© The CodeBreakers-Journal, Vol.1, No.2. (2004)
http://www.CodeBreakers-Journal.com

Minesweeper Reversing

Author: ZaiRoN

Abstract

Minesweeper is one of the millions M$ games and in this tutorial I will explain you how to add a new feature on the game. The new feature will give you the ability to view where the bombs are.

Target:
MineSweeper (English version for WinXP/W2k)

Program description:
Don't you ever played with this game in your boring days? I can't believe you :-p

Tools:
- a debugger (Ollydbg)
- a disassembler (Ida)
- a resource editor (Resource Hacker)
- Spy++
- c and asm compiler (lcc, masm)

music: Black Sabbath - Black Sabbath

Preamble
We will show where the bombs are as a *mouse-over* effect as the cursor moves over each grid square. We will add all the modifications inside a dll and we will write a loader to patch the program in memory. This is not only a reversing exercize but also a programming exercize because you will have to code a dll, work with GDI and finally write a little loader. This tutorial is a resume (with some differences) of a project at the RCE board, you can find the thread here (http://66.98.132.48/forum/showthread.php?t=5003). We start finding all the places where we will have to put a jump to our code; after that we will write the code to add to the program, we will write the dll and finally a little loader.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Contents

1. Find the point where to add the new menu item

We should be able to pass from normal to cheat mode (and viceversa) therefore the best way is to use a new menu item. I will add the new item inside 'Game' menu, after the 'New' item.
We can use a resource editor but remembering that we need to patch the file in memory, we have to think in a different way. How can I add a new item into an existing menu? We can use the function InsertMenuItem. This function will be placed into a cave with all the other modifications needed by the program. We will see later how to use this function, for the moment you only have to know that InsertMenuITem will be called in the initialization part of the program, exactly after the menu has been loaded (because it needs the menu handle...).
Now, we should find the place where to insert the jump to the new cave. The program uses LoadMenu to load the menu:

```
:010014B9 push 1F4h   <------- lpMenuName
:010014BE push hInstance   <-- hInstance
:010014C4 call ds:LoadMenuW
:010014CA push 1F5h   <------- lpTableName
:010014CF mov hMenu, eax
:010014D4 push hInstance   <-- hInstance
:010014DA call ds:LoadAcceleratorsW
:010014E0 mov [ebp+hAccTable], eax
:010014E3 call sub_10023DB
:010014E8 mov eax, dword_10052A4
:010014ED mov ecx, yBottom
```

When the menu has been loaded you can insert the new item, I decided to put a jump to the code that will call the InsertMenuItem directly at 10014E8. I did choose this particular address only because it has the same number of bytes of the new jump instruction...

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

2. Find the point where the program checks whether a menu item has been pressed

This task is useful because we need to add/manage the new menu item, the special mode option. If you check the 'special mode' item, the game will show you the bomb, otherwise the game will remain the same.

There are many ways to solve this second point.

- You can use Spy++, a little application that gives you a view of many informations about a process that is running on your system.
Run Minesweeper and then run Spy++. You will see a list of the running processes, find Minesweeper process and double click on it. A dialog will show you some informations about the game, one of them is the window procedure.

- Another way to locate the window procedure is to use Ida.
This is more or less the way explained by +Spath in his tutorial. Load the file into ida and press ctrl-p; a dialog appears. This dialog contains the functions used by the program and, we are interested in WinMain function, the one used to define the window class. This is useful because from here you can understand where the window procedure is. The program uses RegisterClassW function to register the window class:

```
ATOM RegisterClass(
    CONST WNDCLASS *lpWndClass    // address of structure with class
data
);
```

and:

```
typedef struct _WNDCLASS {
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
} WNDCLASS;
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

This structure contains the window attributes; we are interested in lpfnWndProc because it points to the window procedure. Look at the disasm Ida gave you and take the address we were looking for:

