01002700  . FF15 5C100001  CALL DWORD PTR DS:[<&GDI32.BitBlt>]       └BitBlt
01002706  . 8345 FC 10     ADD DWORD PTR SS:[EBP-4],10
0100270A  . 46             INC ESI
0100270B  . 3B35 34530001  CMP ESI,DWORD PTR DS:[1005334]
01002711  .^7E C8          └JLE SHORT Winmine_.010026DB
01002713  > FF45 F4        INC DWORD PTR SS:[EBP-C]
01002716  . 8B45 F4        MOV EAX,DWORD PTR SS:[EBP-C]
01002719  . 8345 F8 10     ADD DWORD PTR SS:[EBP-8],10
0100271D  . 83C3 20        ADD EBX,20
01002720  . 3B05 38530001  CMP EAX,DWORD PTR DS:[1005338]
01002726  .^7E A1          └JLE SHORT Winmine_.010026C9
01002728  . 5E             POP ESI
01002729  . 5B             POP EBX
0100272A  > C9             LEAVE
0100272B  . C2 0400        RETN 4
0100272E  $ 56             PUSH ESI
0100272F  . FF35 245B0001  PUSH DWORD PTR DS:[1005B24]              ┌hWnd = 000C07A6 ('Minesweeper',class='Minesweeper')
```

To verify we continue running the game and dump the memory starting at 0x1005340 to check what it looks like. Clearly a similar layout is seen in the memory dump. We can slightly recognise the border of the minefield which is represented by **10's** and the empty unused cells are **0F's**. By further testing we get the values of other fields as well and they are represented as follows:

8A – Mine
8E – Flagged Mine
8D – Guess
8F – Mine under a box
F – Unclicked box

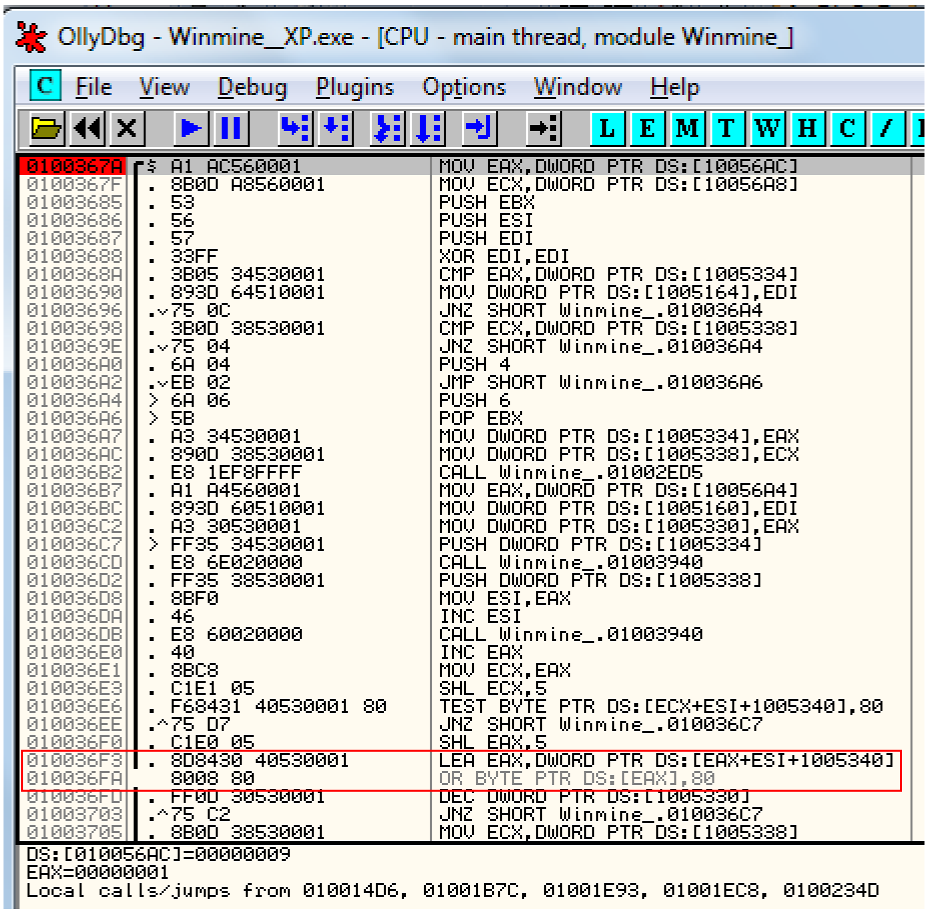

File View Debug Options Window Help
L E M T W H C / K B
CPU - main thread, module winmine

```
010026A7 ┌$ 55          PUSH EBP

Address   Hex dump                                         ASCII
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 0F 41 40 40 41 0F 0F 10 0F 0F 0F 0F 0F  ►☼☼☼A@@A☼☼►☼☼☼☼☼
01005370  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
01005380  10 0F 0F 8F 41 40 40 41 8E 0F 10 0F 0F 0F 0F 0F  ►☼☼ÅA@@AÄ☼►☼☼☼☼☼
01005390  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
010053A0  10 0F 0F 0F 41 40 40 41 41 41 10 0F 0F 0F 0F 0F  ►☼☼☼A@@AAA►☼☼☼☼☼
010053B0  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
010053C0  10 8F 8F 0F 41 41 41 40 40 40 10 0F 0F 0F 0F 0F  ►ÅÅ☼AAA@@@►☼☼☼☼☼
010053D0  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
010053E0  10 0F 0F 0F 0F 8F 41 41 41 41 10 0F 0F 0F 0F 0F  ►☼☼☼☼ÅAAAA►☼☼☼☼☼
