REVERSE ENGINEERING AND EXPLOITATION

Session 3

Questions

To answer these questions you will need to use the *Reverse Engineering Toolkit* virtual machine.

1. Load *winmine.exe* into *ImmunityDebugger* and allow it to run unhindered.

    a. With the games grid setup as a 9 by 9 grid with a maximum number of mines (this is typically 64!), play the game until a mine explodes and then use *ImmunityDebugger* to pause *winmine's* executing state.

        i. What function stack frames can you discover?

        ii. Examine the code that uses the stack frames you have discovered. Can you locate the prologue and epilogue code? Can you verify the function calling convention being applied here?

    b. Use *ImmunityDebugger* to restart *winmine.exe* and follow these instructions:

        - Using *SysInternal's procdump*, dump a copy of *winmine's* initial process state.
        - Now allow *winmine.exe* to run until the grid is drawn and then pause it. Using *procdump*, dump a copy of *winmine* with its grid now drawn.

        Now use *nwDiff* to compare your two process dumps and, by examining *winmine's* data pages, see if you can locate the area of memory where the playing grid is stored?

2. Using PyDBG[1], can you write code to print out *winmine's* playing grid on the console?

3. Using PyDBG, place a memory breakpoint on *winmine's* playing grid and so print out its playing grid every time the grid is accessed (**Note:** if you knew about hooking, we could now write code to print out the playing grid every time it's initialized!).

[1] **Note:** you will find it useful to look at the PyDBG API located at:
http://pedram.redhive.com/PaiMei/docs/PyDbg/