```
:01001472 mov [ebp+WndClass.style], edi   <-----------------------
style param
:01001475 mov [ebp+WndClass.lpfnWndProc], offset sub_100180A   <--
lpfnWndProc param, our address is 100180A
:0100147C mov [ebp+WndClass.cbClsExtra], edi   <------------------
cbClsExtra param
:0100147F mov [ebp+WndClass.cbWndExtra], edi   <------------------
cbWndExtra param
:01001482 mov [ebp+WndClass.hInstance], ecx   <-------------------
hInstance param
:01001485 mov [ebp+WndClass.hIcon], eax   <-----------------------
hIcon param
:01001488 call ds:LoadCursorW
:0100148E push ebx ; int
:0100148F mov [ebp+WndClass.hCursor], eax   <---------------------
hCursor param
:01001492 call ds:GetStockObject
:01001498 mov [ebp+WndClass.hbrBackground], eax   <---------------
hbrBackground param
:0100149B lea eax, [ebp+WndClass]
:0100149E mov esi, offset AppName
:010014A3 push eax ; lpWndClass
:010014A4 mov [ebp+WndClass.lpszMenuName], edi   <----------------
lpszMenuName param
:010014A7 mov [ebp+WndClass.lpszClassName], esi   <---------------
lpszClassName param
:010014AA call ds:RegisterClassW   <------------------------------
RegisterClassW
```

Ok, now that we know where the window procedure is, it's time to use a debugger, Ollydbg.
Load the program and jump to the window procedure, we need to set a breakpoint. Remembering that we are trying to find the part of the procedure which check whether a menu item is clicked, we need to break if and only if a menu item is clicked. When an item is clicked, a specific WM_COMMAND message is sent to the procedure. This specific message contains the menu item identifier (the one you have clicked on) in the loworder word of the wParam parameter.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Take a look at the first part of the window procedure:

```
:0100180A PUSH EBP
:0100180B MOV EBP,ESP
:0100180D SUB ESP,40
:01001810 MOV ECX,DWORD PTR SS:[EBP+C]   <-- ecx is the message
(i.e. WM_COMMAND)
:01001813 PUSH EBX
```

If we want to catch the click over one of the menu item we need to set a conditional breakpoint (Shift+F2) at 1001810. The condition I have used is:

```
[EBP+0Ch]==111H && [EBP+10H]==209H
```

where '[EBP+0Ch]==111H' checks for the WM_COMMAND message and '[EBP+10H]==209H' checks whether the 'Beginner' menu item is been clicked (you can use your preferred menu item :-)). Run the program and click on the chosen item, Olly breaks. Steps a little and you will surely find the place where the program checks which menu items is been pressed:

```
:010018B7 MOV EAX,111
:010018BC CMP ECX,EAX   <-------------------- Is WM_COMMAND
received?
:010018BE JA winmine.01001B3F
:010018C4 JE winmine.010019AD   <------------ We jump if a
WM_COMMAND has been received
    ...
:010019AD MOV ECX,DWORD PTR SS:[EBP+10]   <-- ecx is the wParam
:010019B0 MOVZX EAX,CX   <------------------- eax is the item
identifier
:010019B3 CMP EAX,20B   <-------------------- Is "Expert" item
clicked?
:010019B8 JG SHORT winmine.01001A30
:010019BA CMP EAX,209   <-------------------- Is "Beginner" item
clicked?
:010019BF JGE SHORT winmine.010019EE
```

This piece of code is what we were looking for; since we will have to check whether the new menu item has been clicked, we will change the jump at 10018C4 into a new jump. We will see later how the code will look like.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

3. Find the point where the program checks whether a WM_MOUSEMOVE has been received

This is necessary because from here we will jump to the dll function that shows a possible bomb. In order to find the place where the program checks for WM_MOUSEMOVE message we will change the conditional breakpoint; the new condition will be: [EBP+0Ch]==200H

```
:01001B3F MOV EAX,DWORD PTR SS:[EBP+C]   <-- the received message
:01001B42 MOV ESI,200   <------------------- 200h = WM_MOUSEMOVE
:01001B47 PUSH 1
:01001B49 XOR EDI,EDI
:01001B4B CMP EAX,ESI   <------------------- is WM_MOUSEMOVE
received?
:01001B4D POP EBX
:01001B4E JA SHORT winmine.01001BB4
:01001B50 JE winmine.01001DD5   <----------- yes, WM_MOUSEMOVE
received
```

Stepping a little you will see that this type of message is not so useful for the original program therefore we will change the jump at 1001B50.

4. How to modify the Minesweeper process

First of all we have to find a place where we will go to insert the new code; there are many caves inside the file, you must only choose one of them. I will add the new code starting from 10049A5h. What does this new code is supposed to do? It adds the new item menu and then it simply calls a function from our dll.
InsertMenuItem is not imported by the program and so we will have to use the combination of LoadLibrary/GetProcAddress functions. To use these two functions is pretty easy.

LoadLibrary: it's the first function to use and it maps the dll module into the address space of the calling process. The function returns a handle that can be used in GetProcAddress to get the address of a dll function.