010053F0  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
01005400  10 0F 0F 0F 8F 0F 0F 0F 8F 0F 10 0F 0F 0F 0F 0F  ►☼☼☼Å☼☼☼Å☼►☼☼☼☼☼
01005410  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
01005420  10 0F 0F 0F 0F 0F 8F 0F 0F 8F 10 0F 0F 0F 0F 0F  ►☼☼☼☼☼Å☼☼Å►☼☼☼☼☼
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 8F 0F 0F 0F 0F 0F 0F 0F 0F 10 0F 0F 0F 0F 0F  ►Å☼☼☼☼☼☼☼☼►☼☼☼☼☼
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  ►►►►►►►►►►►☼☼☼☼☼
01005490  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
010054A0  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
010054B0  0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F 0F  ☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼☼
```

2.2 Hex Edit: Display the Mines

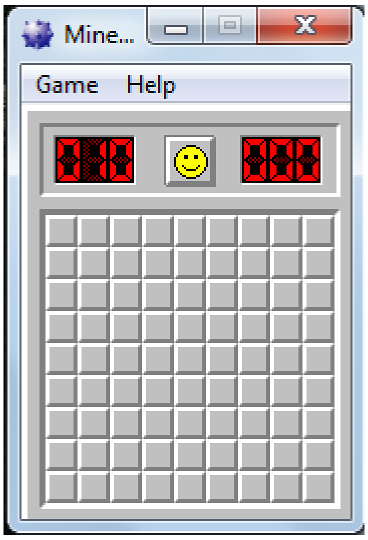
Now we are going to review the functions that deal with the mines. We first find out at which address the count of mines is stored and find references to opcode which writes or accesses that memory location. The mine count is stored at address **0x01001E93** and referenced by the function at **0x0100367A**.


OllyDbg - Winmine_XP.exe - [CPU - main thread, module Winmine_]
File View Debug Plugins Options Window Help
L E M T W H C /

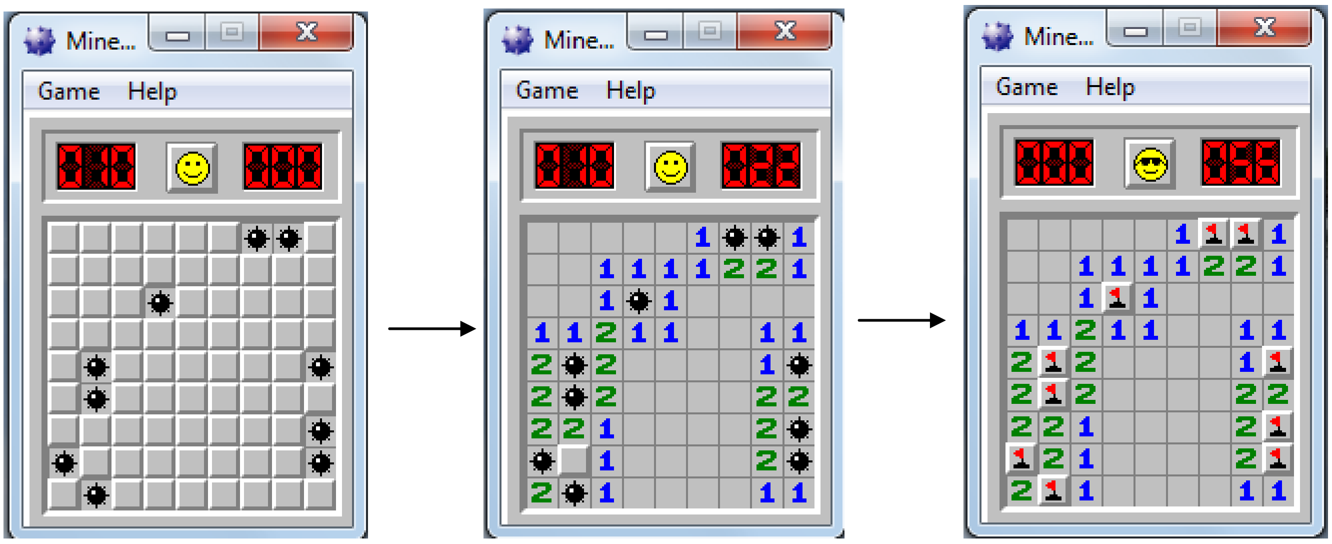
```
0100367A ┌$ A1 AC560001        MOV EAX,DWORD PTR DS:[10056AC]
0100367F │. 8B0D A8560001      MOV ECX,DWORD PTR DS:[10056A8]
01003685 │. 53                 PUSH EBX
01003686 │. 56                 PUSH ESI
01003687 │. 57                 PUSH EDI
01003688 │. 33FF               XOR EDI,EDI
0100368A │. 3B05 34530001      CMP EAX,DWORD PTR DS:[1005334]
01003690 │. 893D 64510001      MOV DWORD PTR DS:[1005164],EDI
01003696 │.v75 0C              JNZ SHORT Winmine_.010036A4
01003698 │. 3B0D 38530001      CMP ECX,DWORD PTR DS:[1005338]
0100369E │.v75 04              JNZ SHORT Winmine_.010036A4
010036A0 │. 6A 04              PUSH 4
010036A2 │.vEB 02              JMP SHORT Winmine_.010036A6
010036A4 │> 6A 06              PUSH 6
010036A6 │> 5B                 POP EBX
010036A7 │. A3 34530001        MOV DWORD PTR DS:[1005334],EAX
010036AC │. 890D 38530001      MOV DWORD PTR DS:[1005338],ECX
010036B2 │. E8 1EF8FFFF        CALL Winmine_.01002ED5