```
HINSTANCE LoadLibrary(
   LPCTSTR lpLibFileName   // address of the string that names the
executable module
);
```

GetProcAddress: returns the address of the exported dll function.

```
FARPROC GetProcAddress(
   HMODULE hModule,    // handle to DLL module (the value returned
by LoadLibrary)
   LPCSTR lpProcName   // address of the string that names the
function
);
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Now that you know how to call InsertMenuItem we can see how this function looks like:

```
BOOL WINAPI InsertMenuItem(
   HMENU hMenu,           // Handle to the menu in which the new
menu item is inserted
   UINT uItem,            // Identifier or position of the menu item
before which to insert the new item
   BOOL fByPosition,      // If FALSE, uItem is a menu item
identifier. Otherwise, it is a menu item position
   LPMENUITEMINFO lpmii   // Pointer to a MENUITEMINFO structure
that contains information about the new menu item
);
```

and:

```
typedef struct tagMENUITEMINFO {
   UINT cbSize;             // Size of structure, in bytes
   UINT fMask;              // Members to retrieve or set
   UINT fType;              // Menu item type
   UINT fState;             // Menu item state
   UINT wID;                // Application-defined 16-bit value that
identifies the menu item
   HMENU hSubMenu;          // Handle to the drop-down menu or
submenu associated with the menu item
   HBITMAP hbmpChecked;     // Handle to the bitmap to display next
to the item if it is checked
   HBITMAP hbmpUnchecked;   // Handle to the bitmap to display next
to the item if it is not checked
   DWORD dwItemData;        // Application-defined value associated
with the menu item
   LPTSTR dwTypeData;       // Content of the menu item
   UINT cch;                // Length of the menu item text
} MENUITEMINFO, FAR *LPMENUITEMINFO;
```

Even if there are alot of parameters, the use of this function is pretty easy. This is the code you have to inject into the Minesweeper program:

```
:010049A5 ; Initial address of the cave I chose
:010049A5 dword_10049A5   dd 0                  ; specialMode=08h if
specialMode is enable otherwise specialMode=00h
:010049A6 aZaiwinmine_dll db 'zaiWinmine.dll',0 ; The name of the
new DLL
:010049B5 aUser32_dll     db 'User32.dll',0     ; InsertMenuItem is
inside USer32.dll
; The name of the 2 exported functions:
:010049C0 aChangeitemstat db 'changeItemStatus',0 ; Changes the
status of the itemmenu
:010049D1 aRevealcell     db 'revealCell',0       ; Reveals a bomb
under a cell
:010049DC dword_10049DC   dd 0                    ; Stores the
address of changeItemStatus function
:010049E0 dword_10049E0   dd 0                    ; Stores the
address of revealCell function
:010049E4 aInsertmenuitem db 'InsertMenuItemA',0
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
:010049F4 aSpecialMode    db 'Special Mode',0     ; Text to be place
in the new menu item
; MENUITEMINFO structure:
:01004A01 dd 2Ch          ; Size of structure: 11 dwords = 2Ch bytes
:01004A05 dd 42h          ; MIIM_ID + MIIM_STRING
:01004A09 dd 0            ; MFT_STRING, displays the menu item using a
text string
:01004A0D dd 0            ; Default state
:01004A11 dd 212h         ; Id of the new item
:01004A15 dd 0            ; NULL because the item does not open a drop-
down menu or submenu
:01004A19 dd 0            ; NULL, default check mark
:01004A1D dd 0            ; NULL
:01004A21 dd 0            ; NULL
:01004A25 dd 10049F4h     ; The pointer to a null-terminated string that
defines the text of the new item
:01004A29 dd 4            ; The length of the string

:01004A2D ; From initialization (10014E8): we add the new item and
save the address of changeItemStatus and revealCell
:01004A2D push offset loc_10014ED     ; Address from where to return
to
:01004A32 pusha
:01004A33 push offset aUser32_dll     ; "User32.dll"
:01004A38 call ds:LoadLibraryA        ; Maps User32.dll
:01004A3E push offset aInsertmenuitem ; "InsertMenuItemA"
:01004A43 push eax                    ; Handle to dll module
:01004A44 call ds:GetProcAddress      ; Get the address of the
function
:01004A4A mov ebx, offset hMenu
:01004A4F push offset dword_1004A01   ; Pointer to a MENUITEMINFO
structure
:01004A54 push 0                      ; FALSE
:01004A56 push 0                      ; The position of the item
:01004A58 push dword ptr [ebx]        ; Handle to the menu
:01004A5A call eax                    ; call InsertMenuItem
:01004A5C push offset aZaiwinmine_dll ; "zaiWinmine.dll", the name
of the dll
:01004A61 call ds:LoadLibraryA
:01004A67 mov esi, eax
:01004A69 push offset aChangeitemstat ; "changeItemStatus"
:01004A6E push esi
:01004A6F call ds:GetProcAddress
:01004A75 mov ds:dword_10049DC, eax   ; Stores the address of
changeItemStatus
:01004A7A push offset aRevealcell     ; "revealCell"
:01004A7F push esi
:01004A80 call ds:GetProcAddress
:01004A86 mov ds:dword_10049E0, eax   ; Stores the address of
revealCell
:01004A8B popa
:01004A8C mov eax, dword_10052A4
:01004A91 retn                        ; Jump to the original code

:01004A92 ; From 'specialMode' item clicked (10018C4): we call
changeItemStatus
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
:01004A92 push offset loc_10019AD     ; Address from where to return
to
:01004A97 pusha
:01004A98 call ds:dword_10049DC       ; call changeItemStatus
:01004A9E jmp short loc_1004AB6

:01004AA0 ; From WM_MOUSEMOVE event (1001B50): we call revealCell
:01004AA0 push offset loc_1001DD5     ; Address from where to return
to
:01004AA5 cmp ds:byte_10049A5, 0      ; Is the special mode enable?
:01004AAC jnz short loc_1004AAF       ; Yes, jump down
:01004AAE retn                        ; specialMode is disable, no
need to show the bomb, jump back to the original code
:01004AAF pusha
:01004AB0 call ds:dword_10049E0       ; call revealCell
:01004AB6 popa
:01004AB7 retn                        ; Jump back to the original
code
```

Now, we have to modify the instructions at 10014E8, 10018C4 and at 1001B50 because they have to jump to the new code:

```
:010014E8 jmp loc_1004A2D
:010018C4 jz loc_1004A92
:01001B50 jz loc_1004AA0
```

4.1. DLL functions

We have seen all the modifications we will have to add to the program; from now on, we will see how to write the 2 dll functions:
- changeItemStatus, this function will check/uncheck the menuitem enabling or disabling the special mode.
- revealCell, this function reveals a bomb under a cell.

4.2. changeItemStatus

To check/uncheck the item I will use CheckMenuItem function:

```
DWORD                                                CheckMenuItem(
      HMENU     hmenu,              //     handle     to     menu
    UINT  uIDCheckItem,    //  menu  item  to  check  or  uncheck
      UINT     uCheck              //     menu     item     flags
);
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

An easy function to use. It needs:

1. the handle of the menu. We don't have the handle but since the function is imported and since some of the items of the menu could be checked/unchecked, the easiest way to retrieve the handle is to put a breakpoint on this function and hit one of the checked items, for example the 'sound' item. Perfect, Olly will break showing us where the handle is stored.
2. the id of the menu. It's the number we gave to our new item.
3. we are interested in two values:
MF_CHECKED (08h): sets the check-mark attribute to the checked state.
MF_UNCHECKED (00h): sets the check-mark attribute to the unchecked state.

I did write the dll in asm but no one will stop you from writing it in other languages. Here is the first function:

```
; Change the status of the new item
changeItemStatus proc
   .IF word ptr [ebp+10h] == 212h   ; 212h is the id I used for the
new item
      mov ebx, 10049A2h             ; byte [10049A2] stores the
state of the menu item
      xor byte ptr [ebx], 08h
      movzx eax, byte ptr [ebx]
      mov ebx, 10052BCh             ; address where the menu handle
is stored
      invoke CheckMenuItem, dword ptr [ebx], 212h, eax
   .ENDIF
   ret
changeItemStatus endp
```

The only thing that requires an explanation is the value stored at 10049A2. This byte is:

00h if the item is unchecked

08h if the item is checked

Why not 00h and 01h? That is because I use this address for store both the state of the item and the menu item flag.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

4.3. revealCell

Before writing the function we must understand where the bombs are stored.
We need to find a way to break when a cell is clicked. Exploiting the work we have done before, I will use another conditional breakpoint at 1001810; the idea is to break when the left mouse button is pressed (and then released). When this button is released, a WM_LBUTTONUP (202h) message is posted and so, the new condition will be: [ebp+0Ch]==202h

Set the new breakpoint and click on a cell; Olly breaks, steps a little until Olly will tell you that you have found the place:

```
01001C30 XOR EDI,EDI                                    ; Cases 202
(WM_LBUTTONUP) <-- thx to Olly :-D
01001C32 CMP DWORD PTR DS:[1005160],EDI
01001C38 JE winmine.01001D4C
01001C3E MOV DWORD PTR DS:[1005160],EDI
01001C44 CALL DWORD PTR DS:[<&USER32.ReleaseCaptu> ; ReleaseCapture
01001C4A TEST BYTE PTR DS:[1005010],BL
01001C50 JE SHORT winmine.01001C5C
01001C52 CALL winmine.0100373E                          ; hmmm...
01001C57 JMP winmine.01001D4C                           ; jump at the end
of the procedure
```

This is the code that manages the WM_LBUTTONUP command; as you can see there is a call to ReleaseCapture, a jump at the end of the procedure and a suspicious call. Maybe it's the call that will show us where the bombs are. Enter the call and take a glance at it:

```
010037F2 MOV EDX,ECX   ; ecx stores the number of the row of the
clicked cell, eax stores the column
010037F4 SHL EDX,5
010037F7 MOV DL,BYTE PTR DS:[EDX+EAX+1005700]   ; dl = value stores
in the clicked cell
```

As you can see the first row is stored starting from 1005721 and it ends with the character with double value 0x10. Doing some tryes you will sooner understand which is the value that identify the bomb, it's 0x8F.
Now that we know almost everything about the program, we have to show these damned bombs. I will try to explain you how to show them when the mouse pass over a cell; the bombs will remain showed until the end of the game. We can write the other function of the dll, the one I called revealCell.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

What this function is supposed to do? Remembering that the function will be called on every single movement of the mouse (and let's suppose specialMode is enable), we have this scheme:

1. Is the pointer inside the grid?
No: quit from the function
Yes: jump to 2

2. Is the cell hiding a bomb?
No: quit from the function
Yes: reveal the bomb

The point number 1 is necessary because the bombs can not be outside the grid while the point number 2 is the final check. The first cell (upper left corner) starts from coordinates 0x0C,0x37 and each cell is 0x10,0x10. Now that we know this information, we are able to implement the point number 1: a simple 'if' based on the coordinates of the point where the mouse has been pressed. How did you know the x and the y coordinates where the mouse has been pressed? WM_MOUSEMOVE will tell you; infact:

```
WM_MOUSEMOVE
   fwKeys = wParam;          // key flags
   xPos = LOWORD(lParam);    // horizontal position of cursor
   yPos = HIWORD(lParam);    // vertical position of cursor
```

Here is all that you need :-D

Point number 2: first of all, we have to check whether a cell hides a bomb; this is not a problem because we know where the program stores the grid in memory and, we will easily perform this check but... how can I reveal a bomb under a cell? The answer is inside Graphics Device Interface (GDI) library; in this specific case we will use GDI to display a bitmap. My new version of the game displays the image that has the bomb on the red background (open the program with a resource editor, you will find the image I am talking about under "Bitmap-410-1033"). You can display the image you prefer taken from the proggie or created by you. To display this image I use these functions: GetDC, BitBlt, ReleaseDC.
GetDC: retrieves a handle of a display Device Context (DC) for the client area of the specified window.