010036B7 │. A1 A4560001        MOV EAX,DWORD PTR DS:[10056A4]
010036BC │. 893D 60510001      MOV DWORD PTR DS:[1005160],EDI
010036C2 │. A3 30530001        MOV DWORD PTR DS:[1005330],EAX
010036C7 │> FF35 34530001      PUSH DWORD PTR DS:[1005334]
010036CD │. E8 6E020000        CALL Winmine_.01003940
010036D2 │. FF35 38530001      PUSH DWORD PTR DS:[1005338]
010036D8 │. 8BF0               MOV ESI,EAX
010036DA │. 46                 INC ESI
010036DB │. E8 60020000        CALL Winmine_.01003940
010036E0 │. 40                 INC EAX
010036E1 │. 8BC8               MOV ECX,EAX
010036E3 │. C1E1 05            SHL ECX,5
010036E6 │. F68431 40530001 80 TEST BYTE PTR DS:[ECX+ESI+1005340],80
010036EE │.^75 D7              JNZ SHORT Winmine_.010036C7
010036F0 │. C1E0 05            SHL EAX,5
010036F3 ││. 8D8430 40530001    LEA EAX,DWORD PTR DS:[EAX+ESI+1005340]
010036FA │  8008 80            OR BYTE PTR DS:[EAX],80
010036FD ││. FF0D 30530001      DEC DWORD PTR DS:[1005330]
01003703 ││.^75 C2              JNZ SHORT Winmine_.010036C7
01003705 ││. 8B0D 38530001      MOV ECX,DWORD PTR DS:[1005338]
DS:[010056AC]=00000009
EAX=00000001
Local calls/jumps from 010014D6, 01001B7C, 01001E93, 01001EC8, 0100234D
```

Analyzing this function we come to this Opcode: **OR BYTE PTR DS:[EAX], 80** which hides the mines in the minefield. We need to change this specific code to be able to view all the mines in the minefield.
Changed Opcode: **MOV BYTE PTR DS:[EAX], 8A**
This change makes the mine cells viewable as mines as the value for the mines is 8A as seen previously in memory dump. After this patch every time a game is started we are able to view the mines.
We know save this modified code as a new executable using OllyDbg.

Before Patch:



After Patch:



3. Work Distribution:

Task	Mustafa Badshah	Jayant Shelke
1. Setting Up Virtual Environment	50%	50%
2. Disassembling and Debugging Code	40%	60%
3. Understanding Disassembled Code	60%	40%
4. Changing Opcodes and Testing	60%	40%
5. Patching Executable	40%	60%
6. Report Creation	50%	50%

4. References

1. https://en.wikipedia.org/wiki/Microsoft_Minesweeper
2. http://www.develop-online.net/analysis/defending-online-games-from-piracy-cheating-and-fraud/0198678
3. https://0x00sec.org/t/game-hacking-winxp-minesweeper/1266