```
HDC GetDC(
   HWND hWnd    // handle of window
);
```

This is the first function we have to call and it takes a single parameter. How do we know the handle of the window? Simple, these 3 functions are used by MineSweeper and so, the best way to understand how to use them is to bpx them and to look how they are used. This time, put a bpx on GetDC and run again the program. Ollydbg will surely break and the only thing you have to do is to take note of the address that contains the handle...
BitBlt: it clips an image from a source DC to a destination DC.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
BOOL BitBlt(
   HDC hdcDest,    // handle to destination DC, the GetDC returned
address
   int nXDest,     // x-coordinate of destination rectangle's upper-
left corner
   int nYDest,     // y-coordinate of destination rectangle's upper-
left corner
   int nWidth,     // width of destination rectangle
   int nHeight,    // height of destination rectangle
   HDC hdcSrc,     // handle to source device context
   int nXSrc,      // x-coordinate of source rectangle's upper-left
corner
   int nYSrc,      // y-coordinate of source rectangle's upper-left
corner
   DWORD dwRop     // raster operation code
);
```

Easy. As before, the best way for to understand how to use the function is to bpx it on Ollydbg. If you want to display another image you have to change the 6° parameter: hdcSrc.

ReleaseDC: the ReleaseDC function releases a device context

```
int ReleaseDC(
   HWND hWnd,    // handle of window
   HDC hDC       // handle of device context
);
```

Ok, it's time to past the source of the revealCell procedure:

```
.data
oldRow    db "0100499D"
oldColumn db "010049A1"

; Display the image under a cell
revealCell proc
   pushad
   ; Is the mouse pointer inside the grid ?
   .IF word ptr [ebp+14h] < 0Ch || word ptr [ebp+16h] < 37h
      jmp @exitRevealCell
   .endif

   xor eax, eax
   mov ax, word ptr [ebp+14h]   ; The x-coord of the mouse
   sub eax, 0Ch
   shr eax, 4
   add eax, 1                   ; The number of the column -inside
the grid- where the mouse is pressed
   xor ebx, ebx
   mov bx, word ptr [ebp+16h]   ; The y-coord of the mouse
   sub ebx, 37h
   shr ebx, 4
   add ebx, 1                   ; The number of the row where the
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
mouse is pressed

   mov edi, 100499Dh
   .IF ax == word ptr [edi] && bx == word ptr [edi+4]
      ; It's in the same cell than before, no need to display the
same image again
      jmp @exitRevealCell
   .ENDIF

   mov ecx, ebx
   mov edx, ecx
   shl edx, 5
   mov ebx, 1005700h
   add ebx, eax
   add ebx,edx
   ; Reveal the cell if and only if there is an hidden bomb
   .IF byte ptr [ebx] == 8Fh
      mov edi, 100499Dh
      mov dword ptr [edi], eax     ; Save the column
      mov dword ptr [edi+4], ecx   ; Save the row
      push eax
      push ecx
      mov esi, 10052A8h            ; Pointer to handle of window
      push [esi]                   ; Handle of window
      call GetDC
      mov esi, eax
      pop ecx
      pop eax
      shl ecx, 4
      add ecx, 27h
      shl eax, 4
      sub eax, 4
      mov ebx, 1005AE0h
      push 0CC0020h   ; SRCCOPY flag
      push 0
      push 0
      push [ebx+48]   ; Source DC, the image to display
      push 10h        ; Height of the rectangle
      push 10h        ; Width of the rectangle
      push ecx        ; y coord
      push eax        ; x coord
      push esi        ; Handle to destination DC
      call BitBlt
      mov ebx, 10052A8h
      push esi                ; Handle of DC
      push dword ptr [ebx]    ; Handle of window
      call ReleaseDC
   .ELSE
      mov dword ptr [edi], 0
      mov dword ptr [edi+4], 0
   .ENDIF

   @exitRevealCell:
   popad
   ret
revealCell endp
```

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

5. Postamble: conclusion.

The tutorial ends here, you only have to write a loader which will load the program and add the new functionality in memory. I will not tell how to write a simple loader, you will find all the sources inside the attached file.

Playing with the new version of the game I have discovered a weird fact; set the special mode feature and make the first click over a bomb. Nothing happens, the bomb disappears; is there something wrong in our work? Eheh, no. Where the bomb has gone? (Hint: it is a specific characteristic of the program.. :-))

6. Greetings and thanks.

I would like to thank all the friends at RCE messageboard and all the UIC members. As usual, feel free to mail me for comments, suggestions, bug reports and/or whatever you want...

7. References:

http://www.woodmann.net/fravia/menusspa.htm
http://www.woodmann.net/fravia/TracePlus_MenuPatch.html
http://www.woodmann.net/fravia/kayaker_RegmonPlus.htm
http://www.codeguru.com/mfc/comments/50642.shtml

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.